



DSpace System Documentation

Mark Diggory

Version: 5
Date: 12/31/09
Creator: Mark Diggory



Table of Contents

1. Introduction	4
2. Functional Overview	4
2.1. Data Model	5
2.2. Plugin Manager	7
2.3. Metadata	7
2.4. Packager Plugins	8
2.5. Crosswalk Plugins	8
2.6. E-People and Groups	8
2.7. Authentication	9
2.8. Authorization	9
2.9. Ingest Process and Workflow	10
2.10. Supervision and Collaboration	12
2.11. Handles	12
2.12. Bitstream 'Persistent' Identifiers	13
2.13. Storage Resource Broker (SRB) Support	14
2.14. Search and Browse	14
2.15. HTML Support	14
2.16. OAI Support	15
2.17. OpenURL Support	15
2.18. Creative Commons Support	15
2.19. Subscriptions	16
2.20. Import and Export	16
2.21. Registration	16
2.22. Statistics	16
2.23. Checksum Checker	17
2.24. Usage Instrumentation	17
3. Installation	17
3.1. For the Impatient	17
3.2. Prerequisite Software	17
3.3. Installation Options	19
3.4. Advanced Installation	23
3.5. Windows Installation	29
3.6. Checking Your Installation	30
3.7. Known Bugs	30
3.8. Common Problems	30
4. Upgrading a DSpace Installation	32
4.1. Upgrading from 1.5.x to 1.6	32
4.2. Upgrading From 1.5 or 1.5.1 to 1.5.2	40
4.3. Upgrading From 1.4.2 to 1.5	46
4.4. Upgrading From 1.4.1 to 1.4.2	50
4.5. Upgrading From 1.4 to 1.4.x	50
4.6. Upgrading From 1.3.2 to 1.4.x	53
4.7. Upgrading From 1.3.1 to 1.3.2	55
4.8. Upgrading From 1.2.x to 1.3.x	56
4.9. Upgrading From 1.2.1 to 1.2.2	57
4.10. Upgrading From 1.2 to 1.2.1	58
4.11. Upgrading From 1.1 (or 1.1.1) to 1.2	59
4.12. Upgrading From 1.1 to 1.1.1	62
4.13. Upgrading From 1.0.1 to 1.1	62
5. Configuration and Customization	64
5.1. Input Conventions	64
5.2. Update Reminder	65
5.3. The dspace.cfg Configuration Properties File	65
5.4. Optional or Advanced Configuration Settings	123
5.5. DSpace Services Framework	135



5.6. DSpace Statistics	139
5.7. JSPUI Configuration and Customization	141
5.8. XMLUI Configuration and Customization	142
6. System Administration	148
6.1. Community and Collection Structure Importer	148
6.2. Package Importer and Exporter	149
6.3. Item Importer and Exporter	150
6.4. Transferring Items Between DSpace Instances	154
6.5. Item Update	154
6.6. Registering (Not Importing) Bitstreams	156
6.7. METS Tools	158
6.8. MediaFilters: Transforming DSpace Content	160
6.9. Sub-Community Management	161
6.10. Batch Metadata Editing	162
6.11. Checksum Checker	165
7. Storage	170
7.1. RDBMS	170
7.2. Bitstream Store	172
8. Directories	175
8.1. Overview	175
8.2. Source Directory Layout	176
8.3. Installed Directory Layout	177
8.4. Contents of JSPUI Web Application	178
8.5. Contents of XMLUI Web Application (aka Manakin)	178
8.6. Log Files	179
9. Architecture	180
9.1. Overview	180
10. Application	182
10.1. Web User Interface	182
10.2. OAI-PMH Data Provider	189
10.3. DSpace Command Launcher	192
11. Business	193
11.1. Core Classes	193
11.2. Content Management API	195
11.3. Plugin Manager	199
11.4. Workflow System	207
11.5. Administration Toolkit	207
11.6. E-person/Group Manager	208
11.7. Authorization	208
11.8. Handle Manager/Handle Plugin	209
11.9. Search	210
11.10. Browse API	211
11.11. Checksum checker	214
11.12. OpenSearch Support	214
11.13. Embargo	215
12. Submission	215
12.1. Understanding the Submission Configuration File	215
12.2. Reordering/Removing Submission Steps	218
12.3. Assigning a custom Submission Process to a Collection	218
12.4. Custom Metadata-entry Pages for Submission	219
12.5. Configuring the File Upload step	224
12.6. Creating new Submission Steps	224
13. Appendices	225
13.1. Appendix	225
13.2. DRI Schema Reference	230
13.3. History	262



1. Introduction

DSpace is an open source software platform that enables organisations to:

- capture and describe digital material using a submission workflow module, or a variety of programmatic ingest options
- distribute an organisation's digital assets over the web through a search and retrieval system
- preserve digital assets over the long term

This system documentation includes a functional overview of the system, which is a good introduction to the capabilities of the system, and should be readable by non-technical folk. Everyone should read this section first because it introduces some terminology used throughout the rest of the documentation.

For people actually running a DSpace service, there is an installation guide, and sections on configuration and the directory structure. Note that as of DSpace 1.2, the administration user interface guide is now on-line help available from within the DSpace system.

Finally, for those interested in the details of how DSpace works, and those potentially interested in modifying the code for their own purposes, there is a detailed architecture and design section.

Other good sources of information are:

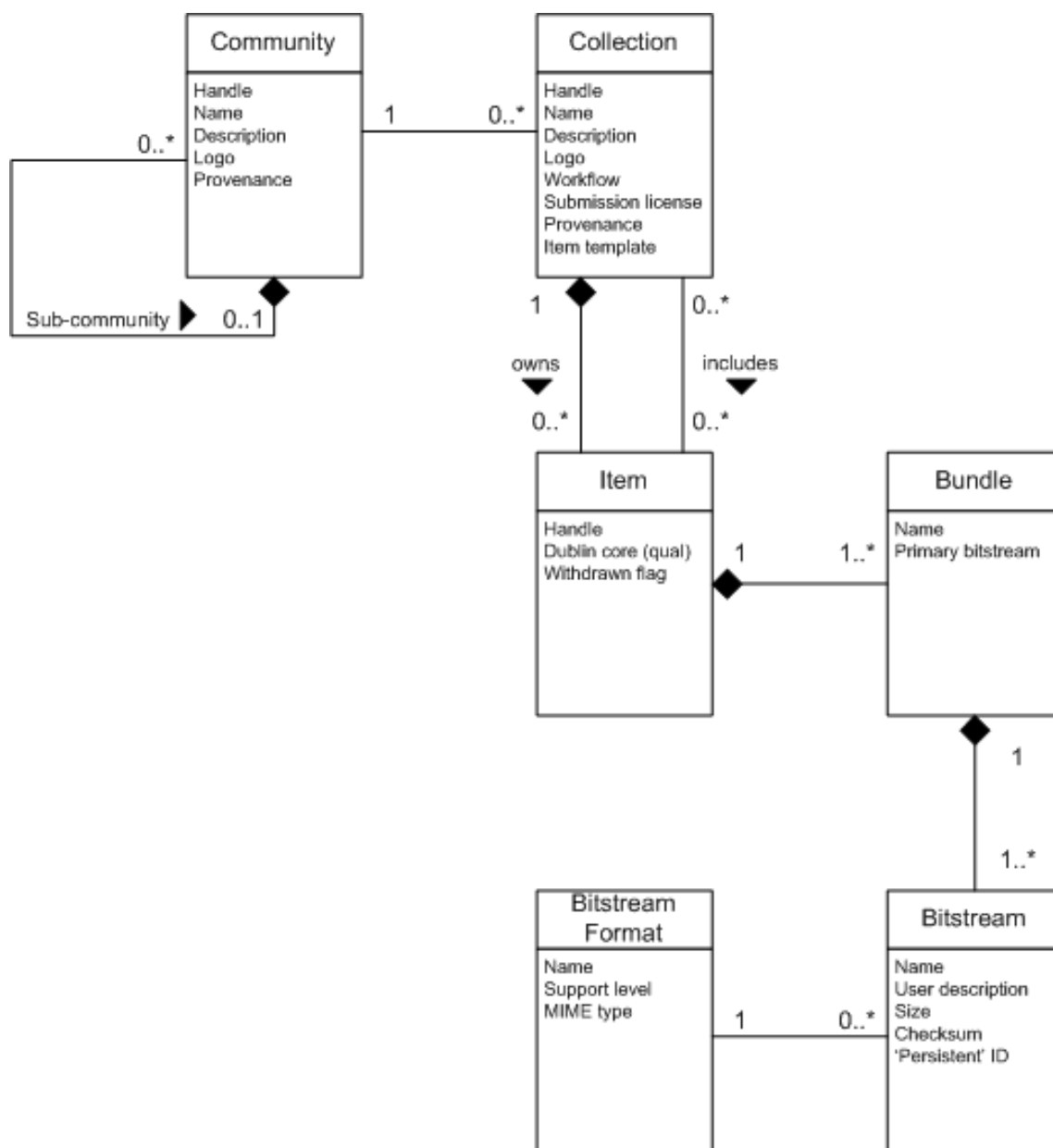
- The DSpace Public API Javadocs. Build these with the command `mvn javadoc:javadoc`.
- The <http://wiki.dspace.org/> contains stacks of useful information about the DSpace platform and the work people are doing with it. You are strongly encouraged to visit this site and add information about your own work. Useful Wiki areas are:
 - <http://wiki.dspace.org/DspaceResources> (Web sites, mailing lists etc.)
 - <http://wiki.dspace.org/TechnicalFaq>
 - <http://wiki.dspace.org/DspaceProjects>
 - <http://wiki.dspace.org/ContributionGuidelines>
- <http://www.dspace.org/> has announcements and contains useful information about bringing up an instance of DSpace at your organization.
- The # is the recommended place to ask questions, since a growing community of DSpace developers and users is on hand on that list to help with any questions you might have. The e-mail archive of that list is a useful resource.
- The #, for those developing with the DSpace with a view to contributing to the core DSpace code.

2. Functional Overview

The following sections describe the various functional aspects of the DSpace system.



2.1. Data Model



Data Model Diagram

The way data is organized in DSpace is intended to reflect the structure of the organization using the DSpace system. Each DSpace site is divided into *communities*, which can be further divided into *sub-communities* reflecting the typical university structure of college, department, research center, or laboratory.

Communities contain *collections*, which are groupings of related content. A collection may appear in more than one community.

Each collection is composed of *items*, which are the basic archival elements of the archive. Each item is owned by one collection. Additionally, an item may appear in additional collections; however every item has one and only one owning collection.

Items are further subdivided into named *bundles* of *bitstreams*. Bitstreams are, as the name suggests, streams of bits, usually ordinary computer files. Bitstreams that are somehow closely related, for example HTML files and images that compose a single HTML document, are organised into bundles.



In practice, most items tend to have these named bundles:

- *ORIGINAL* – the bundle with the original, deposited bitstreams
- *THUMBNAILS* – thumbnails of any image bitstreams
- *TEXT* – extracted full-text from bitstreams in *ORIGINAL*, for indexing
- *LICENSE* – contains the deposit license that the submitter granted the host organization; in other words, specifies the rights that the hosting organization have
- *CC_LICENSE* – contains the distribution license, if any (a <http://www.creativecommons.org> license) associated with the item. This license specifies what end users downloading the content can do with the content

Each bitstream is associated with one *Bitstream Format*. Because preservation services may be an important aspect of the DSpace service, it is important to capture the specific formats of files that users submit. In DSpace, a bitstream format is a unique and consistent way to refer to a particular file format. An integral part of a bitstream format is an either implicit or explicit notion of how material in that format can be interpreted. For example, the interpretation for bitstreams encoded in the JPEG standard for still image compression is defined explicitly in the Standard ISO/IEC 10918-1. The interpretation of bitstreams in Microsoft Word 2000 format is defined implicitly, through reference to the Microsoft Word 2000 application. Bitstream formats can be more specific than MIME types or file suffixes. For example, *application/ms-word* and *.doc* span multiple versions of the Microsoft Word application, each of which produces bitstreams with presumably different characteristics.

Each bitstream format additionally has a *support level*, indicating how well the hosting institution is likely to be able to preserve content in the format in the future. There are three possible support levels that bitstream formats may be assigned by the hosting institution. The host institution should determine the exact meaning of each support level, after careful consideration of costs and requirements. MIT Libraries' interpretation is shown below:

Supported	The format is recognized, and the hosting institution is confident it can make bitstreams of this format useable in the future, using whatever combination of techniques (such as migration, emulation, etc.) is appropriate given the context of need.
Known	The format is recognized, and the hosting institution will promise to preserve the bitstream as-is, and allow it to be retrieved. The hosting institution will attempt to obtain enough information to enable the format to be upgraded to the 'supported' level.
Unsupported	The format is unrecognized, but the hosting institution will undertake to preserve the bitstream as-is and allow it to be retrieved.

Each item has one qualified Dublin Core metadata record. Other metadata might be stored in an item as a serialized bitstream, but we store Dublin Core for every item for interoperability and ease of discovery. The Dublin Core may be entered by end-users as they submit content, or it might be derived from other metadata as part of an ingest process.

Items can be removed from DSpace in one of two ways: They may be 'withdrawn', which means they remain in the archive but are completely hidden from view. In this case, if an end-user attempts to access the withdrawn item, they are presented with a 'tombstone,' that indicates the item has been removed. For whatever reason, an item may also be 'expunged' if necessary, in which case all traces of it are removed from the archive.

Object	Example
--------	---------



Community	Laboratory of Computer Science; Oceanographic Research Center
Collection	LCS Technical Reports; ORC Statistical Data Sets
Item	A technical report; a data set with accompanying description; a video recording of a lecture
Bundle	A group of HTML and image bitstreams making up an HTML document
Bitstream	A single HTML file; a single image file; a source code file
Bitstream Format	Microsoft Word version 6.0; JPEG encoded image format

2.2. Plugin Manager

The PluginManager is a very simple component container. It creates and organizes components (plugins), and helps select a plugin in the cases where there are many possible choices. It also gives some limited control over the lifecycle of a plugin.

A plugin is defined by a Java interface. The consumer of a plugin asks for its plugin by interface. A Plugin is an instance of any class that implements the plugin interface. It is interchangeable with other implementations, so that any of them may be "plugged in".

The mediafilter is a simple example of a plugin implementation. Refer to the Business Logic Layer for more details on Plugins.

2.3. Metadata

Broadly speaking, DSpace holds three sorts of metadata about archived content:

- **Descriptive Metadata:** DSpace can support multiple flat metadata schemas for describing an item. A qualified Dublin Core metadata schema loosely based on the <http://www.dublincore.org/documents/library-application-profile/> set of elements and qualifiers is provided by default. The <http://dspace.org/technology/metadata.html> comes pre-configured with the DSpace source code. However, you can configure multiple schemas and select metadata fields from a mix of configured schemas to describe your items. Other descriptive metadata about items (e.g. metadata described in a hierarchical schema) may be held in serialized bitstreams. *Communities* and *collections* have some simple descriptive metadata (a name, and some descriptive prose), held in the DBMS.
- **Administrative Metadata:** This includes preservation metadata, provenance and authorization policy data. Most of this is held within DSpace's relation DBMS schema. Provenance metadata (prose) is stored in Dublin Core records. Additionally, some other administrative metadata (for example, bitstream byte sizes and MIME types) is replicated in Dublin Core records so that it is easily accessible outside of DSpace.
- **Structural Metadata:** This includes information about how to present an item, or bitstreams within an item, to an end-user, and the relationships between constituent parts of the item. As an example, consider a thesis consisting of a number of TIFF images, each depicting a single page of the thesis. Structural metadata would include the fact that each image is a single page, and the ordering of the TIFF images/pages. Structural metadata in DSpace is currently fairly basic; within an item, bitstreams can be arranged into separate bundles as described above. A bundle may also optionally have a *primary bitstream*. This is currently used by the HTML support to indicate which bitstream in the bundle is the first HTML file to send to a browser. In addition to some basic technical metadata, bitstreams also have a 'sequence ID' that uniquely identifies it within an item. This is used to produce a 'persistent' bitstream identifier for each bitstream. Additional structural metadata can be stored in serialized bitstreams, but DSpace does not currently understand this natively.



2.4. Packager Plugins

Packagers are software modules that translate between DSpace Item objects and a self-contained external representation, or "package". A *Package Ingestor* interprets, or *ingests*, the package and creates an Item. A *Package Disseminator* writes out the contents of an Item in the package format.

A package is typically an archive file such as a Zip or "tar" file, including a *manifest* document which contains metadata and a description of the package contents. The <http://www.imsi.org/content/packaging/> is a typical packaging standard. A package might also be a single document or media file that contains its own metadata, such as a PDF document with embedded descriptive metadata.

Package ingesters and package disseminators are each a type of named plugin (see Plugin Manager), so it is easy to add new packagers specific to the needs of your site. You do not have to supply both an ingester and disseminator for each format; it is perfectly acceptable to just implement one of them.

Most packager plugins call upon Crosswalk plugins to translate the metadata between DSpace's object model and the package format.

2.5. Crosswalk Plugins

Crosswalks are software modules that translate between DSpace object metadata and a specific external representation. An *Ingestion Crosswalk* interprets the external format and crosswalks it to DSpace's internal data structure, while a *Dissemination Crosswalk* does the opposite.

For example, a MODS ingestion crosswalk translates descriptive metadata from the MODS format to the metadata fields on a DSpace Item. A MODS dissemination crosswalk generates a MODS document from the metadata on a DSpace Item.

Crosswalk plugins are named plugins (see Plugin Manager), so it is easy to add new crosswalks. You do not have to supply both an ingester and disseminator for each format; it is perfectly acceptable to just implement one of them.

There is also a special pair of crosswalk plugins which use XSL stylesheets to translate the external metadata to or from an internal DSpace format. You can add and modify XSLT crosswalks simply by editing the DSpace configuration and the stylesheets, which are stored in files in the DSpace installation directory.

The Packager plugins and OAH-PMH server make use of crosswalk plugins.

2.6. E-People and Groups

Although many of DSpace's functions such as document discovery and retrieval can be used anonymously, some features (and perhaps some documents) are only available to certain "privileged" users. E-People and Groups are the way DSpace identifies application users for the purpose of granting privileges. This identity is bound to a session of a DSpace application such as the Web UI or one of the command-line batch programs. Both E-People and Groups are granted privileges by the authorization system described below.

2.6.1. E-Person

DSpace hold the following information about each e-person:

- E-mail address
- First and last names
- Whether the user is able to log in to the system via the Web UI, and whether they must use an X509 certificate to do so;



- A password (encrypted), if appropriate
- A list of collections for which the e-person wishes to be notified of new items
- Whether the e-person 'self-registered' with the system; that is, whether the system created the e-person record automatically as a result of the end-user independently registering with the system, as opposed to the e-person record being generated from the institution's personnel database, for example.
- The network ID for the corresponding LDAP record

2.6.2. Groups

Groups are another kind of entity that can be granted permissions in the authorization system. A group is usually an explicit list of E-People; anyone identified as one of those E-People also gains the privileges granted to the group.

However, an application session can be assigned membership in a group *without* being identified as an E-Person. For example, some sites use this feature to identify users of a local network so they can read restricted materials not open to the whole world. Sessions originating from the local network are given membership in the "LocalUsers" group and gain the corresponding privileges.

Administrators can also use groups as "roles" to manage the granting of privileges more efficiently.

2.7. Authentication

Authentication is when an application session positively identifies itself as belonging to an E-Person and/or Group. In DSpace 1.4, it is implemented by a mechanism called *Stackable Authentication*: the DSpace configuration declares a "stack" of authentication methods. An application (like the Web UI) calls on the Authentication Manager, which tries each of these methods in turn to identify the E-Person to which the session belongs, as well as any extra Groups. The E-Person authentication methods are tried in turn until one succeeds. Every authenticator in the stack is given a chance to assign extra Groups. This mechanism offers the following advantages:

- Separates authentication from the Web user interface so the same authentication methods are used for other applications such as non-interactive Web Services
- Improved modularity: The authentication methods are all independent of each other. Custom authentication methods can be "stacked" on top of the default DSpace username/password method.
- Cleaner support for "implicit" authentication where username is found in the environment of a Web request, e.g. in an X.509 client certificate.

2.8. Authorization

DSpace's authorization system is based on associating actions with objects and the lists of EPeople who can perform them. The associations are called Resource Policies, and the lists of EPeople are called Groups. There are two special groups: 'Administrators', who can do anything in a site, and 'Anonymous', which is a list that contains all users. Assigning a policy for an action on an object to anonymous means giving everyone permission to do that action. (For example, most objects in DSpace sites have a policy of 'anonymous' READ.) Permissions must be explicit - lack of an explicit permission results in the default policy of 'deny'. Permissions also do not 'commute'; for example, if an e-person has READ permission on an item, they might not necessarily have READ permission on the bundles and bitstreams in that item. Currently Collections, Communities and Items are discoverable in the browse and search systems regardless of READ authorization.

The following actions are possible:



Collection

ADD/REMOVE	add or remove items (ADD = permission to submit items)
DEFAULT_ITEM_READ	inherited as READ by all submitted items
DEFAULT_BITSTREAM_READ	inherited as READ by Bitstreams of all submitted items. Note: only affects Bitstreams of an item at the time it is initially submitted. If a Bitstream is added later, it does <i>not</i> get the same default read policy.
COLLECTION_ADMIN	collection admins can edit items in a collection, withdraw items, map other items into this collection.

Table 1. Item

ADD/REMOVE	add or remove bundles
READ	can view item (item metadata is always viewable)
WRITE	can modify item

Table 2. Bundle

ADD/REMOVE	add or remove bitstreams to a bundle
------------	--------------------------------------

Table 3. Bitstream

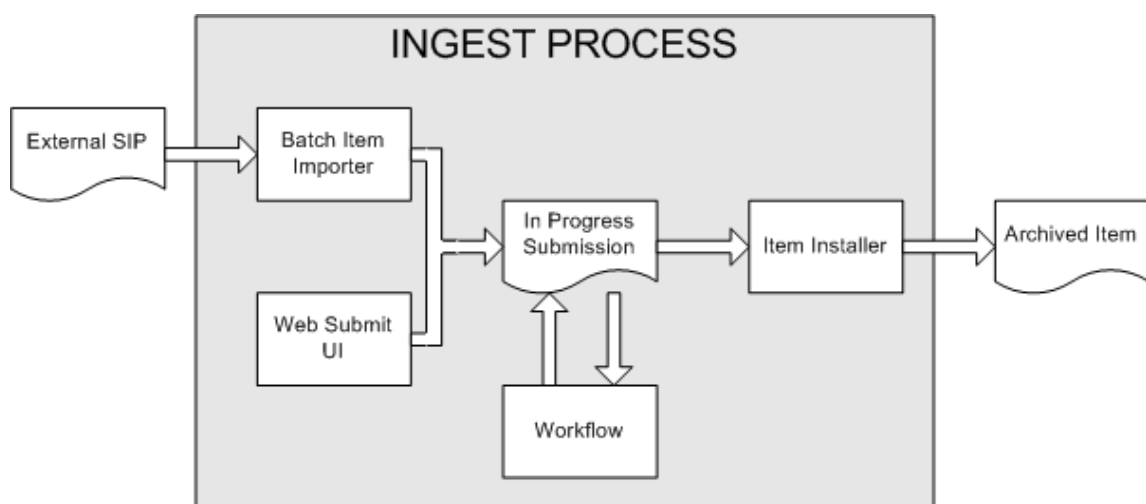
READ	view bitstream
WRITE	modify bitstream

Note that there is no 'DELETE' action. In order to 'delete' an object (e.g. an item) from the archive, one must have REMOVE permission on all objects (in this case, collection) that contain it. The 'orphaned' item is automatically deleted.

Policies can apply to individual e-people or groups of e-people.

2.9. Ingest Process and Workflow

Rather than being a single subsystem, ingesting is a process that spans several. Below is a simple illustration of the current ingesting process in DSpace.



DSpace Ingest Process



The batch item importer is an application, which turns an external SIP (an XML metadata document with some content files) into an "in progress submission" object. The Web submission UI is similarly used by an end-user to assemble an "in progress submission" object.

Depending on the policy of the collection to which the submission is targeted, a workflow process may be started. This typically allows one or more human reviewers or 'gatekeepers' to check over the submission and ensure it is suitable for inclusion in the collection.

When the Batch Ingester or Web Submit UI completes the InProgressSubmission object, and invokes the next stage of ingest (be that workflow or item installation), a provenance message is added to the Dublin Core which includes the filenames and checksums of the content of the submission. Likewise, each time a workflow changes state (e.g. a reviewer accepts the submission), a similar provenance statement is added. This allows us to track how the item has changed since a user submitted it.

Once any workflow process is successfully and positively completed, the InProgressSubmission object is consumed by an "item installer", that converts the InProgressSubmission into a fully blown archived item in DSpace. The item installer:

- Assigns an accession date
- Adds a "date.available" value to the Dublin Core metadata record of the item
- Adds an issue date if none already present
- Adds a provenance message (including bitstream checksums)
- Assigns a Handle persistent identifier
- Adds the item to the target collection, and adds appropriate authorization policies
- **Adds the new item to the search and browse indices Workflow Steps**

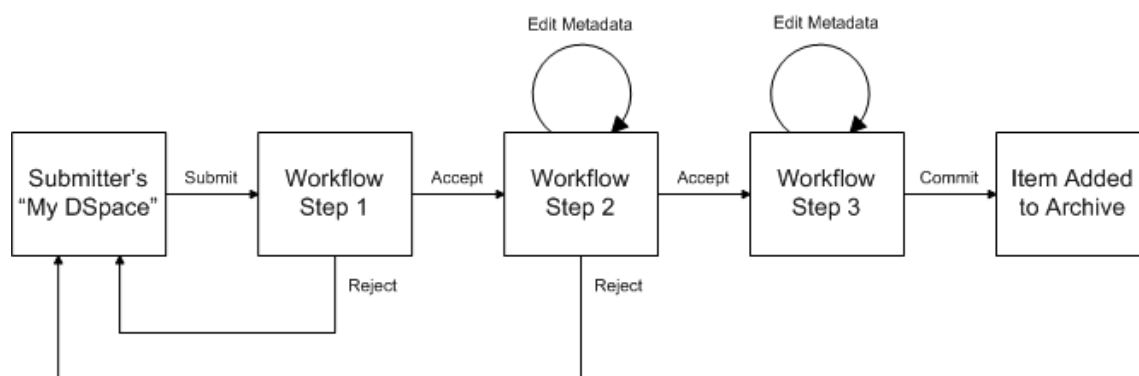
A collection's workflow can have up to three steps. Each collection may have an associated e-person group for performing each step; if no group is associated with a certain step, that step is skipped. If a collection has no e-person groups associated with any step, submissions to that collection are installed straight into the main archive.

In other words, the sequence is this: The collection receives a submission. If the collection has a group assigned for workflow step 1, that step is invoked, and the group is notified. Otherwise, workflow step 1 is skipped. Likewise, workflow steps 2 and 3 are performed if and only if the collection has a group assigned to those steps.

When a step is invoked, the task of performing that workflow step put in the 'task pool' of the associated group. One member of that group takes the task from the pool, and it is then removed from the task pool, to avoid the situation where several people in the group may be performing the same task without realizing it.

The member of the group who has taken the task from the pool may then perform one of three actions:

Workflow Step	Possible actions
1	Can accept submission for inclusion, or reject submission.
2	Can edit metadata provided by the user with the submission, but cannot change the submitted files. Can accept submission for inclusion, or reject submission.
3	Can edit metadata provided by the user with the submission, but cannot change the submitted files. Must then commit to archive; may not reject submission.



Submission Workflow in DSpace

If a submission is rejected, the reason (entered by the workflow participant) is e-mailed to the submitter, and it is returned to the submitter's 'My DSpace' page. The submitter can then make any necessary modifications and re-submit, whereupon the process starts again.

If a submission is 'accepted', it is passed to the next step in the workflow. If there are no more workflow steps with associated groups, the submission is installed in the main archive.

One last possibility is that a workflow can be 'aborted' by a DSpace site administrator. This is accomplished using the administration UI.

The reason for this apparently arbitrary design is that it was the simplest case that covered the needs of the early adopter communities at MIT. The functionality of the workflow system will no doubt be extended in the future.

2.10. Supervision and Collaboration

In order to facilitate, as a primary objective, the opportunity for thesis authors to be supervised in the preparation of their e-thesis, a supervision order system exists to bind groups of other users (thesis supervisors) to an item in someone's pre-submission workspace. The bound group can have system policies associated with it that allow different levels of interaction with the student's item; a small set of default policy groups are provided:

- Full editorial control
- View item contents
- No policies
 - Once the default set has been applied, a system administrator may modify them as they would any other policy set in DSpace

This functionality could also be used in situations where researchers wish to collaborate on a particular submission, although there is no particular collaborative workspace functionality.

2.11. Handles

Researchers require a stable point of reference for their works. The simple evolution from sharing of citations to emailing of URLs broke when Web users learned that sites can disappear or be reconfigured without notice, and that their bookmark files containing critical links to research results couldn't be trusted long term. To help solve this problem, a core DSpace feature is the creation of persistent identifier for every item, collection and community stored in DSpace. To persist identifier, DSpace requires a storage- and location-independent mechanism for creating and maintaining identifiers. DSpace uses the <http://www.handle.net/> for creating these identifiers. The rest of this section assumes a basic familiarity with the Handle system.



DSpace uses Handles primarily as a means of assigning globally unique identifiers to objects. Each site running DSpace needs to obtain a Handle 'prefix' from CNRI, so we know that if we create identifiers with that prefix, they won't clash with identifiers created elsewhere.

Presently, Handles are assigned to communities, collections, and items. Bundles and bitstreams are not assigned Handles, since over time, the way in which an item is encoded as bits may change, in order to allow access with future technologies and devices. Older versions may be moved to off-line storage as a new standard becomes de facto. Since it's usually the *item* that is being preserved, rather than the particular bit encoding, it only makes sense to persistently identify and allow access to the item, and allow users to access the appropriate bit encoding from there.

Of course, it may be that a particular bit encoding of a file is explicitly being preserved; in this case, the bitstream could be the only one in the item, and the item's Handle would then essentially refer just to that bitstream. The same bitstream can also be included in other items, and thus would be citable as part of a greater item, or individually.

The Handle system also features a global resolution infrastructure; that is, an end-user can enter a Handle into any service (e.g. Web page) that can resolve Handles, and the end-user will be directed to the object (in the case of DSpace, community, collection or item) identified by that Handle. In order to take advantage of this feature of the Handle system, a DSpace site must also run a 'Handle server' that can accept and resolve incoming resolution requests. All the code for this is included in the DSpace source code bundle.

Handles can be written in two forms:

```
hdl:1721.123/4567
http://hdl.handle.net/1721.123/4567
```

The above represent the same Handle. The first is possibly more convenient to use only as an identifier; however, by using the second form, any Web browser becomes capable of resolving Handles. An end-user need only access this form of the Handle as they would any other URL. It is possible to enable some browsers to resolve the first form of Handle as if they were standard URLs using `http://www.handle.net/resolver/index.html`, but since the first form can always be simply derived from the second, DSpace displays Handles in the second form, so that it is more useful for end-users.

It is important to note that DSpace uses the CNRI Handle infrastructure only at the 'site' level. For example, in the above example, the DSpace site has been assigned the prefix '1721.123'. It is still the responsibility of the DSpace site to maintain the association between a full Handle (including the '4567' local part) and the community, collection or item in question.

2.12. Bitstream 'Persistent' Identifiers

Similar to handles for DSpace items, bitstreams also have 'Persistent' identifiers. They are more volatile than Handles, since if the content is moved to a different server or organization, they will no longer work (hence the quotes around 'persistent'). However, they are more easily persisted than the simple URLs based on database primary key previously used. This means that external systems can more reliably refer to specific bitstreams stored in a DSpace instance.

Each bitstream has a sequence ID, unique within an item. This sequence ID is used to create a persistent ID, of the form:

dspace url/bitstream/handle/sequence ID/filename

For example:

```
https://dspace.myu.edu/bitstream/123.456/789/24/foo.html
```

The above refers to the bitstream with sequence ID 24 in the item with the Handle *hdl:123.456/789*. The *foo.html* is really just there as a hint to browsers: Although DSpace will provide the appropriate MIME type, some browsers only function correctly if the file has an expected extension.



2.13. Storage Resource Broker (SRB) Support

DSpace offers two means for storing bitstreams. The first is in the file system on the server. The second is using <http://www.sdsc.edu/srb>. Both are achieved using a simple, lightweight API.

SRB is purely an option but may be used in lieu of the server's file system or in addition to the file system. Without going into a full description, SRB is a very robust, sophisticated storage manager that offers essentially unlimited storage and straightforward means to replicate (in simple terms, backup) the content on other local or remote storage resources.

2.14. Search and Browse

DSpace allows end-users to discover content in a number of ways, including:

- Via external reference, such as a Handle
- Searching for one or more keywords in metadata or extracted full-text

- Browsing through title, author, date or subject indices, with optional image thumbnails

Search is an essential component of discovery in DSpace. Users' expectations from a search engine are quite high, so a goal for DSpace is to supply as many search features as possible. DSpace's indexing and search module has a very simple API which allows for indexing new content, regenerating the index, and performing searches on the entire corpus, a community, or collection. Behind the API is the Java freeware search engine <http://jakarta.apache.org/lucene/>. Lucene gives us fielded searching, stop word removal, stemming, and the ability to incrementally add new indexed content without regenerating the entire index. The specific Lucene search indexes are configurable enabling institutions to customize which DSpace metadata fields are indexed.

Another important mechanism for discovery in DSpace is the browse. This is the process whereby the user views a particular index, such as the title index, and navigates around it in search of interesting items. The browse subsystem provides a simple API for achieving this by allowing a caller to specify an index, and a subsection of that index. The browse subsystem then discloses the portion of the index of interest. Indices that may be browsed are item title, item issue date, item author, and subject terms. Additionally, the browse can be limited to items within a particular collection or community.

2.15. HTML Support

For the most part, at present DSpace simply supports uploading and downloading of bitstreams as-is. This is fine for the majority of commonly-used file formats – for example PDFs, Microsoft Word documents, spreadsheets and so forth. HTML documents (Web sites and Web pages) are far more complicated, and this has important ramifications when it comes to digital preservation:

- Web pages tend to consist of several files – one or more HTML files that contain references to each other, and stylesheets and image files that are referenced by the HTML files.
- Web pages also link to or include content from other sites, often imperceptibly to the end-user. Thus, in a few year's time, when someone views the preserved Web site, they will probably find that many links are now broken or refer to other sites than are now out of context. In fact, it may be unclear to an end-user when they are viewing content stored in DSpace and when they are seeing content included from another site, or have navigated to a page that is not stored in DSpace. This problem can manifest when a submitter uploads some HTML content. For example, the HTML document may include an image from an external Web site, or even their local hard drive. When the submitter views the HTML in DSpace, their browser is able to use the reference in the HTML to retrieve the appropriate image, and so to the submitter, the whole HTML document appears to have been deposited correctly. However, later on, when another user tries to view that HTML, their browser might not be able to retrieve the included image since it may have been removed from the external server. Hence the HTML will seem broken.



- Often Web pages are produced dynamically by software running on the Web server, and represent the state of a changing database underneath it.
Dealing with these issues is the topic of much active research. Currently, DSpace bites off a small, tractable chunk of this problem. DSpace can store and provide on-line browsing capability for *self-contained, non-dynamic* HTML documents. In practical terms, this means:
 - No dynamic content (CGI scripts and so forth)
 - All links to preserved content must be *relative links*, that do not refer to 'parents' above the 'root' of the HTML document/site:
 - *diagram.gif* is OK
 - *image/foo.gif* is OK
 - *../index.html* is only OK in a file that is at least a directory deep in the HTML document/site hierarchy
 - */stylesheet.css* is not OK (the link will break)
 - <http://somedomain.com/content.html> is not OK (the link will continue to link to the external site which may change or disappear)
 - Any 'absolute links' (e.g. <http://somedomain.com/content.html>) are stored 'as is', and will continue to link to the external content (as opposed to relative links, which will link to the copy of the content stored in DSpace.) Thus, over time, the content referred to by the absolute link may change or disappear.

2.16. OAI Support

The <http://www.openarchives.org/> has developed a <http://www.openarchives.org/OAI/openarchivesprotocol.html>. This allows sites to programmatically retrieve or 'harvest' the metadata from several sources, and offer services using that metadata, such as indexing or linking services. Such a service could allow users to access information from a large number of sites from one place.

DSpace exposes the Dublin Core metadata for items that are publicly (anonymously) accessible. Additionally, the collection structure is also exposed via the OAI protocol's 'sets' mechanism. OCLC's open source <http://www.oclc.org/research/software/oai/cat.shtm> framework is used to provide this functionality.

You can also configure the OAI service to make use of any crosswalk plugin to offer additional metadata formats, such as MODS.

DSpace's OAI service does support the exposing of deletion information for withdrawn items, but not for items that are 'expunged' (see above). DSpace also supports OAI-PMH resumption tokens.

2.17. OpenURL Support

DSpace supports the <http://www.sfxit.com/OpenURL/> from <http://www.sfxit.com/>, in a rather simple fashion. If your institution has an SFX server, DSpace will display an OpenURL link on every item page, automatically using the Dublin Core metadata. Additionally, DSpace can respond to incoming OpenURLs. Presently it simply passes the information in the OpenURL to the search subsystem. A list of results is then displayed, which usually gives the relevant item (if it is in DSpace) at the top of the list.

2.18. Creative Commons Support

DSpace provides support for Creative Commons licenses to be attached to items in the repository. They represent an alternative to traditional copyright. To learn more about Creative Commons, visit <http://creativecommons.org>. Support for the licenses is controlled by a site-wide configuration option, and since



license selection involves redirection to the Creative Commons website, additional parameters may be configured to work with a proxy server. If the option is enabled, users may select a Creative Commons license during the submission process, or elect to skip Creative Commons licensing. If a selection is made a copy of the license text and RDF metadata is stored along with the item in the repository. There is also an indication - text and a Creative Commons icon - in the item display page of the web user interface when an item is licensed under Creative Commons.

2.19. Subscriptions

As noted above, end-users (e-people) may 'subscribe' to collections in order to be alerted when new items appear in those collections. Each day, end-users who are subscribed to one or more collections will receive an e-mail giving brief details of all new items that appeared in any of those collections the previous day. If no new items appeared in any of the subscribed collections, no e-mail is sent. Users can unsubscribe themselves at any time. RSS feeds of new items are also available for collections and communities.

2.20. Import and Export

DSpace also includes batch tools to import and export items in a simple directory structure, where the Dublin Core metadata is stored in an XML file. This may be used as the basis for moving content between DSpace and other systems.

There is also a METS-based export tool, which exports items as METS-based metadata with associated bitstreams referenced from the METS file.

2.21. Registration

Registration is an alternate means of incorporating items, their metadata, and their bitstreams into DSpace by taking advantage of the bitstreams already being in accessible computer storage. 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.

2.22. Statistics

Various statistical reports about the contents and use of your system can be automatically generated by the system. These are generated by analysing DSpace's log files. Statistics can be broken down monthly.

The report includes data such as:

- A customisable general summary of activities in the archive, by default including:
 - Number of item views
 - Number of collection visits
 - Number of community visits
 - Number of OAI Requests
- Customisable summary of archive contents
- Broken-down list of item viewings
- A full break-down of all system activity



- User logins
 - Most popular searches
- The results of statistical analysis can be presented on a by-month and an in-total report, and are available via the user interface. The reports can also either be made public or restricted to administrator access only.

2.23. Checksum Checker

The purpose of the checker is to verify that the content in a DSpace repository has not become corrupted or been tampered with. The functionality can be invoked on an ad-hoc basis from the command line, or configured via cron or similar. Options exist to support large repositories that cannot be entirely checked in one run of the tool. The tool is extensible to new reporting and checking priority approaches.

2.24. Usage Instrumentation

DSpace can report usage events, such as bitstream downloads, to a pluggable event processor. This can be used for developing customized usage statistics, for example. Sample event processor plugins writes event records to a file as tab-separated values or XML.

3. Installation

3.1. For the Impatient

Since some users might want to get their test version up and running as fast as possible, offered below is an *unsupported* outline of getting DSpace to run quickly.

Only experienced unix admins should even attempt the following without going to Section 3.3

```
useradd -m dspace
gunzip -c dspace-1.x-src-release.tar.gz | tar -xf -
createuser -U postgres -d -A -P dspace
createdb -U dspace -E UNICODE dspace
cd [dspace-source]/dspace/config
vi dspace.cfg
mkdir [dspace]
chown dspace [dspace]
su - dspace
cd [dspace-source]/dspace
mvn package
cd [dspace-source]/dspace/target/dspace-<version>-build.dir
ant fresh_install
cp -r [dspace]/webapps/* [tomcat]/webapps
/etc/init.d/tomcat start
[dspace]/bin/create-administrator
```

3.2. Prerequisite Software

The list below describes the third-party components and tools you'll need to run a DSpace server. These are just guidelines. Since DSpace is built on open source, standards-based tools, there are numerous other possibilities and setups.

Also, please note that the configuration and installation guidelines relating to a particular tool below are here for convenience. You should refer to the documentation for each individual component for complete and up-to-date details. Many of the tools are updated on a frequent basis, and the guidelines below may become out of date.



3.2.1. UNIX-like OS or Microsoft Windows

- UNIX-like OS (Linux, HP/UX etc) : Many distributions of Linux/Unix come with some of the dependencies below pre installed or easily installed via updates, you should consult your particular distributions documentation to determine what is already available.
- Microsoft Windows: (see full Windows Instructions for full set of prerequisites)

3.2.2. Java JDK 5 or later (standard SDK is fine, you don't need J2EE)

DSpace now required Java 5 or greater because of usage of new language capabilities introduced in 5 that make coding easier and cleaner.

Java 5 or later can be downloaded from the following location: <http://java.sun.com/javase/downloads/index.jsp>

3.2.3. Apache Maven 2.0.8 or later (Java build tool)

Maven is necessary in the first stage of the build process to assemble the installation package for your DSpace instance. It gives you the flexibility to customize DSpace using the existing Maven projects found in the `[dspace-source]/dspace/modules` directory or by adding in your own Maven project to build the installation package for DSpace, and apply any custom interface "overlay" changes.

Maven can be downloaded from the the following location: <http://maven.apache.org/download.html>

3.2.4. Apache Ant 1.7 or later (Java build tool)

Apache Ant is still required for the second stage of the build process. It is used once the installation package has been constructed in `[dspace-source]/dspace/target/dspace-<version>-build.dir` and still uses some of the familiar ant build targets found in the 1.4.x build process.

Ant can be downloaded from the following location: <http://ant.apache.org/>

3.2.5. Relational Database: (PostgreSQL or Oracle).

- ***PostgreSQL 7.3 or greater*** PostgreSQL can be downloaded from the following location: <http://www.postgresql.org/> Its highly recommended that you try to work with Postgres 8.x or greater, however, 7.3 or greater should still work. Unicode (specifically UTF-8) support must be enabled. This is enabled by default in 8.0+. For 7.x, be sure to compile with the following options to the 'configure' script: `--enable-multibyte --enable-unicode --with-java` Once installed, you need to enable TCP/IP connections (DSpace uses JDBC). For 7.x, edit `postgresql.conf` (usually in `/usr/local/pgsql/data` or `/var/lib/pgsql/data`), and add this line: `tcpip_socket = true` For 8.0+, in `_postgresql.conf` uncomment the line starting: `listen_addresses = 'localhost'` Then tighten up security a bit by editing `_pg_hba.conf` and adding this line: `_host dspace dspace 127.0.0.1 255.255.255.255 md5` Then restart PostgreSQL.
- ***Oracle 9 or greater*** Details on acquiring Oracle can be downloaded from the following location: <http://www.oracle.com/database/> You will need to create a database for DSpace. Make sure that the character set is one of the Unicode character sets. DSpace uses UTF-8 natively, and it is suggested that the Oracle database use the same character set. You will also need to create a user account for DSpace (e.g. dspace,) and ensure that it has permissions to add and remove tables in the database. Refer to the Quick Installation for more details. *NOTE: DSpace uses sequences to generate unique object IDs - beware Oracle sequences, which are said to lose their values when doing a database export/import, say restoring from a backup. Be sure to run the script `etc/update-sequences.sql`. For people interested in switching from Postgres to Oracle, I know of no tools that would do this automatically. You will need to recreate the community, collection, and eperson structure in the Oracle system, and then use the item export and import tools to move your content over.



3.2.6. Servlet Engine: (Jakarta Tomcat 4.x, Jetty, Caucho Resin or equivalent).

- **Jakarta Tomcat 4.x or later.** Tomcat can be downloaded from the following location: <http://tomcat.apache.org/whichversion.html> Note that DSpace will need to run as the same user as Tomcat, so you might want to install and run Tomcat as a user called 'dspace'. Set the environment variable TOMCAT_USER appropriately. You need to ensure that Tomcat has a) enough memory to run DSpace and b) uses UTF-8 as its default file encoding for international character support. So ensure in your startup scripts (etc) that the following environment variable is set: `JAVA_OPTS="-Xmx512M -Xms64M -Dfile.encoding=UTF-8"` *Modifications in `[/tomcat]/conf/server.xml` : You also need to alter Tomcat's default configuration to support searching and browsing of multi-byte UTF-8 correctly. You need to add a configuration option to the `<Connector>` element in `[tomcat]/conf/server.xml`: `URIEncoding="UTF-8"` e.g. if you're using the default Tomcat config, it should read:

```
<!-- Define a non-SSL HTTP/1.1 Connector on port 8080 -->
<Connector port="8080"
    maxThreads="150"
    minSpareThreads="25"
    maxSpareThreads="75"
    enableLookups="false"
    redirectPort="8443"
    acceptCount="100"
    connectionTimeout="20000"
    disableUploadTimeout="true"
    URIEncoding="UTF-8" />
```

You may change the port from 8080 by editing it in the file above, and by setting the variable `CONNECTOR_PORT` in `server.xml`

- *Jetty or Caucho Resin* DSpace will also run on an equivalent servlet Engine, such as Jetty (<http://www.mortbay.org/jetty/index.html>) or Caucho Resin (<http://www.caucho.com/>). Jetty and Resin are configured for correct handling of UTF-8 by default.

3.2.7. Perl (required for `[dspace]/bin/dspace-info.pl`)

3.3. Installation Options

3.3.1. Overview of Install Options

With the advent of a new Apache <http://maven.apache.org/> based build architecture in DSpace 1.5.x, you now have two options in how you may wish to install and manage your local installation of DSpace. If you've used DSpace 1.4.x, please recognize that the initial build procedure has changed to allow for more customization. You will find the later 'Ant based' stages of the installation procedure familiar. Maven is used to resolve the dependencies of DSpace online from the 'Maven Central Repository' server.

It is important to note that the strategies are identical in terms of the list of procedures required to complete the build process, the only difference being that the Source Release includes "more modules" that will be built given their presence in the distribution package.

- Default Release (`dspace-<version>-release.zip`)
 - This distribution will be adequate for most cases of running a DSpace instance. It is intended to be the quickest way to get DSpace installed and running while still allowing for customization of the themes and branding of your DSpace instance.
 - This method allows you to customize DSpace configurations (in `dspace.cfg`) or user interfaces, using basic pre-built interface "overlays".



- It downloads "precompiled" libraries for the core dspace-api, supporting servlets, taglibraries, aspects and themes for the dspace-xmlui, dspace-xmlui and other webservice/applications.
- This approach exposes the parts of the application that the DSpace committers would prefer to see customized. All other modules are downloaded from the 'Maven Central Repository' The directory structure for this release is the following:
 - *[dspace-source]*
 - *dspace/* - DSpace 'build' and configuration module
 - *pom.xml* - DSpace Parent Project definition
- Source Release (dspace-<version>-src-release.zip)
 - This method is recommended for those who wish to develop DSpace further or alter its underlying capabilities to a greater degree.
 - It contains **all** dspace code for the core dspace-api, supporting servlets, taglibraries, aspects and themes for Manakin (dspace-xmlui), and other webservice/applications.
 - Provides all the same capabilities as the normal release. The directory structure for this release is more detailed:
 - *[dspace-source]*
 - *dspace/* - DSpace 'build' and configuration module
 - *dspace-api/* - Java API source module
 - *dspace-jspui/* - JSP-UI source module
 - *dspace-oai* - OAI-PMH source module
 - *dspace-xmlui* - XML-UI (Manakin) source module
 - *dspace-lni* - Lightweight Network Interface source module
 - *dspace-sword* – SWORD (Simple Web-serve Offering Repository Deposit) deposit service source module
 - *pom.xml* - DSpace Parent Project definition

3.3.2. Overview of DSpace Directories

Before beginning an installation, it is important to get a general understanding of the DSpace directories and the names by which they are generally referred. (Please attempt to use these below directory names when asking for help on the DSpace Mailing Lists, as it will help everyone better understand what directory you may be referring to.)

DSpace uses three separate directory trees. Although you don't need to know all the details of them in order to install DSpace, you do need to know they exist and also know how they're referred to in this document:

1. **The installation directory**, referred to as *_[dspace]_*. This is the location where DSpace is installed and running off of it is the location that gets defined in the dspace.cfg as "dspace.dir". It is where all the DSpace configuration files, command line scripts, documentation and webapps will be installed to.
2. **The source directory**, referred to as *_[dspace-source]_*. This is the location where the DSpace release distribution has been unzipped into. It usually has the name of the archive that you expanded such as



dspace-<version>release or *dspace-<version>-src-release*. It is the directory where all of your "build" commands will be run.

3. **The web deployment directory.** This is the directory that contains your DSpace web application(s). In DSpace 1.5.x and above, this corresponds to `_[dspace]/webapps_` by default. However, if you are using Tomcat, you may decide to copy your DSpace web applications from `_[dspace]/webapps/_` to `[tomcat]/webapps/` (with `[tomcat]` being wherever you installed Tomcat—also known as `$CATALINA_HOME`). For details on the contents of these separate directory trees, refer to `directories.html`. *Note that the `_[dspace-source]` and `[dspace]` directories are always separate!*

3.3.3. Installation

This method gets you up and running with DSpace quickly and easily. It is identical in both the Default Release and Source Release distributions.

1. **Create the DSpace user.** This needs to be the same user that Tomcat (or Jetty etc.) will run as. e.g. as **root** run:

```
useradd -m dspace
```

2. **Download** the <http://sourceforge.net/projects/dspace/> There are two version available with each release of DSpace: (*dspace-1.x-release*. and *dspace-1.x-src-release.xxx*); you only need to choose one. If you want a copy of all underlying Java source code, you should download the *dspace-1.x-src-release.xxx* Within each version, you have a choice of compressed file format. Choose the one that best fits your environment.
3. **Unpack the DSpace software.** After downloading the software, based on the compression file format, choose one of the following methods to unpack your software:

- a. **Zip file.** If you downloaded *dspace-1.6-release.zip* do the following:

```
unzip dspace-1.6-release.zip
```

- b. **.gz file.** If you downloaded *dspace-1.6-release.tar.gz* do the following:

```
gunzip -c dspace-1.6-release.tar.gz | tar -xf -
```

- c. **.bz2 file.** If you downloaded `_dspace-1.6-release.tar.bz2_` do the following:

```
bunzip2 dspace-1.6-release.tar.bz | tar -xf -
```

For ease of reference, we will refer to the location of this unzipped version of the DSpace release as `[dspace-source]` in the remainder of these instructions. After unpacking the file, the user may wish to change the ownership of the *dspace-1.6-release* to the 'dspace' user. (And you may need to change the group).

4. **Database Setup***PostgreSQL:*

- a. A PostgreSQL 8.1-404 jdbc3 driver is configured as part of the default DSpace build. You no longer need to copy any postgresSQL jars to get postgresSQL installed.
- b. Create a *dspace* database, owned by the *dspace* PostgreSQL user (*you are still logged in at 'root'*):

```
createuser -U postgres -d -A -P dspace ; createdb -U dspace -E UNICODE dspace
```

You will be prompted for a password for the DSpace database. (This isn't the same as the *dspace* user's UNIX password.)*Oracle:*



- c. Setting up oracle is a bit different now. You will need still need to get a Copy of the oracle JDBC driver, but instead of copying it into a lib directory you will need to install it into your local Maven repository. You'll need to download it first from this location: http://www.oracle.com/technology/software/tech/java/sqlj_jdbc/htdocs/jdbc_10201.html \$
`mvn install:install-file -Dfile=ojdbc14.jar -DgroupId=com.oracle \ -DartifactId=ojdbc14 -Dversion=10.2.0.2.0 -Dpackaging=jar -DgeneratePom=true`
- d. Create a database for DSpace. Make sure that the character set is one of the Unicode character sets. DSpace uses UTF-8 natively, and it is suggested that the Oracle database use the same character set. Create a user account for DSpace (e.g. *dspace*.) and ensure that it has permissions to add and remove tables in the database.
- e. Edit the `[dspace-source]/dspace/config/dspace.cfg` database settings:

```
db.name      = oracle
db.url       = jdbc:oracle:thin:@//host:port/dspace
db.driver    = oracle.jdbc.OracleDriver
```

5. **Initial Configuration*** Edit `[dspace-source]/dspace/config/dspace.cfg`, in particular you'll need to set these properties: `dspace.dir` – must be set to the `[dspace]` (installation) directory. `dspace.url` – complete URL of this server's DSpace home page. `dspace.hostname` – fully-qualified domain name of web server. `dspace.name` – "Proper" name of your server, e.g. "My Digital Library". `db.password` – the database password you entered in the previous step. `mail.server` – fully-qualified domain name of your outgoing mail server. `mail.from.address` – the "From:" address to put on email sent by DSpace. `feedback.recipient` – mailbox for feedback mail. `mail.admin` – mailbox for DSpace site administrator. `alert.recipient` – mailbox for server errors/alerts (not essential but very useful!) `registration.notify` – mailbox for emails when new users register (optional) *NOTE: You can interpolate the value of one configuration variable in the value of another one. For example, to set `feedback.recipient` to the same value as `mail.admin`, the line would look like:

```
feedback.recipient = ${mail.admin}
```

Refer to 5.2. *General Configuration* for details and examples of the above.

6. ***DSpace Directory*** Create the directory for the DSpace installation (i.e. `[dspace]`). As *root* (or a user with appropriate permissions), run:

```
mkdir [dspace]
chown dspace [dspace]
```

(Assuming the *dspace* UNIX username.)

7. ***Installation Package*** As the *dspace* UNIX user, generate the DSpace installation package in the `[dspace-source]/dspace` directory:

```
cd [dspace-source]/dspace/
mvn package
```

Note: without any extra arguments, the DSpace installation package is initialized for PostgreSQL. *If you want to use Oracle instead, you should build the DSpace installation package as follows:*

```
mvn -Ddb.name=oracle package
```

8. ***Build DSpace and Initialize Database*** As the *dspace* UNIX user, initialize the DSpace database and install DSpace to `[dspace]`:

```
cd [dspace-source]/dspace/target/dspace-[version]-build.dir
ant fresh_install
```



To see a complete list of build targets, run: `ant help` *The most likely thing to go wrong here is the database connection. See the [_3.7 Common Problems Section](#).*

9. **Deploy Web Applications** *You have two choices or techniques for having Tomcat/Jetty/Resin serve up your web applications. ***Technique A.** Simple and complete. You copy only (or all) of the DSpace Web application(s) you wish to use from the `[dSPACE]/webapps` directory to the appropriate directory in your Tomcat/Jetty/Resin installation. For example: `cp -R [dSPACE]/webapps/* [tomcat]/webapps` (This will copy all the web applications to Tomcat.) `cp -R [dSPACE]/webapps/jspui [tomcat]/webapps` (This will copy only the jspui web application to Tomcat.) ***Technique B.** *Tell your Tomcat/Jetty/Resin installation where to find your DSpace web application(s). As an example, in the `<Host>` section of your `[tomcat]/conf/server.xml` you could add lines similar to the following (but replace `[dSPACE]` with your installation location):

```
<!-- Define the default virtual host
Note: XML Schema validation will not work with Xerces 2.2.
-->
<Host name="localhost"  appBase="[dSPACE]/webapps"
....
```

10. *Administrator Account* Create an initial administrator account:

```
[dSPACE]/bin/create-administrator
```

11. *Initial Startup!* Now the moment of truth! Start up (or restart) Tomcat/Jetty/Resin. Visit the base URL(s) of your server, depending on which DSpace web applications you want to use. You should see the DSpace home page. Congratulations! Base URLs of DSpace Web Applications:

- *JSP User Interface* - (e.g.) <http://dSPACE.myu.edu:8080/jspui>
- *XML User Interface (aka. Manakin)* - (e.g.) <http://dSPACE.myu.edu:8080/xmlui>
- *OAI-PMH Interface* - (e.g.) <http://dSPACE.myu.edu:8080/oai/request?verb=Identify> (Should return an XML-based response)

In order to set up some communities and collections, you'll need to login as your DSpace Administrator (which you created with `create-administrator` above) and access the administration UI in either the JSP or XML user interface.

3.4. Advanced Installation

The above installation steps are sufficient to set up a test server to play around with, but there are a few other steps and options you should probably consider before deploying a DSpace production site.

3.4.1. 'cron' Jobs

A couple of DSpace features require that a script is run regularly – the e-mail subscription feature that alerts users of new items being deposited, and the new 'media filter' tool, that generates thumbnails of images and extracts the full-text of documents for indexing.

To set these up, you just need to run the following command as the `dSPACE` UNIX user:

```
crontab -e
```

Then add the following lines:

```
# Send out subscription e-mails at 01:00 every day
0 1 * * * [dSPACE]/bin/sub-daily
# Run the media filter at 02:00 every day
0 2 * * * [dSPACE]/bin/filter-media
# Run the checksum checker at 03:00
0 3 * * * [dSPACE]/bin/checker -lp
# Mail the results to the sysadmin at 04:00
```



```
0 4 * * * [dSPACE]/bin/dsrun org.dSPACE.checker.DailyReportEmailer -c
```

Naturally you should change the frequencies to suit your environment.

PostgreSQL also benefits from regular 'vacuuming', which optimizes the indices and clears out any deleted data. Become the *postgres* UNIX user, run *crontab -e* and add (for example):

```
# Clean up the database nightly at 4.20am
20 4 * * * vacuumdb --analyze dSPACE > /dev/null 2>&1
```

In order that statistical reports are generated regularly and thus kept up to date you should set up the following cron jobs:

```
# Run stat analyses
0 1 * * * [dSPACE]/bin/stat-general
0 1 * * * [dSPACE]/bin/stat-monthly
0 2 * * * [dSPACE]/bin/stat-report-general
0 2 * * * [dSPACE]/bin/stat-report-monthly
```

Obviously, you should choose execution times which are most useful to you, and you should ensure that the *report* scripts run a short while after the analysis scripts to give them time to complete (a run of around 8 months worth of logs can take around 25 seconds to complete); the resulting reports will let you know how long analysis took and you can adjust your cron times accordingly.

3.4.2. Multilingual Installation

In order to deploy a multilingual version of DSpace you have to configure two parameters in *[dSPACE-source]/config/dSPACE.cfg*:

default.locale, e. g. *default.locale = en*

webui.supported_locales, e. g. *webui.supported_locales = en, de*

The Locales might have the form *country*, *country_language*, *country_language_variant*.

According to the languages you wish to support, you have to make sure, that all the *i18n* related files are available see the Multilingual User Interface Configuring MultiLingual Support section for the JSPUI or the Multilingual Support for XMLUI in the configuration documentation.

3.4.3. DSpace over HTTPS

If your DSpace is configured to have users login with a username and password (as opposed to, say, client Web certificates), then you should consider using HTTPS. Whenever a user logs in with the Web form (e.g. *dSPACE.myuni.edu/dSPACE/password-login*) their DSpace password is exposed in plain text on the network. This is a very serious security risk since network traffic monitoring is very common, especially at universities. If the risk seems minor, then consider that your DSpace administrators also login this way and they have ultimate control over the archive.

The solution is to use *HTTPS* (HTTP over SSL, i.e. Secure Socket Layer, an encrypted transport), which protects your passwords against being captured. You can configure DSpace to require SSL on all "authenticated" transactions so it only accepts passwords on SSL connections.

The following sections show how to set up the most commonly-used Java Servlet containers to support HTTP over SSL.

To enable the HTTPS support in Tomcat 5.0:

1. **For Production use:** Follow this procedure to set up SSL on your server. Using a "real" server certificate ensures your users' browsers will accept it without complaints. In the examples below, *\$CATALINA_BASE* is the directory under which your Tomcat is installed.
 - a. Create a Java keystore for your server with the password *changeit*, and install your server certificate under the alias *"tomcat"*. This assumes the certificate was put in the file *server.pem*:



```
$JAVA_HOME/bin/keytool -import -noprompt -v -storepass changeit
-keystore $CATALINA_BASE/conf/keystore -alias tomcat -file
myserver.pem
```

- b. Install the CA (Certifying Authority) certificate for the CA that granted your server cert, if necessary. This assumes the server CA certificate is in *ca.pem*:

```
$JAVA_HOME/bin/keytool -import -noprompt -storepass changeit
-trustcacerts -keystore $CATALINA_BASE/conf/keystore -alias ServerCA
-file ca.pem
```

- c. Optional – ONLY if you need to accept client certificates for the X.509 certificate stackable authentication module. See the configuration section for instructions on enabling the X.509 authentication method. Load the keystore with the CA (certifying authority) certificates for the authorities of any clients whose certificates you wish to accept. For example, assuming the client CA certificate is in *client1.pem*:

```
$JAVA_HOME/bin/keytool -import -noprompt -storepass changeit
-trustcacerts -keystore $CATALINA_BASE/conf/keystore -alias client1
-file client1.pem
```

- d. Now add another Connector tag to your *server.xml* Tomcat configuration file, like the example below. The parts affecting or specific to SSL are shown in bold. (You may wish to change some details such as the port, pathnames, and keystore password)

```
<Connector port="8443"
    maxThreads="150" minSpareThreads="25"
    maxSpareThreads="75"
    enableLookups="false"
    disableUploadTimeout="true"
    acceptCount="100" debug="0"
    scheme="https" secure="true" sslProtocol="TLS"
    keystoreFile="conf/keystore" keystorePass="changeit" clientAuth="true" - ONLY if
    using client X.509 certs for authentication!
    truststoreFile="conf/keystore" trustedstorePass="changeit" />
```

Also, check that the default Connector is set up to redirect "secure" requests to the same port as your SSL connector, e.g.:

```
<Connector port="8080"
    maxThreads="150" minSpareThreads="25"
    maxSpareThreads="75"
    enableLookups="false"
    redirectPort="8443"
    acceptCount="100" debug="0" />
```

2. **Quick-and-dirty Procedure for Testing:** If you are just setting up a DSpace server for testing, or to experiment with HTTPS, then you don't need to get a real server certificate. You can create a "self-signed" certificate for testing; web browsers will issue warnings before accepting it but they will function exactly the same after that as with a "real" certificate. In the examples below, *\$CATALINA_BASE* is the directory under which your Tomcat is installed.

- a. Optional – ONLY if you don't already have a server certificate. Follow this sub-procedure to request a new, signed server certificate from your Certifying Authority (CA):

- Create a new key pair under the alias name "*tomcat*". When generating your key, give the Distinguished Name fields the appropriate values for your server and institution. CN should be the fully-qualified domain name of your server host. Here is an example:

```
$JAVA_HOME/bin/keytool -genkey -alias tomcat -keyalg RSA -keysize
1024 \
-keystore $CATALINA_BASE/conf/keystore -storepass changeit
```



```
-validity 365 \  
-dname 'CN=dspace.myuni.edu, OU=MIT Libraries, O=Massachusetts  
Institute of Technology, L=Cambridge, S=MA, C=US'
```

- Then, create a *CSR* (Certificate Signing Request) and send it to your Certifying Authority. They will send you back a signed Server Certificate. This example command creates a CSR in the file `_tomcat.csr_`

```
$JAVA_HOME/bin/keytool -keystore $CATALINA_BASE/conf/keystore  
-storepass changeit \  
-certreq -alias tomcat -v -file tomcat.csr
```

- Before importing the signed certificate, you must have the CA's certificate in your keystore as a *trusted certificate*. Get their certificate, and import it with a command like this (for the example `mitCA.pem`):

```
$JAVA_HOME/bin/keytool -keystore $CATALINA_BASE/conf/keystore  
-storepass changeit \  
-import -alias mitCA -trustcacerts -file mitCA.pem
```

- Finally, when you get the signed certificate from your CA, import it into the keystore with a command like the following example: (cert is in the file `signed-cert.pem`)

```
$JAVA_HOME/bin/keytool -keystore $CATALINA_BASE/conf/keystore  
-storepass changeit \  
-import -alias tomcat -trustcacerts -file signed-cert.pem
```

Since you now have a signed server certificate in your keystore, you can, obviously, skip the next steps of installing a signed server certificate and the server CA's certificate.

- Create a Java keystore for your server with the password `changeit`, and install your server certificate under the alias `"tomcat"`. This assumes the certificate was put in the file `server.pem`:

```
$JAVA_HOME/bin/keytool -genkey -alias tomcat -keyalg RSA -keystore  
$CATALINA_BASE/conf/keystore -storepass changeit
```

When answering the questions to identify the certificate, be sure to respond to "First and last name" with the fully-qualified domain name of your server (e.g. `test-dspace.myuni.edu`). The other questions are not important.

- Optional – ONLY if you need to accept client certificates for the X.509 certificate stackable authentication module See the configuration section for instructions on enabling the X.509 authentication method. Load the keystore with the CA (certifying authority) certificates for the authorities of any clients whose certificates you wish to accept. For example, assuming the client CA certificate is in `client1.pem`:

```
$JAVA_HOME/bin/keytool -import -noprompt -storepass changeit  
-trustcacerts -keystore $CATALINA_BASE/conf/keystore -alias client1  
-file client1.pem
```

- Follow the procedure in the section above to add another Connector tag, for the HTTPS port, to your `server.xml` file.

To use SSL on Apache HTTPD with mod_jk:

If you choose `http://httpd.apache.org/` as your primary HTTP server, you can have it forward requests to the `http://tomcat.apache.org/` via `http://tomcat.apache.org/connectors-doc/`. This can be configured to work over SSL as well. First, you must configure Apache for SSL; for Apache 2.0 see [`http://httpd.apache.org/docs/2.0/ssl/|Apache SSL/TLS Encryption`] for information about using `http://httpd.apache.org/docs/2.0/mod/mod_ssl.html`.



If you are using X.509 Client Certificates for authentication: add these configuration options to the appropriate *httpd* configuration file, e.g. *ssl.conf*, and be sure they are in force for the virtual host and namespace locations dedicated to DSpace:

```
## SSLVerifyClient can be "optional" or
"require"
    SSLVerifyClient optional
    SSLVerifyDepth 10
    SSLCACertificateFile
    path-to-your-client-CA-certificate
    SSLOptions StdEnvVars ExportCertData
```

Now consult the <http://tomcat.apache.org/connectors-doc/> documentation to configure the *mod_jk* (note: **NOT** *mod_jk2*) module. Select the AJP 1.3 connector protocol. Also follow the instructions there to configure your Tomcat server to respond to AJP.

To use SSL on Apache HTTPD with *mod_webapp* consult the DSpace 1.3.2 documentation. Apache have deprecated the *mod_webapp* connector and recommend using *mod_jk*.

To use Jetty's HTTPS support consult the documentation for the relevant tool.

3.4.4. The Handle Server

First a few facts to clear up some common misconceptions:

- You don't **have** to use CNRI's Handle system. At the moment, you need to change the code a little to use something else (e.g PURLs) but that should change soon.
- You'll notice that while you've been playing around with a test server, DSpace has apparently been creating handles for you looking like *hdl:123456789/24* and so forth. These aren't really Handles, since the global Handle system doesn't actually know about them, and lots of other DSpace test installs will have created the same IDs. They're only really Handles once you've registered a prefix with CNRI (see below) and have correctly set up the Handle server included in the DSpace distribution. This Handle server communicates with the rest of the global Handle infrastructure so that anyone that understands Handles can find the Handles your DSpace has created.

If you want to use the Handle system, you'll need to set up a Handle server. This is included with DSpace. Note that this is not required in order to evaluate DSpace; you only need one if you are running a production service. You'll need to obtain a Handle prefix from <http://www.handle.net/>.

A Handle server runs as a separate process that receives TCP requests from other Handle servers, and issues resolution requests to a global server or servers if a Handle entered locally does not correspond to some local content. The Handle protocol is based on TCP, so it will need to be installed on a server that can broadcast and receive TCP on port 2641.

1. To configure your DSpace installation to run the handle server, run the following command: *[dspace]/bin/dspace make-handle-config* Ensure that *_[dspace]/handle-server* matches whatever you have in *dspace.cfg* for the *handle.dir* property.
2. Edit the resulting *[dspace]/handle-server/config.dct* file to include the following lines in the "server_config" clause:

```
"storage_type" = "CUSTOM"
"storage_class" = "org.dspace.handle.HandlePlugin"
```

This tells the Handle server to get information about individual Handles from the DSpace code.

3. Once the configuration file has been generated, you will need to go to <http://hdl.handle.net/4263537/5014> to upload the generated sitebndl.zip file. The upload page will ask you for your contact information. An administrator will then create the naming authority/prefix on the root service (known as the Global Handle Registry), and notify you when this has been completed. You will not be able to continue the handle server installation until you receive further information concerning your naming authority.



4. When CNRI has sent you your naming authority prefix, you will need to edit the *config.dct* file. The file will be found in *[dspace]/handle-server*. Look for "*300:0.NA/YOUR_NAMING_AUTHORITY*" *_Replace* *_YOUR_NAMING_AUTHORITY* with the assigned naming authority prefix sent to you.
5. Now start your handle server (as the dspace user):

```
[dspace]/bin/start-handle-server
```

Note that since the DSpace code manages individual Handles, administrative operations such as Handle creation and modification aren't supported by DSpace's Handle server.

Updating Existing Handle Prefixes

If you need to update the handle prefix on items created before the CNRI registration process you can run the *[dspace]/bin/update-handle-prefix script*. You may need to do this if you loaded items prior to CNRI registration (e.g. setting up a demonstration system prior to migrating it to production). The script takes the current and new prefix as parameters. For example:

```
[dspace]/bin/update-handle-prefix 123456789 1303
```

This script will change any handles currently assigned prefix 123456789 to prefix 1303, so for example handle 123456789/23 will be updated to 1303/23 in the database.

3.4.5. Google and HTML sitemaps

To aid web crawlers index the content within your repository, you can make use of sitemaps. There are currently two forms of sitemaps included in DSpace; Google sitemaps and HTML sitemaps.

Sitemaps allow DSpace to expose its content without the crawlers having to index every page. HTML sitemaps provide a list of all items, collections and communities in HTML format, whilst Google sitemaps provide the same information in gzipped XML format.

To generate the sitemaps, you need to run *[dspace]/bin/generate-sitemaps*. This creates the sitemaps in *[dspace]/sitemaps/*

The sitemaps can be accessed from the following URLs:

- <http://dspace.example.com/dspace/sitemap> - Index sitemap
- <http://dspace.example.com/dspace/sitemap?map=0> - First list of items (up to 50,000)
- <http://dspace.example.com/dspace/sitemap?map=n> - Subsequent lists of items (e.g. 50,0001 to 100,000) etc...

HTML sitemaps follow the same procedure:

- <http://dspace.example.com/dspace/htmlmap> - Index sitemap
- etc...

When running *[dspace]/bin/generate-sitemaps* the script informs Google that the sitemaps have been updated. For this update to register correctly, you must first register your Google sitemap index page (*/dspace/sitemap*) with Google at <http://www.google.com/webmasters/sitemaps/>. If your DSpace server requires the use of a HTTP proxy to connect to the Internet, ensure that you have set *http.proxy.host* and *http.proxy.port* in *[dspace]/config/dspace.cfg*

The URL for pinging Google, and in future, other search engines, is configured in *[dspace-space]/config/dspace.cfg* using the *sitemap.engineurls* setting where you can provide a comma-separated list of URLs to 'ping'.

You can generate the sitemaps automatically every day using an additional cron job:

```
# Generate sitemaps
```



```
0 6 * * * [dSPACE]/bin/generate-sitemaps
```

3.5. Windows Installation

3.5.1. Pre-requisite Software

You'll need to install this pre-requisite software:

- <http://java.sun.com/> or later (standard SDK is fine, you don't need J2EE)
- <http://www.postgresql.org/ftp/> OR <http://www.oracle.com/database/>.
 - If you install PostgreSQL, it's recommended to select to install the pgAdmin III tool
- <http://ant.apache.org/>. Unzip the package in *C:\and add C:\apache-ant-1.6.2\bin* to the *PATH* environment variable. For Ant to work properly, you should ensure that *JAVA_HOME* is set.
- <http://tomcat.apache.org/>
- <http://maven.apache.org/>

3.5.2. Installation Steps

1. Download the DSpace source from <http://sourceforge.net/projects/dspace> and untar it (<http://www.winzip.com/> will do this)
2. Ensure the PostgreSQL service is running, and then run pgAdmin III (Start -> PostgreSQL 8.0 -> pgAdmin III). Connect to the local database as the postgres user and:
 - Create a 'Login Role' (user) called *dSPACE* with the password *dSPACE*
 - Create a database called *dSPACE* owned by the user *dSPACE*, with UTF-8 encoding
3. Update paths in *[dSPACE-source]\dSPACE\config\dSPACE.cfg*. **Note:** Use forward slashes / for path separators, though you can still use drive letters, e.g.: *_dSPACE.dir = C:/DSpace_* Make sure you change all of the parameters with file paths to suit, specifically:

```
dSPACE.dir
  config.template.log4j.properties
  config.template.log4j-handle-plugin.properties
  config.template.oaicat.properties
  assetstore.dir
  log.dir
  upload.temp.dir
  report.dir
  handle.dir
```

4. Create the directory for the DSpace installation (e.g. *C:\DSpace*)
5. Generate the DSpace installation package by running the following from commandline (cmd) from your *[dSPACE-source]/dSPACE/* directory:

```
mvn package
```

Note #1: This will generate the DSpace installation package in your *[dSPACE-source]/dSPACE/target/dSPACE-[version]-build.dir/* directory. Note #2: Without any extra arguments, the DSpace installation package is initialized for PostgreSQL. If you want to use Oracle instead, you should build the DSpace installation package as follows:



```
mvn -Ddb.name=oracle package
```

6. Initialize the DSpace database and install DSpace to *[dSPACE]* (e.g. *C:\DSpace*) by running the following from commandline from your *[dSPACE-source]/dSPACE/target/dSPACE-[version]-build.dir/* directory:

```
ant fresh_install
```

Note: to see a complete list of build targets, run

```
ant help
```

7. Create an administrator account, by running the following from your *[dSPACE]* (e.g. *C:\DSpace*) directory *[dSPACE]\bin\dsrun org.dspace.administer.CreateAdministrator_* and enter the required information
8. Copy the Web application directories from *[dSPACE]\webapps_* to Tomcat's webapps dir, which should be somewhere like *_C:\Program Files\Apache Software Foundation\Tomcat 5.5\webapps*
 - Alternatively, Tell your Tomcat installation where to find your DSpace web application(s). As an example, in the *<Host>* section of your *[tomcat]/conf/server.xml* you could add lines similar to the following (but replace *[dSPACE]* with your installation location):

```
<!-- DEFINE A CONTEXT PATH FOR DSpace JSP User Interface -->
<Context path="/jspui" docBase="[dSPACE]\webapps\jspui" debug="0"
  reloadable="true" cachingAllowed="false"
  allowLinking="true"/>

<!-- DEFINE A CONTEXT PATH FOR DSpace OAI User Interface -->
<Context path="/oai" docBase="[dSPACE]\webapps\oai" debug="0"
  reloadable="true" cachingAllowed="false"
  allowLinking="true"/>
```

9. Start the Tomcat service
10. Browse to either <http://localhost:8080/jspui> or <http://localhost:8080/xmlui>. You should see the DSpace home page for either the JSPUI or XMLUI, respectively.

3.6. Checking Your Installation

TODO

3.7. Known Bugs

In any software project of the scale of DSpace, there will be bugs. Sometimes, a stable version of DSpace includes known bugs. We do not always wait until every known bug is fixed before a release. If the software is sufficiently stable and an improvement on the previous release, and the bugs are minor and have known workarounds, we release it to enable the community to take advantage of those improvements.

The known bugs in a release are documented in the *KNOWN_BUGS* file in the source package.

Please see the # for further information on current bugs, and to find out if the bug has subsequently been fixed. This is also where you can report any further bugs you find.

3.8. Common Problems

In an ideal world everyone would follow the above steps and have a fully functioning DSpace. Of course, in the real world it doesn't always seem to work out that way. This section lists common problems that people



encounter when installing DSpace, and likely causes and fixes. This is likely to grow over time as we learn about users' experiences.

- **Database errors occur when you run `ant fresh_install`:** There are two common errors that occur. If your error looks like this--

```
[java] 2004-03-25 15:17:07,730 INFO
      org.dspace.storage.rdbms.InitializeDatabase @ Initializing Database
[java] 2004-03-25 15:17:08,816 FATAL
      org.dspace.storage.rdbms.InitializeDatabase @ Caught exception:
[java] org.postgresql.util.PSQLException: Connection refused. Check
      that the hostname and port are correct and that the postmaster is
      accepting TCP/IP connections.
[java]     at
      org.postgresql.jdbc1.AbstractJdbc1Connection.openConnection(AbstractJd
      bclConnection.java:204)
[java]     at org.postgresql.Driver.connect(Driver.java:139)
```

it usually means you haven't yet added the relevant configuration parameter to your PostgreSQL configuration (see above), or perhaps you haven't restarted PostgreSQL after making the change. Also, make sure that the `db.username` and `db.password` properties are correctly set in `[dspace-source]/config/dspace.cfg`. An easy way to check that your DB is working OK over TCP/IP is to try this on the command line:

```
psql -U dspace -w -h localhost
```

Enter the `dspace` database password, and you should be dropped into the `psql` tool with a `dspace=>` prompt. Another common error looks like this:

```
[java] 2004-03-25 16:37:16,757 INFO
      org.dspace.storage.rdbms.InitializeDatabase @ Initializing Database
[java] 2004-03-25 16:37:17,139 WARN
      org.dspace.storage.rdbms.DatabaseManager @ Exception initializing DB
      pool
[java] java.lang.ClassNotFoundException: org.postgresql.Driver
[java]     at java.net.URLClassLoader$1.run(URLClassLoader.java:198)
[java]     at java.security.AccessController.doPrivileged(Native
      Method)
[java]     at
      java.net.URLClassLoader.findClass(URLClassLoader.java:186)
```

This means that the PostgreSQL JDBC driver is not present in `[dspace-source]/lib`. See above.

- **Tomcat doesn't shut down:** If you're trying to tweak Tomcat's configuration but nothing seems to make a difference to the error you're seeing, you might find that Tomcat hasn't been shutting down properly, perhaps because it's waiting for a stale connection to close gracefully which won't happen. To see if this is the case, try:

```
ps -ef | grep java
```

and look for Tomcat's Java processes. If they stay around after running Tomcat's `shutdown.sh` script, trying *kill_in them* (with `-9` if necessary), then starting Tomcat again.

- **Database connections don't work, or accessing DSpace takes forever:** If you find that when you try to access a DSpace Web page and your browser sits there connecting, or if the database connections fail, you might find that a 'zombie' database connection is hanging around preventing normal operation. To see if this is the case, try:

```
ps -ef | grep postgres
```

You might see some processes like this

```
dspace 16325 1997 0 Feb 14 ?        0:00 postgres: dspace dspace
```



```
127.0.0.1 idle in transaction
```

This is normal--DSpace maintains a 'pool' of open database connections, which are re-used to avoid the overhead of constantly opening and closing connections. If they're 'idle' it's OK; they're waiting to be used. However sometimes, if something went wrong, they might be stuck in the middle of a query, which seems to prevent other connections from operating, e.g.:

```
dspace 16325 1997 0 Feb 14 ? 0:00 postgres: dspace dspace
127.0.0.1 SELECT
```

This means the connection is in the middle of a *SELECT* operation, and if you're not using DSpace right that instant, it's probably a 'zombie' connection. If this is the case, try `_kill_`ing the process, and stopping and restarting Tomcat.

4. Upgrading a DSpace Installation

4.1. Upgrading from 1.5.x to 1.6

In the notes below *[dspace]* refers to the install directory for your existing DSpace installation, and *[dspace-source]* to the source directory for DSpace 1.5. Whenever you see these path references, be sure to replace them with the actual path names on your local system.

1. **Backup Your DSpace.** First, and foremost, make a complete backup of your system, including:
 - A snapshot of the database. `_To` have a "snapshot" of the PostgreSQL database, you need to shut it down during the backup. You should also have your regular Postgresql Backup output. `_`
 - The asset store (*[dspace]/assetstore* by default)
 - Your configuration files and customizations to DSpace (including any customized scripts).
2. **Download DSpace 1.6.** Retrieve the new DSpace 1.6 source code either as a download from <http://www.dspace.org/current-release/latest-release/> or check it out directly from the <http://scm.dspace.org/svn/repo/dspace/trunk>. If you downloaded DSpace do not unpack it on top of your existing installation. Refer to Chapter 3.3.3 Installation, Step 3 for unpacking directives.
3. **Stop Tomcat.** Take down your servlet container. For Tomcat, use the `$CATALINA/shutdown.sh` script. (Many installations will have a startup/shutdown script in the `/etc/init.d` or `/etc/rc.d` directories.
4. **Apply any customizations.** If you have made any local customizations to your DSpace installation they will need to be migrated over to the new DSpace. These are housed in one of the following places: JSPUI modifications: *[dspace-source]/dspace-jspui/dspace-jspui-webapp/src/main/webapp/_XMLUI* modifications: *[dspace-source]/dspace-xmlui/dspace-xmlui-webapp/src/main/webapp*
5. **Update Configuration Files.** Some of the parameters have change and some are new. Changes will be noted below:
 - The base url and oai urls property keys are set differently ****CHANGE****

```
# DSpace host name - should match base URL. Do not include port number
dspace.hostname = localhost

# DSpace base host URL. Include port number etc.
dspace.baseUrl = http://localhost:8080

# DSpace base URL. Include port number etc., but NOT trailing slash
# Change to xmlui if you wish to use the xmlui as the default, or remove
# "/jspui" and set webapp of your choice as the "ROOT" webapp in
# the servlet engine.
dspace.url = ${dspace.baseUrl}/xmlui
```




```
# The base URL of the OAI webapp (do not include /request).
dSPACE.oai.url = ${dSPACE.baseUrl}/oai
```

- The PostgreSQL database property key has changed ****CHANGE****

```
# URL for connecting to database
#db.url = ${default.db.url}
db.url = jdbc:postgresql://localhost:5432/dSPACE-services
```

- New email options:

```
# A comma separated list of hostnames that are allowed to refer browsers to email
forms.
# Default behaviour is to accept referrals only from dSPACE.hostname
#mail.allowed.referrers = localhost

# Pass extra settings to the Java mail library. Comma separated, equals sign between
# the key and the value.
#mail.extraproperties = mail.smtp.socketFactory.port=465, \
#
# mail.smtp.socketFactory.class=javax.net.ssl.SSLSocketFactory, \
# mail.smtp.socketFactory.fallback=false

# An option is added to disable the mailserver. By default, this property is set to
false
# By setting mail.server.disabled = true, DSpace will not send out emails.
# It will instead log the subject of the email which should have been sent
# This is especially useful for development and test environments where production
data is used when testing functionality.
#mail.server.disabled = false
```

- New Authorization levels and parameters. See Section 5.2.45 in Configuration for further information.

```
##### Authorization system configuration - Delegate ADMIN #####

# COMMUNITY ADMIN configuration
# subcommunities and collections
#core.authorization.community-admin.create-subelement = true
#core.authorization.community-admin.delete-subelement = true
# his community
#core.authorization.community-admin.policies = true
#core.authorization.community-admin.admin-group = true
# collections in his community
#core.authorization.community-admin.collection.policies = true
#core.authorization.community-admin.collection.template-item = true
#core.authorization.community-admin.collection.submitters = true
#core.authorization.community-admin.collection.workflows = true
#core.authorization.community-admin.collection.admin-group = true
# item owned by collections in his community
#core.authorization.community-admin.item.delete = true
#core.authorization.community-admin.item.withdraw = true
#core.authorization.community-admin.item.reinstatiate = true
#core.authorization.community-admin.item.policies = true
# also bundle...
#core.authorization.community-admin.item.create-bitstream = true
#core.authorization.community-admin.item.delete-bitstream = true
#core.authorization.community-admin.item-admin.cc-license = true

# COLLECTION ADMIN
#core.authorization.collection-admin.policies = true
#core.authorization.collection-admin.template-item = true
#core.authorization.collection-admin.submitters = true
#core.authorization.collection-admin.workflows = true
#core.authorization.collection-admin.admin-group = true
# item owned by his collection
#core.authorization.collection-admin.item.delete = true
#core.authorization.collection-admin.item.withdraw = true
#core.authorization.collection-admin.item.reinstatiate = true
```



```
#core.authorization.collection-admin.item.policies = true
# also bundle...
#core.authorization.collection-admin.item.create-bitstream = true
#core.authorization.collection-admin.item.delete-bitstream = true
#core.authorization.collection-admin.item-admin.cc-license = true

# ITEM ADMIN
#core.authorization.item-admin.policies = true
# also bundle...
#core.authorization.item-admin.create-bitstream = true
#core.authorization.item-admin.delete-bitstream = true
#core.authorization.item-admin.cc-license = true
```

- METS ingester has been revised. ****CHANGE****

```
# Option to make use of collection templates when using the METS ingester (default
is false)
mets.submission.useCollectionTemplate = false

# Crosswalk Plugins:
plugin.named.org.dspace.content.crosswalk.IngestionCrosswalk = \
  org.dspace.content.crosswalk.PREMISCrosswalk = PREMIS \
  org.dspace.content.crosswalk.OREIngestionCrosswalk = ore \
  org.dspace.content.crosswalk.NullIngestionCrosswalk = NIL \
  org.dspace.content.crosswalk.QDCCrosswalk = qdc \
  org.dspace.content.crosswalk.OAIDCIngestionCrosswalk = dc \
  org.dspace.content.crosswalk.DIMIngestionCrosswalk = dim

plugin.selfnamed.org.dspace.content.crosswalk.IngestionCrosswalk = \
  org.dspace.content.crosswalk.XSLTIngestionCrosswalk

plugin.named.org.dspace.content.crosswalk.DisseminationCrosswalk = \
  org.dspace.content.crosswalk.SimpleDCDisseminationCrosswalk = DC \
  org.dspace.content.crosswalk.SimpleDCDisseminationCrosswalk = dc \
  org.dspace.content.crosswalk.PREMISCrosswalk = PREMIS \
  org.dspace.content.crosswalk.METSDisseminationCrosswalk = METS \
  org.dspace.content.crosswalk.METSDisseminationCrosswalk = mets \
  org.dspace.content.crosswalk.OREDisseminationCrosswalk = ore \
  org.dspace.content.crosswalk.QDCCrosswalk = qdc \
  org.dspace.content.crosswalk.DIMDisseminationCrosswalk = dim
```

- Event Settings have had the following revision with the addition of 'harvester': ****CHANGE****

```
#### Event System Configuration ####

# default synchronous dispatcher (same behavior as traditional DSpace)
event.dispatcher.default.class = org.dspace.event.BasicDispatcher
event.dispatcher.default.consumers = search, browse, eperson, harvester
```

also:

```
# consumer to clean up harvesting data
event.consumer.harvester.class = org.dspace.harvest.HarvestConsumer
event.consumer.harvester.filters = Item+Delete
```

- New option for the Embargo of Thesis and Dissertations.

```
#### Embargo Settings ####
# DC metadata field to hold the user-supplied embargo terms
embargo.field.terms = SCHEMA.ELEMENT.QUALIFIER

# DC metadata field to hold computed "lift date" of embargo
embargo.field.lift = SCHEMA.ELEMENT.QUALIFIER

# string in terms field to indicate indefinite embargo
embargo.terms.open = forever
```



```
# implementation of embargo setter plugin - replace with local implementation if
applicable
plugin.single.org.dspace.embargo.EmbargoSetter =
  org.dspace.embargo.DefaultEmbargoSetter

# implementation of embargo lifter plugin - - replace with local implementation if
applicable
plugin.single.org.dspace.embargo.EmbargoLifter =
  org.dspace.embargo.DefaultEmbargoLifter
```

- New option for using the Batch Editing capabilities. See Section 5.2.46 in Configuration and also 8.10 in System Administration

```
#### Bulk metadata editor settings ####
# The delimiter used to separate values within a single field (defaults to a double
pipe ||)
# bulkedit.valueseparator = ||

# The delimiter used to separate fields (defaults to a comma for CSV)
# bulkedit.fieldseparator = ,

# A hard limit of the number of items allowed to be edited in one go in the UI
# (does not apply to the command line version)
# bulkedit.gui-item-limit = 20

# Metadata elements to exclude when exporting via the user interfaces, or when using
the
# command line version and not using the -a (all) option.
# bulkedit.ignore-on-export = dc.date.accessioned, dc.date.available, \
#                               dc.date.updated, dc.description.provenance
```

- Ability to hide metadata fields is now available.

```
##### Hide Item Metadata Fields #####
# Fields named here are hidden in the following places UNLESS the
# logged-in user is an Administrator:
# 1. XMLUI metadata XML view, and Item splash pages (long and short views).
# 2. JSPUI Item splash pages
# 3. OAI-PMH server, "oai_dc" format.
#   (NOTE: Other formats are _not_ affected.)
# To designate a field as hidden, add a property here in the form:
#   metadata.hide.SCHEMA.ELEMENT.QUALIFIER = true
#
# This default configuration hides the dc.description.provenance field,
# since that usually contains email addresses which ought to be kept
# private and is mainly of interest to administrators:
metadata.hide.dc.description.provenance = true
```

- New Choice Control and Authority Control options are available

```
## example of authority-controlled browse category - see authority control config
#webui.browse.index.5 = lcAuthor:metadataAuthority:dc.contributor.author:authority
```

And also:

```
##### Authority Control Settings #####

#plugin.named.org.dspace.content.authority.ChoiceAuthority = \
# org.dspace.content.authority.SampleAuthority = Sample, \
# org.dspace.content.authority.LCNameAuthority = LCNameAuthority, \
# org.dspace.content.authority.SHERPAROMEOPublisher = SRPublisher, \
# org.dspace.content.authority.SHERPAROMEOJournalTitle = SRJournalTitle

## This ChoiceAuthority plugin is automatically configured with every
## value-pairs element in input-forms.xml, namely:
##   common_identifiers, common_types, common_iso_languages
#plugin.selfnamed.org.dspace.content.authority.ChoiceAuthority = \
# org.dspace.content.authority.DCInputAuthority
```



```

## configure LC Names plugin
#lcname.url = http://alcm.e.oclc.org/srw/search/lcnaf

## configure SHERPA/RoMEO authority plugin
#sherpa.romeo.url = http://www.sherpa.ac.uk/romeo/api24.php

##
## This sets the default lowest confidence level at which a metadata value is
  included
## in an authority-controlled browse (and search) index. It is a symbolic
## keyword, one of the following values (listed in descending order):
##   accepted
##   uncertain
##   ambiguous
##   notfound
##   failed
##   rejected
##   novalue
##   unset
## See manual or org.dspace.content.authority.Choices source for descriptions.
authority.minconfidence = ambiguous

## demo: use LC plugin for author
#choices.plugin.dc.contributor.author = LCNameAuthority
#choices.presentation.dc.contributor.author = lookup
#authority.controlled.dc.contributor.author = true
##
## This sets the lowest confidence level at which a metadata value is included
## in an authority-controlled browse (and search) index. It is a symbolic
## keyword from the same set as for the default "authority.minconfidence"
#authority.minconfidence.dc.contributor.author = accepted

## Demo: publisher name lookup through SHERPA/RoMEO:
#choices.plugin.dc.publisher = SRPublisher
#choices.presentation.dc.publisher = suggest

## demo: journal title lookup, with ISSN as authority
#choices.plugin.dc.title.alternative = SRJournalTitle
#choices.presentation.dc.title.alternative = suggest
#authority.controlled.dc.title.alternative = true

## demo: use choice authority (without authority-control) to restrict dc.type on
  EditItemMetadata page
# choices.plugin.dc.type = common_types
# choices.presentation.dc.type = select

## demo: same idea for dc.language.iso
# choices.plugin.dc.language.iso = common_iso_languages
# choices.presentation.dc.language.iso = select

# Change number of choices shown in the select in Choices lookup popup
#xmlui.lookup.select.size = 12

```

- RSS Feeds now support Atom 1.0

```

##### Syndication Feed (RSS) Settings #####

# enable syndication feeds - links display on community and collection home pages
# (This setting is not used by XMLUI, as you enable feeds in your theme)
webui.feed.enable = false
# number of DSpace items per feed (the most recent submissions)
webui.feed.items = 4
# maximum number of feeds in memory cache
# value of 0 will disable caching
webui.feed.cache.size = 100
# number of hours to keep cached feeds before checking currency
# value of 0 will force a check with each request
webui.feed.cache.age = 48
# which syndication formats to offer

```



```

# use one or more (comma-separated) values from list:
# rss_0.90, rss_0.91, rss_0.92, rss_0.93, rss_0.94, rss_1.0, rss_2.0
webui.feed.formats = rss_1.0,rss_2.0,atom_1.0
# URLs returned by the feed will point at the global handle server (e.g. http://
hdl.handle.net/123456789/1)
# Set to true to use local server URLs (i.e. http://myserver.myorg/
handle/123456789/1)
webui.feed.localresolve = false

# Customize each single-value field displayed in the
# feed information for each item. Each of
# the below fields takes a *single* metadata field
#
# The form is <schema prefix>.<element>[.<qualifier>|.]*
webui.feed.item.title = dc.title
webui.feed.item.date = dc.date.issued

# Customise the metadata fields to show in the feed for each item's description.
# Elements will be displayed in the order that they are specified here.
#
# The form is <schema prefix>.<element>[.<qualifier>|.]*[(date)], ...
#
# Similar to the item display UI, the name of the field for display
# in the feed will be drawn from the current UI dictionary,
# using the key:
# "metadata.<field>"
#
# e.g.   "metadata.dc.title"
#        "metadata.dc.contributor.author"
#        "metadata.dc.date.issued"
webui.feed.item.description = dc.title, dc.contributor.author, \
                               dc.contributor.editor, \
                               dc.description.abstract, \
                               dc.description

# name of field to use for authors (Atom only) - repeatable
webui.feed.item.author = dc.contributor.author

# Customize the extra namespaced DC elements added to the item (RSS) or entry
# (Atom) element. These let you include individual metadata values in a
# structured format for easy extraction by the recipient, instead of (or in
# addition to) appending these values to the Description field.
## dc:creator value(s)
#webui.feed.item.dc.creator = dc.contributor.author
## dc:date value (may be contradicted by webui.feed.item.date)
#webui.feed.item.dc.date = dc.date.issued
## dc:description (e.g. for a distinct field that is ONLY the abstract)
#webui.feed.item.dc.description = dc.description.abstract

# Customize the image icon included with the site-wide feeds:
# Must be an absolute URL, e.g.
## webui.feed.logo.url = ${dSPACE.url}/themes/mysite/images/mysite-logo.png

```

- Opensearch Feature is new to DSpace

```

#### OpenSearch Settings ####
# NB: for result data formatting, OpenSearch uses Syndication Feed Settings
# so even if Syndication Feeds are not enabled, they must be configured
# enable open search
websvc.opensearch.enable = false
# context for html request URLs - change only for non-standard servlet mapping
websvc.opensearch.uicontext = simple-search
# context for RSS/Atom request URLs - change only for non-standard servlet mapping
websvc.opensearch.svccontext = open-search/
# present autodiscovery link in every page head
websvc.opensearch.autolink = true
# number of hours to retain results before recalculating
websvc.opensearch.validity = 48
# short name used in browsers for search service
# should be 16 or fewer characters
websvc.opensearch.shortname = DSpace
# longer (up to 48 characters) name

```



```
websvc.opensearch.longname = ${dspace.name}
# brief service description
websvc.opensearch.description = ${dspace.name} DSpace repository
# location of favicon for service, if any must be 16X16 pixels
websvc.opensearch.faviconurl = http://www.dspace.org/images/favicon.ico
# sample query - should return results
websvc.opensearch.samplequery = photosynthesis
# tags used to describe search service
websvc.opensearch.tags = IR DSpace
# result formats offered - use 1 or more comma-separated from: html,atom,rss
# NB: html is required for autodiscovery in browsers to function,
# and must be the first in the list if present
websvc.opensearch.formats = html,atom,rss
```

- Exposure of METS metadata can be now hidden.

```
# When exposing METS/MODS via OAI-PMH all metadata that can be mapped to MODS
# is exported. This includes description.provenance which can contain personal
# email addresses and other information not intended for public consumption. To
# hide this information set the following property to true
oai.mets.hide-provenance = true
```

- SWORD has added the following to accept MIME/types.

```
# A comma separated list of MIME types that SWORD will accept
sword.accepts = application/zip
```

- New OAI Harvesting Configuration settings are now available.

```
#####
#-----OAI HARVESTING CONFIGURATIONS-----#
#####
# These configs are only used by the OAI-ORE related functions #
#####

### Harvester settings

# Crosswalk settings; the {name} value must correspond to a declared ingestion
crosswalk
# harvester.oai.metadataformats.{name} = {namespace},{optional display name}
harvester.oai.metadataformats.dc = http://www.openarchives.org/OAI/2.0/oai_dc/,
Simple Dublin Core
harvester.oai.metadataformats.qdc = http://purl.org/dc/terms/, Qualified Dublin Core
harvester.oai.metadataformats.dim = http://www.dspace.org/xmlns/dspace/dim, DSpace
Intermediate Metadata

# Determines whether the harvester scheduling process should be started
# automatically when the DSpace webapp is deployed.
# default: false
harvester.autoStart=false

# How frequently the harvest scheduler checks the remote provider for updates,
# measured in minutes. The default value is 12 hours (or 720 minutes)
#harvester.harvestFrequency = 720

# How many harvest process threads the scheduler can spool up at once. Default value
is 3.
#harvester.maxThreads = 3

# How much time passes before a harvest thread is terminated. The termination
process
# waits for the current item to complete ingest and saves progress made up to that
point.
# Measured in hours. Default value is 24.
#harvester.threadTimeout = 24

# When harvesting an item that contains an unknown schema or field within a schema
what
```



```
# should the harvester do? Either add a new registry item for the field or schema,
# ignore
# the specific field or schema (importing everything else about the item), or fail
# with
# an error. The default value if undefined is: fail.
# Possible values: 'fail', 'add', or 'ignore'
harvester.unknownField = add
harvester.unknownSchema = fail

##### Usage Logging #####
solr.log.server = ${dspace.baseUrl}/solr/statistics
solr.spidersfile = ${dspace.dir}/config/spiders.txt
solr.dbfile = ${dspace.dir}/config/GeoLiteCity.dat
useProxies = true

statistics.items.dc.1=dc.identifier
statistics.items.dc.2=dc.date.accessioned
statistics.items.type.1=dcinput
statistics.items.type.2=date
statistics.default.start.datepick = 01/01/1977

statistics.item.authorization.admin=true
```

6. **Apply any Customizations.** If you have made any local customizations to your DSpace installation they will need to be migrated over to the new DSpace. Commonly these are customizations to the JSPUI or Manakin (XMLUI) interface pages.
7. **Build DSpace.** Run the following commands to compile DSpace.:

```
cd [dspace-source]/dspace/
mv package
```

You will find the result in `[dspace-source]/dspace/target/dspace-[version]-build.dir`. Inside this directory is the compiled binary distribution of DSpace.

8. **Update the database.** The database schema needs to be updated to accommodate changes to the database. SQL files contain the relevant updates are provided. Please note that if you have made any local customizations to the database schema, you should consult these updates and make sure they will work for you.

- For PostgreSQL: `psql -U [dspace-user] -f [dspace-source]/dspace/etc/postgres/schema_15_16.sql`
- For Oracle: Execute the upgrade script, e.g. with sqlplus, recording the output:
 - a. Start SQL*Plus with "`sqlplus [connect args]`"
 - b. Record the output: `SQL> spool 'upgrade.lst'`
 - c. Run the upgrade script `SQL> @[dspace-source]/dspace/etc/oracle/database_schema_15_16.sql``SQL> spool off`
 - d. Please note: The final few statements WILL FAIL. That is because you have run some queries and use the results to construct the statements to remove the constraints, manually---Oracle doesn't have any easy way to automate this (unless you know PL/SQL). So, look for the comment line beginning:

```
"--You need to remove the already in place constraints"
and follow the instructions in the actual SQL file.
Refer to the contents of the spool file "upgrade.lst" for
the output of the queries you'll need.
```

9. **Update DSpace.** Update the DSpace installed directory with the new code and libraries. Issue the following commands:

```
cd [dspace-source]/dspace/target/dspace-[version]-build.dir
```



```
ant -Dconfig=[dspace]/config/dspace.cfg update
```

10. Generate Browse and Search Indexes. It makes good policy to rebuild your search and browse indexes when upgrading to a new release. Almost every release has database changes and indexes can be affected by this. In the DSpace 1.6 release there is Authority Control features and those will need the indexes to be regenerated. To do this, run the following command from your DSpace install directory (as the dspace user): `[dspace]/bin/dspace index-init`

11. Deploy Web Applications. Copy the web applications files from your `[dspace]/webapps` directory to the subdirectory of your servlet container (e.g. tomcat): `cp -R [dspace]/webapps/* [tomcat]/webapps/`

4.2. Upgrading From 1.5 or 1.5.1 to 1.5.2

The changes in DSpace 1.5.2 do not include any database schema upgrades, and the upgrade should be straightforward.

In the notes below `[dspace]` refers to the install directory for your existing DSpace installation, and `[dspace-source]` to the source directory for DSpace 1.5. Whenever you see these path references, be sure to replace them with the actual path names on your local system.

1. **Backup your DSpace** First and foremost, make a complete backup of your system, including:

- A snapshot of the database
- The asset store (`[dspace]/assetstore` by default)
- Your configuration files and customizations to DSpace
- Your statistics scripts (`[dspace]/bin/stat*`) which contain customizable dates

2. **Download DSpace 1.5.2** Get the new DSpace 1.5.2 source code either as a download from <http://www.dspace.org/current-release/latest-release/> or check it out directly from the <http://scm.dspace.org/svn/repo/dspace/trunk>. If you downloaded DSpace do not unpack it on top of your existing installation.

3. **Build DSpace** Run the following commands to compile DSpace.

```
cd [dspace-source]/dspace/
mvn package
```

You will find the result in `[dspace-source]/dspace/target/dspace-1.5.2-build.dir/`; inside this directory is the compiled binary distribution of DSpace.

4. **Stop Tomcat** Take down your servlet container, for Tomcat use the `bin/shutdown.sh` script.

5. **Apply any customizations** If you have made any local customizations to your DSpace installation they will need to be migrated over to the new DSpace. Commonly these modifications are made to "JSP" pages located inside the `[dspace 1.4.2]/jsp/local` directory. These should be moved `[dspace-source]/dspace/modules/jspui/src/main/webapp/` in the new build structure. See Customizing the JSP Pages for more information.

6. **Update DSpace** Update the DSpace installed directory with new code and libraries. Inside the `[dspace-source]/dspace/target/dspace-1.5-build.dir/` directory run:

```
cd [dspace-source]/dspace/target/dspace-1.5-build.dir/
ant -Dconfig=[dspace]/config/dspace.cfg update
```

7. **Update configuration files** This ant target preserves existing files in `[dspace]/config` _ and will copy any new configuration files in place. If an existing file prevents copying the new file in place, the new file will have the suffix `_new`, for example `[dspace]/local/dspace.cfg.new`. Note: there is also a configuration option `-Doverwrite=true` which will instead copy the conflicting target files to `*.old` suffixes and



overwrite target file then with the new file (essentially the opposite) this is beneficial for developers and those who use the `[dspace-source]/dspace/config` to maintain their changes.

```
cd [dspace-source]/dspace/target/dspace-1.5-build.dir/
ant -Dconfig=[dspace]/config/dspace.cfg update_configs
```

You must then verify that you've merged and differenced in the `[dspace]/config/*/.new` files into your configuration. Some of the new parameters you should look out for in `dspace.cfg` include:

- New option to restrict the expose of private items. The following needs to be added to `dspace.cfg`:

```
##### Restricted item visibility settings ###
# By default RSS feeds, OAI-PMH and subscription emails will include ALL items
# regardless of permissions set on them.
#
# If you wish to only expose items through these channels where the ANONYMOUS
# user is granted READ permission, then set the following options to false
#harvest.includerestricted.rss = true
#harvest.includerestricted.oai = true
#harvest.includerestricted.subscription = true
```

- Special groups for LDAP and password authentication.

```
##### Password users group #####
# If required, a group name can be given here, and all users who log in
# using the DSpace password system will automatically become members of
# this group. This is useful if you want a group made up of all internal
# authenticated users.
#password.login.specialgroup = group-name

##### LDAP users group #####
# If required, a group name can be given here, and all users who log in
# to LDAP will automatically become members of this group. This is useful
# if you want a group made up of all internal authenticated users.
#ldap.login.specialgroup = group-name
```

- new option for case insensitivity in browse tables.

```
# By default, the display of metadata in the browse indexes is case sensitive
# So, you will get separate entries for the terms
#
#   Olive oil
#   olive oil
#
# However, clicking through from either of these will result in the same set of items
# (ie. any item that contains either representation in the correct field).
#
# Uncommenting the option below will make the metadata items case-insensitive. This
# will
# result in a single entry in the example above. However the value displayed may be
# either 'Olive oil'
# or 'olive oil' - depending on what representation was present in the first item
# indexed.
#
# If you care about the display of the metadata in the browse index - well, you'll
# have to go and
# fix the metadata in your items.
#
# webui.browse.metadata.case-insensitive = true
```

- New usage event handler for collecting statistics:

```
### Usage event settings ###
# The usage event handler to call. The default is the "passive" handler, which
# ignores events.
```



```
# plugin.single.org.dspace.app.statistics.AbstractUsageEvent = \
#   org.dspace.app.statistics.PassiveUsageEvent
```

- The location where sitemaps are stored is now configurable.

```
##### Sitemap settings #####
# the directory where the generated sitemaps are stored
sitemap.dir = ${dspace.dir}/sitemaps
```

- MARC 21 ordering should now be used as default. Unless you have it set already, or you have it set to a different value, the following should be set:

```
plugin.named.org.dspace.sort.OrderFormatDelegate =
  org.dspace.sort.OrderFormatTitleMarc21=title
```

- Hierarchical LDAP support.

```
##### Hierarchical LDAP Settings #####
# If your users are spread out across a hierarchical tree on your
# LDAP server, you will need to use the following stackable authentication
# class:
#   plugin.sequence.org.dspace.authenticate.AuthenticationMethod = \
#     org.dspace.authenticate.LDAPHierarchicalAuthentication
#
# You can optionally specify the search scope. If anonymous access is not
# enabled on your LDAP server, you will need to specify the full DN and
# password of a user that is allowed to bind in order to search for the
# users.
#
# This is the search scope value for the LDAP search during
# autoregistering. This will depend on your LDAP server setup.
# This value must be one of the following integers corresponding
# to the following values:
# object scope : 0
# one level scope : 1
# subtree scope : 2
#ldap.search_scope = 2
#
# The full DN and password of a user allowed to connect to the LDAP server
# and search for the DN of the user trying to log in. If these are not specified,
# the initial bind will be performed anonymously.
#ldap.search.user = cn=admin,ou=people,o=myu.edu
#ldap.search.password = password
#
# If your LDAP server does not hold an email address for a user, you can use
# the following field to specify your email domain. This value is appended
# to the netid in order to make an email address. E.g. a netid of 'user' and
# ldap.netid_email_domain as '@example.com' would set the email of the user
# to be 'user@example.com'
#ldap.netid_email_domain = @example.com
```

- Shibboleth authentication support.

```
##### Shibboleth Authentication Configuration Settings #####
# Check https://mams.melcoe.mq.edu.au/zoep/mams/pubs/Installation/dspace15/view
# for installation detail.
#
#   org.dspace.authenticate.ShibAuthentication
#
# DSpace requires email as user's credential. There are 2 ways of providing
# email to DSpace:
# 1) by explicitly specifying to the user which attribute (header)
#    carries the email address.
# 2) by turning on the user-email-using-tomcat=true which means
#    the software will try to acquire the user's email from Tomcat
# The first option takes PRECEDENCE when specified. Both options can
# be enabled to allow fallback.
```



```
# this option below specifies that the email comes from the mentioned header.
# The value is CASE-Sensitive.
authentication.shib.email-header = MAIL

# optional. Specify the header that carries user's first name
# this is going to be used for creation of new-user
authentication.shib.firstname-header = SHIB-EP-GIVENNAME

# optional. Specify the header that carries user's last name
# this is used for creation of new user
authentication.shib.lastname-header = SHIB-EP-SURNAME

# this option below forces the software to acquire the email from Tomcat.
authentication.shib.email-use-tomcat-remote-user = true

# should we allow new users to be registered automatically
# if the IdP provides sufficient info (and user not exists in DSpace)
authentication.shib.autoregister = true

# this header here specifies which attribute that is responsible
# for providing user's roles to DSpace. When not specified, it is
# defaulted to 'Shib-EP-UnscopedAffiliation'. The value is specified
# in AAP.xml (Shib 1.3.x) or attribute-filter.xml (Shib 2.x).
# The value is CASE-Sensitive. The values provided in this
# header are separated by semi-colon or comma.
# authentication.shib.role-header = Shib-EP-UnscopedAffiliation

# when user is fully authN on IdP but would not like to release
# his/her roles to DSpace (for privacy reason?), what should be
# the default roles be given to such users?
# The values are separated by semi-colon or comma
# authentication.shib.default-roles = Staff, Walk-ins

# The following mappings specify role mapping between IdP and Dspace.
# the left side of the entry is IdP's role (prefixed with
# "authentication.shib.role.") which will be mapped to
# the right entry from DSpace. DSpace's group as indicated on the
# right entry has to EXIST in DSpace, otherwise user will be identified
# as 'anonymous'. Multiple values on the right entry should be separated
# by comma. The values are CASE-Sensitive. Heuristic one-to-one mapping
# will be done when the IdP groups entry are not listed below (i.e.
# if "X" group in IdP is not specified here, then it will be mapped
# to "X" group in DSpace if it exists, otherwise it will be mapped
# to simply 'anonymous')
#
# Given sufficient demand, future release could support regex for the mapping
# special characters need to be escaped by \
authentication.shib.role.Senior\ Researcher = Researcher, Staff
authentication.shib.role.Librarian = Administrator
```

- DOI and handle identifiers can now be rendered in the JSPUI.

```
# When using "resolver" in webui.itemdisplay to render identifiers as resolvable
# links, the base URL is taken from <code>webui.resolver.<n>.baseurl</code>
# where <code>webui.resolver.<n>.baseurl</code> matches the urn specified in the
# metadata value.
# The value is appended to the "baseurl" as is, so the baseurl need to end with slash
# almost in any case.
# If no urn is specified in the value it will be displayed as simple text.
#
#webui.resolver.1.urn = doi
#webui.resolver.1.baseurl = http://dx.doi.org/
#webui.resolver.2.urn = hdl
#webui.resolver.2.baseurl = http://hdl.handle.net/
#
# For the doi and hdl urn defaults values are provided, respectively http://dx.doi.org
# and
# http://hdl.handle.net are used.
#
# If a metadata value with style: "doi", "handle" or "resolver" matches a URL
# already, it is simply rendered as a link with no other manipulation.
```



In configuration sections such as `webui.itemdisplay.default`, values can be changed from (e.g.) `metadata.dc.identifier.doi` to `metadata.doi.dc.identifier.doi`

- The whole of the SWORD configuration has changed. The SWORD section must be removed and replaced with

```

#-----#
#-----SWORD SPECIFIC CONFIGURATIONS-----#
#-----#
# These configs are only used by the SWORD interface      #
#-----#

# tell the SWORD METS implementation which package ingester to use
# to install deposited content. This should refer to one of the
# classes configured for:
#
# plugin.named.org.dspace.content.packager.PackageIngester
#
# The value of sword.mets-ingester.package-ingester tells the
# system which named plugin for this interface should be used
# to ingest SWORD METS packages
#
# The default is METS
#
# sword.mets-ingester.package-ingester = METS

# Define the metadata type EPDCX (EPrints DC XML)
# to be handled by the SWORD crosswalk configuration
#
mets.submission.crosswalk.EPDCX = SWORD

# define the stylesheet which will be used by the self-named
# XSLTIngestionCrosswalk class when asked to load the SWORD
# configuration (as specified above). This will use the
# specified stylesheet to crosswalk the incoming SWAP metadata
# to the DIM format for ingestion
#
crosswalk.submission.SWORD.stylesheet = crosswalks/sword-swap-ingest.xsl

# The base URL of the SWORD deposit. This is the URL from
# which DSpace will construct the deposit location urls for
# collections.
#
# The default is {dspace.url}/sword/deposit
#
# In the event that you are not deploying DSpace as the ROOT
# application in the servlet container, this will generate
# incorrect URLs, and you should override the functionality
# by specifying in full as below:
#
# sword.deposit.url = http://www.myu.ac.uk/sword/deposit

# The base URL of the SWORD service document. This is the
# URL from which DSpace will construct the service document
# location urls for the site, and for individual collections
#
# The default is {dspace.url}/sword/servicedocument
#
# In the event that you are not deploying DSpace as the ROOT
# application in the servlet container, this will generate
# incorrect URLs, and you should override the functionality
# by specifying in full as below:
#
# sword.servicedocument.url = http://www.myu.ac.uk/sword/servicedocument

# The base URL of the SWORD media links. This is the URL
# which DSpace will use to construct the media link urls
# for items which are deposited via sword
#
# The default is {dspace.url}/sword/media-link

```



```
#
# In the event that you are not deploying DSpace as the ROOT
# application in the servlet container, this will generate
# incorrect URLs, and you should override the functionality
# by specifying in full as below:
#
# sword.media-link.url = http://www.myu.ac.uk/sword/media-link
#
# The URL which identifies the sword software which provides
# the sword interface. This is the URL which DSpace will use
# to fill out the atom:generator element of its atom documents.
#
# The default is:
#
# http://www.dspace.org/ns/sword/1.3.1
#
# If you have modified your sword software, you should change
# this URI to identify your own version. If you are using the
# standard dspace-sword module you will not, in general, need
# to change this setting
#
# sword.generator.url = http://www.dspace.org/ns/sword/1.3.1
#
# The metadata field in which to store the updated date for
# items deposited via SWORD.
#
sword.updated.field = dc.date.updated
#
# The metadata field in which to store the value of the slug
# header if it is supplied
#
sword.slug.field = dc.identifier.slug
#
# the accept packaging properties, along with their associated
# quality values where appropriate.
#
# Global settings; these will be used on all DSpace collections
#
sword.accept-packaging.METSDSpaceSIP.identifier =
http://purl.org/net/sword-types/METSDSpaceSIP
sword.accept-packaging.METSDSpaceSIP.q = 1.0
#
# Collection Specific settings: these will be used on the collections
# with the given handles
#
# sword.accept-packaging.[handle].METSDSpaceSIP.identifier =
http://purl.org/net/sword-types/METSDSpaceSIP
# sword.accept-packaging.[handle].METSDSpaceSIP.q = 1.0
#
# Should the server offer up items in collections as sword deposit
# targets. This will be effected by placing a URI in the collection
# description which will list all the allowed items for the depositing
# user in that collection on request
#
# NOTE: this will require an implementation of deposit onto items, which
# will not be forthcoming for a short while
#
sword.expose-items = false
#
# Should the server offer as the default the list of all Communities
# to a Service Document request. If false, the server will offer
# the list of all collections, which is the default and recommended
# behaviour at this stage.
#
# NOTE: a service document for Communities will not offer any viable
# deposit targets, and the client will need to request the list of
# Collections in the target before deposit can continue
#
sword.expose-communities = false
#
# The maximum upload size of a package through the sword interface,
# in bytes
```



```

#
# This will be the combined size of all the files, the metadata and
# any manifest data.  It is NOT the same as the maximum size set
# for an individual file upload through the user interface.  If not
# set, or set to 0, the sword service will default to no limit.
#
sword.max-upload-size = 0

# Should DSpace store a copy of the original sword deposit package?
#
# NOTE: this will cause the deposit process to run slightly slower,
# and will accelerate the rate at which the repository consumes disk
# space.  BUT, it will also mean that the deposited packages are
# recoverable in their original form.  It is strongly recommended,
# therefore, to leave this option turned on
#
# When set to "true", this requires that the configuration option
# "upload.temp.dir" above is set to a valid location
#
sword.keep-original-package = true

# The bundle name that SWORD should store incoming packages under if
# sword.keep-original-package is set to true.  The default is "SWORD"
# if not value is set
#
# sword.bundle.name = SWORD

# Should the server identify the sword version in deposit response?
#
# It is recommended to leave this enabled.
#
sword.identify-version = true

# Should we support mediated deposit via sword?  Enabled, this will
# allow users to deposit content packages on behalf of other users.
#
# See the SWORD specification for a detailed explanation of deposit
# On-Behalf-Of another user
#
sword.on-behalf-of.enable = true

# Configure the plugins to process incoming packages.  The form of this
# configuration is as per the Plugin Manager's Named Plugin documentation:
#
# plugin.named.[interface] = [implementation] = [package format identifier] \
#
# Package ingesters should implement the SWORDIngester interface, and
# will be loaded when a package of the format specified above in:
#
# sword.accept-packaging.[package format].identifier = [package format identifier]
#
# is received.
#
# In the event that this is a simple file deposit, with no package
# format, then the class named by "SimpleFileIngester" will be loaded
# and executed where appropriate.  This case will only occur when a single
# file is being deposited into an existing DSpace Item
#
plugin.named.org.dspace.sword.SWORDIngester = \
  org.dspace.sword.SWORDMETSIngester =
http://purl.org/net/sword-types/METSDSpaceSIP \
  org.dspace.sword.SimpleFileIngester = SimpleFileIngester

```

1. **Restart Tomcat** Restart your servlet container, for Tomcat use the *bin/startup.sh* script.

4.3. Upgrading From 1.4.2 to 1.5

The changes in DSpace 1.5 are significant and wide spread involving database schema upgrades, code restructuring, completely new user and programatic interfaces, and new build system.



In the notes below *[dspace]* refers to the install directory for your existing DSpace installation, and *[dspace-source]* to the source directory for DSpace 1.5. Whenever you see these path references, be sure to replace them with the actual path names on your local system.

1. **Backup your DSpace** First and foremost, make a complete backup of your system, including:

- A snapshot of the database
- The asset store (*[dspace]/assetstore* by default)
- Your configuration files and customizations to DSpace
- Your statistics scripts (*[dspace]/bin/stat**) which contain customizable dates

2. **Download DSpace 1.5** Get the new DSpace 1.5 source code either as a download from # or check it out directly from the #. If you downloaded DSpace do not unpack it on top of your existing installation.

3. **Build DSpace** The build process has radically changed for DSpace 1.5. With this new release the build system has moved to a maven-based system enabling the various projects (JSPUI, XMLUI, OAI, and Core API) into separate projects. See the Installation section for more information on building DSpace using the new maven-based build system. Run the following commands to compile DSpace.

```
cd [dspace-source]/dspace/;
mvn package
```

You will find the result in *[dspace-source]/dspace/target/dspace-1.5-build.dir/*; inside this directory is the compiled binary distribution of DSpace.

4. **Stop Tomcat** Take down your servlet container, for Tomcat use the *bin/shutdown.sh* script.

5. **Updatedspace.cfg** Several new parameters need to be added to your *[dspace]/config/dspace.cfg*. While it is advisable to start with a fresh *DSpace 1.5 _dspace.cfg* configuration file_ here are the minimum set of parameters that need to be added to an old *DSpace 1.4.2 configuration*.

```
##### Stackable Authentication Methods #####
#
# Stack of authentication methods
# (See org.dspace.authenticate.AuthenticationManager)
# Note when upgrading you should remove the parameter:
# plugin.sequence.org.dspace.eperson.AuthenticationMethod
plugin.sequence.org.dspace.authenticate.AuthenticationMethod = \
    org.dspace.authenticate.PasswordAuthentication

##### JSPUI item sytle plugin #####
#
# Specify which strategy use for select the style for an item
plugin.single.org.dspace.app.webui.util.StyleSelection = \
    org.dspace.app.webui.util.CollectionStyleSelection

##### Browse Configuration #####
#
# The following configuration will mimic the previous
# behavior exhibited by DSpace 1.4.2. For alternative
# configurations see the manual.

# Browse indexes
webui.browse.index.1 = dateissued:item:dateissued
webui.browse.index.2 = author:metadata:dc.contributor.*:text
webui.browse.index.3 = title:item:title
webui.browse.index.4 = subject:metadata:dc.subject.*:text

# Sorting options
webui.itemlist.sort-option.1 = title:dc.title:title
webui.itemlist.sort-option.2 = dateissued:dc.date.issued:date
```



```
webui.itemlist.sort-option.3 =
  dateaccessioned:dc.date.accessioned:date

# Recent submissions
recent.submissions.count = 5

# Itemmapper browse index
itemmap.author.index = author

# Recent submission processor plugins
plugin.sequence.org.dspace.plugin.CommunityHomeProcessor = \
  org.dspace.app.webui.components.RecentCommunitySubmissions
plugin.sequence.org.dspace.plugin.CollectionHomeProcessor = \
  org.dspace.app.webui.components.RecentCollectionSubmissions

#### Content Inline Disposition Threshold ####
#
# Set the max size of a bitstream that can be served inline
# Use -1 to force all bitstream to be served inline
# webui.content_disposition_threshold = -1
webui.content_disposition_threshold = 8388608

#### Event System Configuration ####
#
# default synchronous dispatcher (same behavior as traditional
  DSpace)
event.dispatcher.default.class = org.dspace.event.BasicDispatcher
event.dispatcher.default.consumers = search, browse, eperson

# consumer to maintain the search index
event.consumer.search.class = org.dspace.search.SearchConsumer
event.consumer.search.filters =
  Item|Collection|Community|Bundle+Create|Modify|Modify_Metadata|Delete:
  Bundle+Add|Remove

# consumer to maintain the browse index
event.consumer.browse.class = org.dspace.browse.BrowseConsumer
event.consumer.browse.filters =
  Item+Create|Modify|Modify_Metadata:Collection+Add|Remove

# consumer related to EPerson changes
event.consumer.eperson.class = org.dspace.eperson.EPersonConsumer
event.consumer.eperson.filters = EPerson+Create
```

6. **Addxmlui.xconfManakin configuration** The new Manakin user interface available with DSpace 1.5 requires an extra configuration file that you will need to manually copy it over to your configuration directory.

```
cp [dspace-source]/dspace/config/xmlui.xconf
  [dspace]/config/xmlui.xconf
```

7. **Additem-submission.xmlanditem-submission.dtdconfigurable submission configuration** The new configurable submission system that enables an administrator to re-arrange, or add/remove item submission steps requires this configuration file. You need to manually copy it over to your configuration directory.

```
cp [dspace-source]/dspace/config/item-submission.xml
  [dspace]/config/item-submission.xml

cp [dspace-source]/dspace/config/item-submission.dtd
  [dspace]/config/item-submission.dtd
```

8. **Add newinput-forms.xmlandinput-forms.dtdconfigurable submission configuration** The input-forms.xml now has an included dtd reference to support validation. You'll need to merge in your changes to both file/and or copy them into place.

```
cp [dspace-source]/dspace/config/input-forms.xml
  [dspace]/config/input-forms.xml
```




```
cp [dspace-source]/dSPACE/config/input-forms.dtd
[dSPACE]/config/inputforms.dtd
```

9. **Addsword-swap-ingest.xsl and xhtml-head-item.properties crosswalk files** New crosswalk files are required to support SWORD and the inclusion of metadata into the head of items.

```
cp [dSPACE-source]/dSPACE/config/crosswalks/sword-swap-ingest.xsl
[dSPACE]/config/crosswalks/sword-swap-ingest.xsl

cp
[dSPACE-source]/dSPACE/config/crosswalks/xhtml-head-item.properties
[dSPACE]/config/crosswalks/xhtml-head-item.properties
```

10. **Add registration_notify email files** A new configuration option (*registration.notify = you@your-email.com*) can be set to send a notification email whenever a new user registers to use your DSpace. The email template for this email needs to be copied.

```
cp [dSPACE-source]/dSPACE/config/emails/registration_notify
[dSPACE]/config/emails/registration_notify
```

11. **Update the database** The database schema needs updating. SQL files contain the relevant updates are provided, note if you have made any local customizations to the database schema you should consult these updates and make sure they will work for you.

- For PostgreSQL *psql -U [dSPACE-user] -f [dSPACE-source]/dSPACE/etc/database_schema_14-15.sql [database-name]*
- For Oracle *[dSPACE-source]/dSPACE/etc/oracle/database_schema_14-15.sql* contains the commands necessary to upgrade your database schema on oracle.

12. **Apply any customizations** If you have made any local customizations to your DSpace installation they will need to be migrated over to the new DSpace. Commonly these modifications are made to "JSP" pages located inside the *[dSPACE 1.4.2]/jsp/local* directory. These should be moved *[dSPACE-source]/dSPACE/modules/jspui/src/main/webapp/* in the new build structure. See Customizing the JSP Pages for more information.

13. **Update DSpace** Update the DSpace installed directory with new code and libraries. Inside the *[dSPACE-source]/dSPACE/target/dSPACE-1.5-build.dir/* directory run:

```
cd [dSPACE-source]/dSPACE/target/dSPACE-1.5-build.dir/;
ant -Dconfig=[dSPACE]/config/dSPACE.cfg update
```

14. **Update the Metadata Registry** New Metadata Registry updates are required to support SWORD.

```
cp [dSPACE-source]/dSPACE/config/registries/sword-metadata.xml
[dSPACE]/config/registries/sword-metadata.xml;

[dSPACE]/bin/dsrun org.dSPACE.administer.MetadataImporter -f
[dSPACE]/config/registries/sword-metadata.xml
```

15. **Rebuild browse and search indexes** One of the major new features of DSpace 1.5 is the browse system which necessitates that the indexes be recreated. To do this run the following command from your DSpace installed directory:

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

16. **Update statistics scripts** The statistics scripts have been rewritten for DSpace 1.5. Prior to 1.5 they were written in Perl, but have been rewritten in Java to avoid having to install Perl. First, make a note of the dates you have specified in your statistics scripts for the statistics to run from. You will find these in



`[dSPACE]/bin/stat-initial`, as `$start_year` and `$start_month`. Note down these values. Copy the new stats scripts:

```
cp [dSPACE-source]/dSPACE/bin/stat* [dSPACE]/bin/
```

Then edit your statistics configuration file with the start details. Add the following to `[dSPACE]/conf/dstat.cfg` *# the year and month to start creating reports from# - year as four digits (e.g. 2005)# - month as a number (e.g. January is 1, December is 12)*
`start.year = 2005`
`start.month = 1`
 Replace '2005' and '1' as with the values you noted down. `dstat.cfg` also used to contain the hostname and service name as displayed at the top of the statistics. These values are now taken from `dSPACE.cfg` so you can remove `host.name` and `host.url` from `dstat.cfg` if you wish. The values now used are `dSPACE.hostname` and `dSPACE.name` from `dSPACE.cfg`

17. Deploy webapplications Copy the webapplications files from your `[dSPACE]/webapps` directory to the subdirectory of your servlet container (e.g. Tomcat):

```
cp [dSPACE]/webapps/* [tomcat]/webapps/
```

18. Restart Tomcat Restart your servlet container, for Tomcat use the `bin/startup.sh` script.

4.4. Upgrading From 1.4.1 to 1.4.2

See Upgrading From 1.4 to 1.4.x; the same instructions apply.

4.5. Upgrading From 1.4 to 1.4.x

The changes in 1.4.x releases are only code and configuration changes so the update is simply a matter of rebuilding the wars and slight changes to your config file.

In the notes below `[dSPACE]` refers to the install directory for your existing DSpace installation, and `[dSPACE-1.4.x-source]` to the source directory for DSpace 1.4.x. Whenever you see these path references, be sure to replace them with the actual path names on your local system.

1. Get the new DSpace 1.4.x source code from <http://sourceforge.net/projects/dSPACE/> and unpack it somewhere. Do not unpack it on top of your existing installation!!
2. Copy the PostgreSQL driver JAR to the source tree. For example:

```
cd [dSPACE]/lib
cp postgresql.jar [dSPACE-1.4.x-source]/lib
```

3. **Note:** Licensing conditions for the `handle.jar` file have changed. As a result, the latest version of the `handle.jar` file is not included in this distribution. It is recommended you read the http://www.handle.net/upgrade_6-2_DSPACE.html and decide whether you wish to update your installation's `handle.jar`. If you decide to update, you should replace the existing `handle.jar` in `[dSPACE-1.4.x-source]/lib` with the new version.
4. Take down Tomcat (or whichever servlet container you're using).
5. A new configuration item `webui.html.max-depth-guess` has been added to avoid infinite URL spaces. Add the following to the `dSPACE.cfg` `##### Multi-file HTML document/site settings #####`
`#`
6. When serving up composite HTML items, how deep can the request be for us to
7. serve up a file with the same name?
`#`
8. e.g. if we receive a request for "foo/bar/index.html"



9. and we have a bitstream called just "index.html"
10. we will serve up that bitstream for the request if
webui.html.max-depth-guess
11. is 2 or greater. If webui.html.max-depth-guess is 1 or less, we
would not
12. serve that bitstream, as the depth of the file is greater.
#
13. If webui.html.max-depth-guess is zero, the request filename and
path must
14. always exactly match the bitstream name. Default value is 3.

webui.html.max-depth-guess = 3
If " rel="nofollow" linktype="raw" wikidestination="file:#### Multi-file HTML document/site settings

#
15. When serving up composite HTML items, how deep can the request be
for us to
16. serve up a file with the same name?
#
17. e.g. if we receive a request for "foo/bar/index.html"
18. and we have a bitstream called just "index.html"
19. we will serve up that bitstream for the request if
webui.html.max-depth-guess
20. is 2 or greater. If webui.html.max-depth-guess is 1 or less, we
would not
21. serve that bitstream, as the depth of the file is greater.
#
22. If webui.html.max-depth-guess is zero, the request filename and
path must
23. always exactly match the bitstream name. Default value is 3.

webui.html.max-depth-guess = 3
If " originalalias="file:#### Multi-file HTML document/site settings #####
#
24. When serving up composite HTML items, how deep can the request be
for us to
25. serve up a file with the same name?
#
26. e.g. if we receive a request for "foo/bar/index.html"
27. and we have a bitstream called just "index.html"
28. we will serve up that bitstream for the request if
webui.html.max-depth-guess
29. is 2 or greater. If webui.html.max-depth-guess is 1 or less, we



- would not
30. serve that bitstream, as the depth of the file is greater.
#
31. If `webui.html.max-depth-guess` is zero, the request filename and path must
32. always exactly match the bitstream name. Default value is 3.

`webui.html.max-depth-guess = 3`
If" >[file:####](#) Multi-file HTML document/site settings #####
#
33. When serving up composite HTML items, how deep can the request be for us to
34. serve up a file with the same name?
#
35. e.g. if we receive a request for "foo/bar/index.html"
36. and we have a bitstream called just "index.html"
37. we will serve up that bitstream for the request if
`webui.html.max-depth-guess`
38. is 2 or greater. If `webui.html.max-depth-guess` is 1 or less, we would not
39. serve that bitstream, as the depth of the file is greater.
#
40. If `webui.html.max-depth-guess` is zero, the request filename and path must
41. always exactly match the bitstream name. Default value is 3.

`webui.html.max-depth-guess = 3`

<code>file:%3Cdiv%20class=]</code>

<code>file:%3Cdiv%20class=]</code>

If `webui.html.max-depth-guess` is not present in `dspace.cfg` the default value is used. If archiving entire web sites or deeply nested HTML documents it is advisable to change the default to a higher value more suitable for these types of materials.

1. Your 'localized' JSPs (those in `jsp/local`) now need to be maintained in the `source` directory. If you have locally modified JSPs in your `[dSPACE]/jsp/local` directory, you will need to merge the changes in the new 1.4.x versions into your locally modified ones. You can use the `diff` command to compare your JSPs against the 1.4.x versions to do this. You can also check against the <http://dSPACE.cvs.sourceforge.net/dSPACE/>.
2. In `[dSPACE-1.4.x-source]` run:

```
ant -Dconfig= [dSPACE]/config/dSPACE.cfg update
```

3. Copy the `.war` Web application files in `[dSPACE-1.4.x-source]/build` to the `webapps` sub-directory of your servlet container (e.g. Tomcat). e.g.:

```
cp [dSPACE-1.4.x-source]/build/*.war  
[tomcat]/webapps
```



If you're using Tomcat, you need to delete the directories corresponding to the old *.war* files. For example, if *dspace.war* is installed in *[tomcat]/webapps/dspace.war*, you should delete the *[tomcat]/webapps/dspace* directory. Otherwise, Tomcat will continue to use the old code in that directory.

4. Restart Tomcat.

4.6. Upgrading From 1.3.2 to 1.4.x

1. First and foremost, **make a complete backup** of your system, including:
 - A snapshot of the database
 - The asset store (*[dspace]/assetstore* by default)
 - Your configuration files and localized JSPs
2. Download the <http://sourceforge.net/projects/dspace/> and unpack it in a suitable location (*not* over your existing DSpace installation or source tree!)
3. Copy the PostgreSQL driver JAR to the source tree. For example:

```
cd [dspace]/lib
cp postgresql.jar [dspace-1.4.x-source]/lib
```

4. **Note:** Licensing conditions for the *handle.jar* file have changed. As a result, the latest version of the *handle.jar* file is not included in this distribution. It is recommended you read the http://www.handle.net/upgrade_6-2_DSpace.html and decide whether you wish to update your installation's *handle.jar*. If you decide to update, you should replace the existing *handle.jar* in *[dspace-1.4.x-source]/lib* with the new version.
5. Take down Tomcat (or whichever servlet container you're using).
6. Your DSpace configuration will need some updating:
 - In *dspace.cfg*, paste in the following lines for the new stackable authentication feature, the new method for managing Media Filters, and the Checksum Checker.

```
##### Stackable Authentication Methods #####
# Stack of authentication methods
# (See org.dspace.eperson.AuthenticationManager)
plugin.sequence.org.dspace.eperson.AuthenticationMethod = \
    org.dspace.eperson.PasswordAuthentication

##### Example of configuring X.509 authentication
##### (to use it, add org.dspace.eperson.X509Authentication to stack)

## method 1, using keystore
#authentication.x509.keystore.path = /var/local/tomcat/conf/keystore
#authentication.x509.keystore.password = changeit

## method 2, using CA certificate
#authentication.x509.ca.cert = ${dspace.dir}/config/mitClientCA.der

## Create e-persons for unknown names in valid certificates?
#authentication.x509.autoregister = true

##### Media Filter plugins (through PluginManager) #####

plugin.sequence.org.dspace.app.mediafilter.MediaFilter = \
    org.dspace.app.mediafilter.PDFFilter,
    org.dspace.app.mediafilter.HTMLFilter, \
    org.dspace.app.mediafilter.WordFilter,
    org.dspace.app.mediafilter.JPGFilter
```



```
# to enable branded preview: remove last line above, and uncomment 2
# lines below
# org.dspace.app.mediafilter.WordFilter,
# org.dspace.app.mediafilter.JPEGFilter, \
# org.dspace.app.mediafilter.BrandedPreviewJPEGFilter

filter.org.dspace.app.mediafilter.PDFFilter.inputFormats = Adobe PDF
filter.org.dspace.app.mediafilter.HTMLFilter.inputFormats = HTML,
Text
filter.org.dspace.app.mediafilter.WordFilter.inputFormats = Microsoft
Word
filter.org.dspace.app.mediafilter.JPEGFilter.inputFormats = GIF,
JPEG, image/png
filter.org.dspace.app.mediafilter.BrandedPreviewJPEGFilter.inputFormat
s = GIF, JPEG, image/png

##### Settings for Item Preview #####
webui.preview.enabled = false
# max dimensions of the preview image
webui.preview.maxwidth = 600
webui.preview.maxheight = 600
# the brand text
webui.preview.brand = My Institution Name
# an abbreviated form of the above text, this will be used
# when the preview image cannot fit the normal text
webui.preview.brand.abbrev = MyOrg
# the height of the brand
webui.preview.brand.height = 20
# font settings for the brand text
webui.preview.brand.font = SansSerif
webui.preview.brand.fontpoint = 12
#webui.preview.dc = rights

##### Checksum Checker Settings #####
# Default dispatcher in case none specified
plugin.single.org.dspace.checker.BitstreamDispatcher=org.dspace.checker.
SimpleDispatcher
# Standard interface implementations. You shouldn't need to tinker
# with these.
plugin.single.org.dspace.checker.ReporterDAO=org.dspace.checker.Report
erDAOImpl

# check history retention
checker.retention.default=10y
checker.retention.CHECKSUM_MATCH=8w
```

- If you have customised advanced search fields (*search.index.n* fields, note that you now need to include the schema in the values. Dublin Core is specified as *dc*. So for example, if in 1.3.2 you had:

```
search.index.1 = title:title.alternative
```

That needs to be changed to:

```
search.index.1 = title:dc.title.alternative
```

- If you use LDAP or X509 authentication, you'll need to add *org.dspace.eperson.LDAPAuthentication* or *org.dspace.eperson.X509Authentication* respectively. See also configuring custom authentication code.
- If you have custom Media Filters, note that these are now configured through *dspace.cfg* (instead of *mediafilter.cfg* which is obsolete.)
- Also, take a look through the default *dspace.cfg* file supplied with DSpace 1.4.x, as this contains configuration options for various new features you might like to use. In general, these new features default



to 'off' and you'll need to add configuration properties as described in the default 1.4.x *dspace.cfg* to activate them.

- Your 'localized' JSPs (those in *jsp/local*) now need to be maintained in the *source* directory. If you have locally modified JSPs in your *[dspace]/jsp/local* directory, you will need to merge the changes in the new 1.4.x versions into your locally modified ones. You can use the *diff* command to compare your JSPs against the 1.4.x versions to do this. You can also check against the <http://dspace.cvs.sourceforge.net/dspace/>.

- In *[dspace-1.4.x-source]* run:

```
ant -Dconfig= [dspace]/config/dspace.cfg update
```

- The database schema needs updating. SQL files containing the relevant file are provided. If you've modified the schema locally, you may need to check over this and make alterations.

- For PostgreSQL:** *[dspace-1.4.x-source]/etc/database_schema_13-14.sql* contains the SQL commands to achieve this for PostgreSQL. To apply the changes, go to the source directory, and run: *psql -f etc/database_schema_13-14.sql [DSpace database name] -h localhost*
- For Oracle:** *[dspace-1.4.x-source]/etc/oracle/database_schema_13-14.sql* should be run on the DSpace database to update the schema.

- Rebuild the search indices: *[dspace]/bin/index-all*

- Copy the *.war* Web application files in *[dspace-1.4-source]/build* to the *webapps* sub-directory of your servlet container (e.g. Tomcat). e.g.:

```
cp [dspace-1.4-source]/build/*.war  
[tomcat]/webapps
```

If you're using Tomcat, you need to delete the directories corresponding to the old *.war* files. For example, if *dspace.war* is installed in *[tomcat]/webapps/dspace.war*, you should delete the *[tomcat]/webapps/dspace* directory. Otherwise, Tomcat will continue to use the old code in that directory.

- Restart Tomcat.

4.7. Upgrading From 1.3.1 to 1.3.2

The changes in 1.3.2 are only code changes so the update is simply a matter of rebuilding the wars.

In the notes below *[dspace]* refers to the install directory for your existing DSpace installation, and *[dspace-1.3.2-source]* to the source directory for DSpace 1.3.2. Whenever you see these path references, be sure to replace them with the actual path names on your local system.

- Get the new DSpace 1.3.2 source code from <http://sourceforge.net/projects/dspace/> and unpack it somewhere. Do not unpack it on top of your existing installation!!
- Copy the PostgreSQL driver JAR to the source tree. For example:

```
cd [dspace]/lib  
cp postgresql.jar [dspace-1.3.2-source]/lib
```

- Take down Tomcat (or whichever servlet container you're using).
- Your 'localized' JSPs (those in *jsp/local*) now need to be maintained in the *source* directory. If you have locally modified JSPs in your *[dspace]/jsp/local* directory, you will need to merge the changes in the new 1.3.2 versions into your locally modified ones. You can use the *diff* command to compare the 1.3.1 and 1.3.2 versions to do this.



- In `[dSPACE-1.3.2-source]` run:

```
ant -Dconfig= [dSPACE]/config/dSPACE.cfg update
```

- Copy the `.war` Web application files in `[dSPACE-1.3.2-source]/build` to the `webapps` sub-directory of your servlet container (e.g. Tomcat). e.g.:

```
cp [dSPACE-1.3.2-source]/build/*.war
[ tomcat ]/webapps
```

If you're using Tomcat, you need to delete the directories corresponding to the old `.war` files. For example, if `dSPACE.war` is installed in `[tomcat]/webapps/dSPACE.war`, you should delete the `[tomcat]/webapps/dSPACE` directory. Otherwise, Tomcat will continue to use the old code in that directory.

- Restart Tomcat.

4.8. Upgrading From 1.2.x to 1.3.x

In the notes below `[dSPACE]` refers to the install directory for your existing DSpace installation, and `[dSPACE-1.3.x-source]` to the source directory for DSpace 1.3.x. Whenever you see these path references, be sure to replace them with the actual path names on your local system.

- Step one is, of course, to **back up all your data** before proceeding!! Include all of the contents of `[dSPACE]` and the PostgreSQL database in your backup.
- Get the new DSpace 1.3.x source code from <http://sourceforge.net/projects/dSPACE/> and unpack it somewhere. Do not unpack it on top of your existing installation!!
- Copy the PostgreSQL driver JAR to the source tree. For example: `cd [dSPACE]/libcp postgresql.jar [dSPACE-1.2.2-source]/lib`
- Take down Tomcat (or whichever servlet container you're using).
- Remove the old version of `xerces.jar` from your installation, so it is not inadvertently later used:
`[dSPACE]/lib/xerces.jar`
- Install the new config files by moving `dstat.cfg` and `dstat.map` from `[dSPACE-1.3.x-source]/config/` to `[dSPACE]/config`
- You need to add new parameters to your `[dSPACE]/dSPACE.cfg`:

```
##### Statistical Report Configuration Settings #####

# should the stats be publicly available?  should be set to false if
# you only
# want administrators to access the stats, or you do not intend to
# generate
# any
report.public = false

# directory where live reports are stored
report.dir = /dSPACE/reports/
```

- Build and install the updated DSpace 1.3.x code. Go to the `[dSPACE-1.3.x-source]` directory, and run:
`ant -Dconfig=[dSPACE]/config/dSPACE.cfg update`
- You'll need to make some changes to the database schema in your PostgreSQL database. `[dSPACE-1.3.x-source]/etc/database_schema_12-13.sql` contains the SQL commands to achieve this. If you've modified the schema locally, you may need to check over this and make alterations. To apply the changes, go to the source directory, and run: `psql -f etc/database_schema_12-13.sql [DSpace database name] -h localhost`



10. Customise the stat generating statistics as per the instructions in System Statistical Reports
11. Initialise the statistics using: `[dspace]/bin/stat-initial[dspace]/bin/stat-general[dspace]/bin/stat-report-initial[dspace]/bin/stat-report-general`
12. Rebuild the search indices: `[dspace]/bin/index-all`
13. Copy the `.war` Web application files in `[dspace-1.3.x-source]/build` to the `webapps` sub-directory of your servlet container (e.g. Tomcat). e.g.: `cp [dspace-1.3.x-source]/build/*.war [tomcat]/webapps`
14. Restart Tomcat.

4.9. Upgrading From 1.2.1 to 1.2.2

The changes in 1.2.2 are only code and config changes so the update should be fairly simple.

In the notes below `[dspace]` refers to the install directory for your existing DSpace installation, and `[dspace-1.2.2-source]` to the source directory for DSpace 1.2.2. Whenever you see these path references, be sure to replace them with the actual path names on your local system.

1. Get the new DSpace 1.2.2 source code from <http://sourceforge.net/projects/dspace/> and unpack it somewhere. Do not unpack it on top of your existing installation!!
2. Copy the PostgreSQL driver JAR to the source tree. For example:

```
cd [dspace]/lib
cp postgresql.jar [dspace-1.2.2-source]/lib
```

3. Take down Tomcat (or whichever servlet container you're using).
4. Your 'localized' JSPs (those in `jsp/local`) now need to be maintained in the `source` directory. If you have locally modified JSPs in your `[dspace]/jsp/local` directory, you might like to merge the changes in the new 1.2.2 versions into your locally modified ones. You can use the `diff` command to compare the 1.2.1 and 1.2.2 versions to do this. Also see the version history for a list of modified JSPs.
5. You need to add a new parameter to your `[dspace]/dspace.cfg` for configurable fulltext indexing

```
##### Fulltext Indexing settings #####
# Maximum number of terms indexed for a single field in Lucene.
# Default is 10,000 words - often not enough for full-text indexing.
# If you change this, you'll need to re-index for the change
# to take effect on previously added items.
# -1 = unlimited (Integer.MAX_VALUE)
search.maxfieldlength = 10000
```

6. In `[dspace-1.2.2-source]` run:

```
ant -Dconfig= [dspace]/config/dspace.cfg update
```

7. Copy the `.war` Web application files in `[dspace-1.2.2-source]/build` to the `webapps` sub-directory of your servlet container (e.g. Tomcat). e.g.:

```
cp [dspace-1.2.2-source]/build/*.war
[tomcat]/webapps
```

If you're using Tomcat, you need to delete the directories corresponding to the old `.war` files. For example, if `dspace.war` is installed in `[tomcat]/webapps/dspace.war`, you should delete the `[tomcat]/webapps/dspace` directory. Otherwise, Tomcat will continue to use the old code in that directory.

8. To finalise the install of the new configurable submission forms you need to copy the file `[dspace-1.2.2-source]/config/input-forms.xml` into `[dspace]/config`.



- Restart Tomcat.

4.10. Upgrading From 1.2 to 1.2.1

The changes in 1.2.1 are only code changes so the update should be fairly simple.

In the notes below *[dspace]* refers to the install directory for your existing DSpace installation, and *[dspace-1.2.1-source]* to the source directory for DSpace 1.2.1. Whenever you see these path references, be sure to replace them with the actual path names on your local system.

- Get the new DSpace 1.2.1 source code from <http://sourceforge.net/projects/dspace/> and unpack it somewhere. Do not unpack it on top of your existing installation!!
- Copy the PostgreSQL driver JAR to the source tree. For example:

```
cd [dspace]/lib
cp postgresql.jar [dspace-1.2.1-source]/lib
```

- Take down Tomcat (or whichever servlet container you're using).
- Your 'localized' JSPs (those in *jsp/local*) now need to be maintained in the *source* directory. If you have locally modified JSPs in your *[dspace]/jsp/local* directory, you might like to merge the changes in the new 1.2.1 versions into your locally modified ones. You can use the *diff* command to compare the 1.2 and 1.2.1 versions to do this. Also see the version history for a list of modified JSPs.
- You need to add a few new parameters to your *[dspace]/dspace.cfg* for browse/search and item thumbnails display, and for configurable DC metadata fields to be indexed.

```
# whether to display thumbnails on browse and search results pages
(1.2+)
webui.browse.thumbnail.show = false

# max dimensions of the browse/search thumbs. Must be <=
thumbnail.maxwidth
# and thumbnail.maxheight. Only need to be set if required to be
smaller than
# dimension of thumbnails generated by mediafilter (1.2+)
#webui.browse.thumbnail.maxheight = 80
#webui.browse.thumbnail.maxwidth = 80

# whether to display the thumb against each bitstream (1.2+)
webui.item.thumbnail.show = true

# where should clicking on a thumbnail from browse/search take the
user
# Only values currently supported are "item" and
"bitstream"
#webui.browse.thumbnail.linkbehaviour = item

##### Fields to Index for Search #####

# DC metadata elements.qualifiers to be indexed for search
# format: - search.index.[number] = [search field]:element.qualifier
#         - * used as wildcard

###      changing these will change your search results,      ###
###      but will NOT automatically change your search displays  ###

search.index.1 = author:contributor.*
search.index.2 = author:creator.*
search.index.3 = title:title.*
search.index.4 = keyword:subject.*
search.index.5 = abstract:description.abstract
search.index.6 = author:description.statementsofresponsibility
```



```
search.index.7 = series:relation.ispartofseries
search.index.8 = abstract:description.tableofcontents
search.index.9 = mime:format.mimetype
search.index.10 = sponsor:description.sponsorship
search.index.11 = id:identifier.*
```

6. In `[dspace-1.2.1-source]` run:

```
ant -Dconfig= [dspace]/config/dspace.cfg update
```

7. Copy the `.war` Web application files in `[dspace-1.2.1-source]/build` to the `webapps` sub-directory of your servlet container (e.g. Tomcat). e.g.:

```
cp [dspace-1.2.1-source]/build/*.war
   [tomcat]/webapps
```

If you're using Tomcat, you need to delete the directories corresponding to the old `.war` files. For example, if `dspace.war` is installed in `[tomcat]/webapps/dspace.war`, you should delete the `[tomcat]/webapps/dspace` directory. Otherwise, Tomcat will continue to use the old code in that directory.

8. Restart Tomcat.

4.11. Upgrading From 1.1 (or 1.1.1) to 1.2

The process for upgrading to 1.2 from either 1.1 or 1.1.1 is the same. If you are running DSpace 1.0 or 1.0.1, you need to follow the instructions for upgrading from 1.0.1 to 1.1 to before following these instructions.

Note also that if you've substantially modified DSpace, these instructions apply to an unmodified 1.1.1 DSpace instance, and you'll need to adapt the process to any modifications you've made.

This document refers to the install directory for your existing DSpace installation as `[dspace]`, and to the source directory for DSpace 1.2 as `[dspace-1.2-source]`. Whenever you see these path references below, be sure to replace them with the actual path names on your local system.

1. Step one is, of course, to **back up all your data** before proceeding!! Include all of the contents of `[dspace]` and the PostgreSQL database in your backup.
2. Get the new DSpace 1.2 source code from <http://sourceforge.net/projects/dspace/> and unpack it somewhere. Do not unpack it on top of your existing installation!!
3. Copy the required Java libraries that we couldn't include in the bundle to the source tree. For example:

```
cd [dspace]/lib
cp activation.jar servlet.jar mail.jar
   [dspace-1.2-source]/lib
```

4. Stop Tomcat (or other servlet container.)

5. It's a good idea to upgrade all of the various third-party tools that DSpace uses to their latest versions:

- Java (note that now version 1.4.0 or later is *required*)
- Tomcat (Any version after 4.0 will work; symbolic links are no longer an issue)
- PostgreSQL (don't forget to build/download an updated JDBC driver .jar file! Also, **back up the database** first.)
- Ant

6. You need to add the following new parameters to your `[dspace]/dspace.cfg`:



```
##### Media Filter settings #####
# maximum width and height of generated thumbnails
thumbnail.maxwidth 80
thumbnail.maxheight 80
```

There are one or two other, optional extra parameters (for controlling the pool of database connections). See the version history for details. If you leave them out, defaults will be used. Also, to avoid future confusion, you might like to **remove** the following property, which is no longer required:

```
config.template.oai-web.xml =
[dSPACE]/oai/WEB-INF/web.xml
```

- The layout of the installation directory (i.e. the structure of the contents of *[dSPACE]*) has changed somewhat since 1.1.1. First up, your 'localized' JSPs (those in *jsp/local*) now need to be maintained in the *source* directory. So make a copy of them now! Once you've done that, you can remove *[dSPACE]/jsp* and *[dSPACE]/oai*, these are no longer used. (.war Web application archive files are used instead). Also, if you're using the same version of Tomcat as before, you need to **remove the lines from Tomcat's conf/server.xml file that enable symbolic links for DSpace**. These are the `<Context>` elements you added to get DSpace 1.1.1 working, looking something like this:

```
<Context path="/dSPACE" docBase="dSPACE" debug="0" reloadable="true"
crossContext="true">
  <Resources className="org.apache.naming.resources.FileDirContext"
allowLinking="true" />
</Context>
```

Be sure to remove the `<Context>` elements for both the Web UI and the OAI Web applications.

- Build and install the updated DSpace 1.2 code. Go to the DSpace 1.2 source directory, and run:

```
ant -Dconfig= [dSPACE]/config/dSPACE.cfg update
```

- Copy the new config files in *config* to your installation, e.g.:

```
cp [dSPACE-1.2-source]/config/news-*
[dSPACE-1.2-source]/config/mediafilter.cfg
[dSPACE-1.2-source]/config/dc2mods.cfg
[dSPACE]/config
```

- You'll need to make some changes to the database schema in your PostgreSQL database. *[dSPACE-1.2-source]/etc/database_schema_11-12.sql* contains the SQL commands to achieve this. If you've modified the schema locally, you may need to check over this and make alterations. To apply the changes, go to the source directory, and run:

```
psql -f etc/database_schema_11-12.sql [DSpace database name] -h
localhost
```

- A tool supplied with the DSpace 1.2 codebase will then update the actual data in the relational database. Run it using:

```
[dSPACE]/bin/dsrun
org.dSPACE.administer.UpgradellTo12
```

- Then rebuild the search indices:

```
[dSPACE]/bin/index-all
```

- Delete the existing symlinks from your servlet container's (e.g. Tomcat's) *webapp* sub-directory. Copy the .war Web application files in *[dSPACE-1.2-source]/build* to the *webapps* sub-directory of your servlet container (e.g. Tomcat). e.g.:



```
cp [dspace-1.2-source]/build/*.war
[ tomcat]/webapps
```

14. Restart Tomcat.

15. To get image thumbnails generated and full-text extracted for indexing automatically, you need to set up a 'cron' job, for example one like this:

```
# Run the media filter at 02:00 every day
0 2 * * * [dspace]/bin/filter-media
```

You might also wish to run it now to generate thumbnails and index full text for the content already in your system.

16. **Note 1:** This update process has effectively 'touched' all of your items. Although the dates in the Dublin Core metadata won't have changed (accession date and so forth), the 'last modified' date in the database for each will have been changed. This means the e-mail subscription tool may be confused, thinking that all items in the archive have been deposited that day, and could thus send a rather long email to lots of subscribers. So, it is recommended that you **turn off the e-mail subscription feature for the next day**, by commenting out the relevant line in DSpace's cron job, and then re-activating it the next day. Say you performed the update on 08-June-2004 (UTC), and your e-mail subscription cron job runs at 4am (UTC). When the subscription tool runs at 4am on 09-June-2004, it will find that everything in the system has a modification date in 08-June-2004, and accordingly send out huge emails. So, immediately after the update, you would edit DSpace's 'crontab' and comment out the `/dspace/bin/subs-daily` line. Then, after 4am on 09-June-2004 you'd 'un-comment' it out, so that things proceed normally. Of course this means, any *real* new deposits on 08-June-2004 won't get e-mailed, however if you're updating the system it's likely to be down for some time so this shouldn't be a big problem.

17. **Note 2:** After consultation with the OAI community, various OAI-PMH changes have occurred:

- The OAI-PMH identifiers have changed (they're now of the form `oai:hostname:handle` as opposed to just Handles)
- The set structure has changed, due to the new sub-communities feature.
- The default base URL has changed
- As noted in note 1, every item has been 'touched' and will need re-harvesting. The above means that, if already registered and harvested, you will need to re-register your repository, effectively as a 'new' OAI-PMH data provider. You should also consider posting an announcement to the <http://www.openarchives.org/mailman/listinfo/OAI-implementers> so that harvesters know to update their systems. Also note that your site may, over the next few days, take quite a big hit from OAI-PMH harvesters. The resumption token support should alleviate this a little, but you might want to temporarily whack up the database connection pool parameters in `[dspace]/config/dspace.cfg`. See the `dspace.cfg` distributed with the source code to see what these parameters are and how to use them. (You need to stop and restart Tomcat after changing them.) I realize this is not ideal; for discussion as to the reasons behind this please see relevant posts to the OAI community: <http://openarchives.org/pipermail/oai-implementers/2004-June/001214.html>, <http://openarchives.org/pipermail/oai-implementers/2004-June/001224.html>, as well as #. If you really can't live with updating the base URL like this, you can fairly easily have things proceed more-or-less as they are, by doing the following:
- Change the value of `OAI_ID_PREFIX` at the top of the `org.dspace.app.oai.DSpaceOAI Catalog` class to `hdl`:
- Change the servlet mapping for the `OAIHandler` servlet back to `/` (from `/request`)
- Rebuild and deploy `_oai.war` However, note that in this case, all the records will be re-harvested by harvesters anyway, so you still need to brace for the associated DB activity; also note that the set spec



changes may not be picked up by some harvesters. It's recommended you read the above-linked mailing list posts to understand why the change was made. Now, you should be finished!

4.12. Upgrading From 1.1 to 1.1.1

Fortunately the changes in 1.1.1 are only code changes so the update is fairly simple.

In the notes below *[dspace]* refers to the install directory for your existing DSpace installation, and *[dspace-1.1.1-source]* to the source directory for DSpace 1.1.1. Whenever you see these path references, be sure to replace them with the actual path names on your local system.

1. Take down Tomcat.
2. It would be a good idea to update any of the third-party tools used by DSpace at this point (e.g. PostgreSQL), following the instructions provided with the relevant tools.
3. In *[dspace-1.1.1-source]* run:

```
ant -Dconfig= [dspace]/config/dspace.cfg update
```

4. If you have locally modified JSPs of the following JSPs in your *[dspace]/jsp/local* directory, you might like to merge the changes in the new 1.1.1 versions into your locally modified ones. You can use the *diff* command to compare the 1.1 and 1.1.1 versions to do this. The changes are quite minor.

```
collection-home.jsp
admin/authorize-collection-edit.jsp
admin/authorize-community-edit.jsp
admin/authorize-item-edit.jsp
admin/eperson-edit.jsp
```

5. Restart Tomcat.

4.13. Upgrading From 1.0.1 to 1.1

To upgrade from DSpace 1.0.1 to 1.1, follow the steps below. Your *dspace.cfg* does not need to be changed. In the notes below *[dspace]* refers to the install directory for your existing DSpace installation, and *[dspace-1.1-source]* to the source directory for DSpace 1.1. Whenever you see these path references, be sure to replace them with the actual path names on your local system.

1. Take down Tomcat (or whichever servlet container you're using).
2. We recommend that you upgrade to the latest version of PostgreSQL (7.3.2). Included are some postgres-upgrade-notes.txt. Note you will also have to upgrade Ant to version 1.5 if you do this.
3. Make the necessary changes to the DSpace database. These include a couple of minor schema changes, and some new indices which should improve performance. Also, the names of a couple of database views have been changed since the old names were so long they were causing problems. First run *psql* to access your database (e.g. *psql -U dspace -W* and then enter the password), and enter these SQL commands:

```
ALTER TABLE bitstream ADD store_number INTEGER;
UPDATE bitstream SET store_number = 0;

ALTER TABLE item ADD last_modified TIMESTAMP;
CREATE INDEX last_modified_idx ON Item(last_modified);

CREATE INDEX eperson_email_idx ON EPerson(email);
CREATE INDEX item2bundle_item_idx on Item2Bundle(item_id);
REATE INDEX bundle2bitstream_bundle_idx ON
Bundle2Bitstream(bundle_id);
```



```
CREATE INDEX dcvalue_item_idx on DCValue(item_id);
CREATE INDEX collection2item_collection_idx ON
Collection2Item(collection_id);
CREATE INDEX resourcepolicy_type_id_idx ON ResourcePolicy
(resource_type_id,resource_id);
CREATE INDEX epersongroup2eperson_group_idx on
EPersonGroup2EPerson(eperson_group_id);
CREATE INDEX handle_handle_idx ON Handle(handle);
CREATE INDEX sort_author_idx on ItemsByAuthor(sort_author);
CREATE INDEX sort_title_idx on ItemsByTitle(sort_title);
CREATE INDEX date_issued_idx on ItemsByDate(date_issued);

DROP VIEW CollectionItemsByDateAccessioned;

DROP VIEW CommunityItemsByDateAccessioned;
CREATE VIEW CommunityItemsByDateAccession as SELECT
Community2Item.community_id, ItemsByDateAccessioned.* FROM
ItemsByDateAccessioned, Community2Item WHERE
ItemsByDateAccessioned.item_id = Community2Item.item_id;
CREATE VIEW CollectionItemsByDateAccession AS SELECT
collection2item.collection_id,
itemsbydateaccessioned.items_by_date_accessioned_id,
itemsbydateaccessioned.item_id,
itemsbydateaccessioned.date_accessioned FROM itemsbydateaccessioned,
collection2item WHERE (itemsbydateaccessioned.item_id =
collection2item.item_id);
```

4. Fix your JSPs for Unicode. If you've modified the site 'skin' (*jsp/local/layout/header-default.jsp*) you'll need to add the Unicode header, i.e.:

```
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
```

to the <HEAD> element. If you have any locally-edited JSPs, you need to add this page directive to the top of all of them:

```
<%@ page contentType="text/html; charset=UTF-8" %>
```

(If you haven't modified any JSPs, you don't have to do anything.)

5. Copy the required Java libraries that we couldn't include in the bundle to the source tree. For example:

```
cd [dSPACE]/lib
cp *.policy.activation.jar servlet.jar mail.jar
[dSPACE-1.1-source]/lib
```

6. Compile up the new DSpace code, replacing *[dSPACE]/config/dSPACE.cfg* with the path to your current, LIVE configuration. (The second line, *touch `find .`*, is a precaution, which ensures that the new code has a current datestamp and will overwrite the old code. Note that those are back quotes.)

```
cd [dSPACE-1.1-source]
touch `find .`
ant
ant -Dconfig= [dSPACE]/config/dSPACE.cfg update
```

7. Update the database tables using the upgrader tool, which sets up the new *>last_modified* date in the item table:

```
Run [dSPACE]/bin/dsrun
org.dSPACE.administer.Upgrade101To11
```

8. Run the collection default authorisation policy tool:

```
[dSPACE]/bin/dsrun
org.dSPACE.authorize.FixDefaultPolicies
```



9. Fix the OAI Cat properties file. Edit `[dSPACE]/config/templates/oaicat.properties`. Change the line that says

```
Identify.deletedRecord=yes
```

To:

```
Identify.deletedRecord=persistent
```

This is needed to fix the OAI-PMH 'Identity' verb response. Then run `[dSPACE]/bin/install-configs`.

10. Re-run the indexing to index abstracts and fill out the renamed database views:

```
[dSPACE]/bin/index-all
```

11. Restart Tomcat. Tomcat should be run with the following environment variable set, to ensure that Unicode is handled properly. Also, the default JVM memory heap sizes are rather small. Adjust `-Xmx512M` (512Mb maximum heap size) and `-Xms64M` (64Mb Java thread stack size) to suit your hardware.

```
JAVA_OPTS="-Xmx512M -Xms64M -Dfile.encoding=UTF-8"
```

5. Configuration and Customization

There are a number of ways in which DSpace may be configured and/or customized. This chapter of the documentation will discuss the configuration of the software and will also reference customizations that may be performed in the chapter following.

For ease of use, the Configuration documentation is broken into several parts:

- Section 5.1 addresses general conventions used with configuring not only the `dSPACE.cfg` file, but other configuration files which use similar conventions.
- Section 5.2 specifies the basic `dSPACE.cfg` file settings
- Section 5.3 contains other more advanced settings that are optional in the `dSPACE.cfg` configuration file. **General Configuration**

In the following sections you will learn about the different configuration files that you will need to edit so that you may make your DSpace installation work. Of the several configuration files which you will work with, it is the `dSPACE.cfg` file you need to learn to configure first and foremost.

In general, most of the configuration files, namely `dSPACE.cfg` and `xmlui.xconf` will provide a good source of information not only with configuration but also with customization (cf. Customization chapter)

5.1. Input Conventions

We will use the `dSPACE.cfg` as our example for input conventions used throughout the system. It is a basic Java properties file, where lines are either comments, starting with a '#', blank lines, or property/value pairs of the form:

```
property.name = property value
```

Some property defaults are "commented out". That is, they have a '#' preceding them, and the DSpace software ignores the config property. This may cause the feature not to be enabled, or, cause a default property to be used when the software is compiled and updated.

The property value may contain *references* to other configuration properties, in the form `${property.name}`. This follows the *ant* convention of allowing references in property files. A property may not refer to itself. Examples:



```
property.name = word1 ${other.property.name} more words
property2.name = ${dspace.dir}/rest/of/path
```

Property values can include other, previously defined values, by enclosing the property name in `${...}`. For example, if your `dspace.cfg` contains:

```
dspace.dir = /dspace
dspace.history = ${dspace.dir}/history
```

Then the value of `dspace.history` property is expanded to be `/dspace/history`. This method is especially useful for handling commonly used file paths.

5.2. Update Reminder

Things you should know about editing `dspace.cfg` files.

It is important to remember that there are * two `dspace.cfg` files after an installation of DSpace.*

1. The "source" file that is found in `[dspace-source]/dspace/config/dspace.cfg`
2. The "runtime" file that is found in `[dspace]/config/dspace.cfg`
The runtime file is supposed to be the **copy** of the source file, which is considered the *master* version. However, the DSpace server and command programs only look at the *runtime* configuration file, so when you are revising your configuration values, it is tempting to *only edit the runtime file*. DO NOT do this. Always make the same changes to the source version of `dspace.cfg` in addition to the runtime file. The two files *should* always be identical, since the source `dspace.cfg` will be the basis of your next upgrade.

To keep the two files in synchronization, you can edit your files in `[dspace-source]/dspace/config/` and then you would run the following commands:

```
cd [dspace-source]/dspace/target/dspace-<version>-build.dir
ant update_configs
```

This will copy the source `dspace.cfg` (along with other configuration files) into the runtime (`[dspace]/config`) directory.

You should remember that after editing your configuration file(s), and you are done and wish to implement the changes, you will need to:

- To run `ant -Dconfig=[dspace]/config/dspace.cfg update` if you are updating your `dspace.cfg` file and wish to see the changes appear. Follow the usual sequence with copying your webapps.
- If you edit `dspace.cfg` in `[dspace-source]/dspace/config/`, you should then run '`ant init_configs`' in the directory `[dspace-source]/dspace/target/dspace-1.5.2-build.dir` so that any changes you may have made are reflected in the configuration files of other applications, for example Apache. You may then need to restart those applications, depending on what you changed.

5.3. The `dspace.cfg` Configuration Properties File

The primary way of configuring DSpace is to edit the `dspace.cfg`. You will definitely have to do this before you can run DSpace properly. `dspace.cfg` contains basic information about a DSpace installation, including system path information, network host information, and other like items. To assist you in this endeavor, below is a place for you to write down some of the preliminary data so that you may facilitate faster configuration. Server IP: _____ *Host Name (Server name)* _____ *dspace.url* _____ *Administrator's email:* _____ *handle prefix:* _____ *assetstore directory:* _____ *SMTP server:* _____



5.3.1. The dspace.cfg file

Below is a brief "Properties" table for the *dspace.cfg* file and the documented details are referenced. Please refer to those sections for the complete details of the parameter

Property	Ref. Sect.
General Configurations	
dspace.dir dspace.url dspace.baseUrl dspace.oai.url dspace.hostname dspace.name	5.2.2
Database Configurations	
db.name db.url db.driver db.username db.password	3.2.3 or 5.2.3
Advanced Database Configuration	
db.schema db.maxconnection db.maxwait db.maxidle db.statementpool db.poolname	5.2.3
Email Settings	
mail.server mail.server.username mail.server.password mail.server.port mail.from.address feedback.recipient mail.admin alert.recipient registration.notify mail.charset mail.allowed.referrers mail.extraproperties mail.server.disabled	5.2.4
File Storage	
assetstore.dir [assetstore.dir.1 assetstore.dir.2 assetstore.incoming]	5.2.5
SRB File Storage	
srb.hosts.1 srb.port.1 srb.mcatzone.1 srb.mdasdomainname.1 srb.defaultstorageresource.1 srb.username.1 srb.password.1 srb.homedirectory.1 srb.parentdir.1	5.2.6
Handle Configuration	
handle.prefix handle.dir	5.2.7



Property	Ref. Sect.
Authorization System Configuration	
core.authorization.community-admin.create-subelement core.authorization.community-admin.delete-subelement core.authorization.community-admin.policies core.authorization.community-admin.admin-group core.authorization.community-admin.collection.policies core.authorization.community-admin.collection.template-item core.authorization.community-admin.collection.submitters core.authorization.community-admin.collection.workflows core.authorization.community-admin.collection.admin-group core.authorization.community-admin.item.delete core.authorization.community-admin.item.withdraw core.authorization.community-admin.item.reinstatiate core.authorization.community-admin.item.policies core.authorization.community-admin.item.create-bitstream core.authorization.community-admin.item.delete-bitstream core.authorization.community-admin.item-admin.cc-license core.authorization.collection-admin.policies core.authorization.collection-admin.template-item core.authorization.collection-admin.submitters core.authorization.collection-admin.workflows core.authorization.collection-admin.admin-group core.authorization.collection-admin.item.delete core.authorization.collection-admin.item.withdraw core.authorization.collection-admin.item.reinstatiate core.authorization.collection-admin.item.policies core.authorization.collection-admin.item.create-bitstream core.authorization.collection-admin.item.delete-bitstream core.authorization.collection-admin.item-admin.cc-license core.authorization.item-admin.policies core.authorization.item-admin.create-bitstream core.authorization.item-admin.delete-bitstream core.authorization.item-admin.cc-license	5.2.45
Stackable Authentication Methods	
plugin.sequence.org.dspace.authenticate.AuthenticationMethod	5.2.8 or 5.2.9
LDAP Authentication	



Property	Ref. Sect.
ldap.enable ldap.provider_url ldap.id_field ldap.object_context ldap.search_context ldap.email_field ldap.surname_field ldap.givenname_field ldap.phone_field webui.ldap.autoregister ldap.login.specialgroup	5.2.8.5
<i>Hierarchical LDAP Settings:</i>	5.2.8.5
ldap.search_scope ldap.search.user ldap.netid_email_domain	
Shibboleth Authentication Settings	
authentication.shib.email-header authentication.shib.firstname-header authentication.shib.lastname-header authentication.shib.email-use-tomcat-remote-user authentication.shib.autoregister authentication.shib.role-header authentication.shib.default-roles	5.2.9-
Log Configuration	
<i>log.init.config</i>	5.2.10
Lucene Search Indexes	
search.dir search.max-clauses search.analyzer search.operator search.maxfieldlengthsearch.index.n	5.2.11
Proxy Settings	
http.proxy.host http.proxy.port	5.2.12
Media Filter	
<i>filter.plugin</i>	5.2.13
plugin.named.org.dspace.app.mediafilter.FormatFilter filter.org.dspace.app.mediafilter.PDFFilter.inputFormats filter.org.dspace.app.mediafilter.HTMLFilter.inputFormats filter.org.dspace.app.mediafilter.WordFilter.inputFormats filter.org.dspace.app.mediafilter.JPEGFilter.inputFormats filter.org.dspace.app.mediafilter.BrandedPreviewJPEGFilter.inputFormats	5.2.13
pdffilter.largepdfs pdffilter.skiponmemoryexception	5.2.13
Crosswalks (MODS, QDC, XSLT, etc.)	
crosswalk.mods.properties.MODS crosswalk.submission.MODS.stylesheet crosswalk.qdc.namespace.QDC.dc crosswalk.qdc.namespace.QDC.dcterms	5.2.18



Property	Ref. Sect.
crosswalk.qdc.schemaLocation.QDC plugin.named.org.dspace.content.crosswalk.IngestionCrosswalk plugin.named.org.dspace.content.crosswalk.DisseminationCrosswalk	
Event Settings	
event.dispatcher.default.class event.dispatcher.default.consumers event.dispatcher.noindex.class event.dispatcher.noindex.consumers event.consumer.search.class event.consumer.search.filters event.consumer.browse.class event.consumer.browse.filters event.consumer.eperson.class event.consumer.eperson.filters event.consumer.test.class event.consumer.test.filters testConsumer.verbose	5.2.19
Checksum Checker	
plugin.single.org.dspace.checker.BitstreamDispatcher checker.retention.default checker.retention.CHECKSUM-MATCH	5.2.20
Item Export and Download	
org.dspace.app.itemexport.work.dir org.dspace.app.itemexport.download.dir org.dspace.app.itemexport.life.span.hours org.dspace.app.itemexport.max.size	5.2.21
Bulk Metadata Editing	
bulkedit.valueseparator bulkedit.fieldseparator bulkedit.gui-item-limit bulkedit.ignore-on-export	5.2.46
Subscription Email Option	
<i>eperson.subscription.onlynew</i>	5.2.22
Submission Process	
<i>webui.submit.blocktheses</i>	5.2.23
<i>webui.submit.upload.required</i>	5.2.23
<i>webui.submit.enable-cc</i>	5.2.24
WEBUI Configurations [General]	
webui.browse.thumbnail.max.height webui.browse.thumbnail.max.width	5.2.25
<i>webui.browse.thumbnail.linkbehaviour</i>	5.2.25
thumbnail.maxwidth thumbnail.maxheight	5.2.25
<i>webui.preview.enabled</i>	5.2.25
webui.preview.maxwidth webui.preview.maxheight	5.2.25
webui.preview.brand webui.preview.brand.abbrev	5.2.25
<i>webui.preview.brand.height</i>	5.2.25
webui.preview.brand.font	5.2.25



Property	Ref. Sect.
<code>webui.preview.brnk.fontpoint</code>	
<code>webui.preview.dc</code>	5.2.25
<code>webui.strengths.show</code> <code>webui.strengths.cache</code>	5.2.25
Browse Index Configuration	
<code>webui.browse.index.n</code>	5.2.26
<code>webui.itemlist.sort-option.n</code>	5.2.26
<code>webui.browse.metadata.case-insensitive</code>	5.2.26.3
<code>webui.browse.value_columns.max</code> <code>webui.browse.sort_columns.max</code> <code>webui.browse.value_columns.omission_mark</code> <code>plugin.named.org.dspace.sort.OrderFormatDelegate</code>	5.2.26.4
Multiple Metadata Value Display	
<code>webui.browse.author-field</code>	5.2.27
<code>webui.browse.author-limit</code>	5.2.27
Other Browse Contexts	
<code>webui.browse.link.n</code>	5.2.28
Recent Submission	
<code>recent.submission.sort-option</code> <code>recent.submissions.count</code>	5.2.29
Syndication Feed (RSS) Settings	
<code>webui.feed.enable</code> <code>webui.feed.items</code> <code>webui.feed.cache.size</code> <code>webui.cache.age</code> <code>webui.feed.formats</code> <code>webui.feed.localresolve</code> <code>webui.feed.item.title</code> <code>webui.feed.item.date</code> <code>webui.feed.item.description</code>	5.2.30
Content Inline Disposition Threshold	
<code>webui.content_disposition_threshold</code> <code>xmlui.content_disposition_threshold</code>	5.2.31
Multifile HTML Settings	
<code>webui.html.max-depth-guess</code> <code>xmlui.html.max-depth-guess</code>	5.2.32
Other General Configuration Settings	
<code>sitemap.dir</code>	5.2.33
<code>sitemap.engineurls</code>	5.2.33
<code>upload.temp.dir</code>	5.2.34
<code>default.locale</code>	5.2.37
<code>itemmap.author.index</code>	5.2.38
<code>webui.mydspace.showgroupmembership</code>	5.2.39
<code>sfx.server.url</code>	5.2.40
<code>webui.suggest.enable</code> <code>webui.suggest.loggedinusers.only</code>	5.2.41



Property	Ref. Sect.
<code>webui.controlledvocabulary.enable</code>	5.2.42
<code>oai.didl.maxresponse</code>	5.2.44
JSP Web Interface Settings	
<pre>webui.licence_bundle.show webui.itemdisplay.default webui.resolver.1.urn webui.resolver.1.baseurl webui.resolver.2.urn webui.resolver.2.baseurl plugin.single.org.dspace.app.webui.util.StyleSelection webui.itemdisplay.thesis.collections webui.itemdisplay.metadata-style webui.itemlist.column webui.itemlist.width webui.itemlist.browse.<index name>.sort.<sort name>.columns webui.itemlist.sort<sort name>.columns webui.itemlist.browse.<browse name>.columns webui.itemlist.*lt;sort or index name>.columns webui.itemlist.dateaccessioned.columns webui.itemlist.dateaccessioned.widths webui.itemlist.tablewidth</pre>	5.2.36
XMLUI Settings (Manakin)	
<pre>xmlui.supported.locales xmlui.force.ssl xmlui.user.registration xmlui.user.assumelogon xmlui.user.logindirect xmlui.theme.allowoverrides xmlui.bundle.upload xmlui.community-list.render.full xmlui.community-list.cache xmlui.bitstream.mods xmlui.bitstream.mets xmlui.google.analytics.key xmlui.controlpanel.activity.max xmlui.controlpanel.activity.ipheader</pre>	5.2.43

5.3.2. Main DSpace Configurations

Property:	<code>dspace.dir</code>
Example Value:	<code>/dspace</code>
Informational Note:	Root directory of DSpace installation. Omit the trailing '/'. Note that if you change this, there are several other parameters you will probably want to change to match, e.g. <code>assetstore.dir</code> .
Property:	<code>dspace.baseUrl</code>
Example Value:	<code>_ http://dspace-test.myu.edu:8080</code>
Informational Note:	Main URL at which DSpace Web UI webapp is deployed. Include any port number, but do not include the trailing '/'.
Property:	<code>dspace.url</code>
Example Value:	<code>dspace.url = \${dspace.baseUrl}/jspui</code>



Informational note	DSpace base URL. Include port number etc., but NOT trailing slash. Change to <i>/xmlui</i> if you wish to use the xmlui (Manakin) as the default, or remove <i>"/jspui"</i> and set webapp of your choice as the "ROOT" webapp in the servlet engine.
Property:	<i>dspace.oai.url</i>
Example Value:	<i>dspace.oai.url = \${dspace.baseUrl}/oai</i>
Informational note:	The base URL of the OAI webapp (do not include / request).
Property:	<i>_dspace.hostname _</i>
Example Value:	<i>dspace.hostname = dspace.mysu.edu</i>
Informational Note:	Fully qualified hostname; do not include port number.
Property:	<i>dspace.name</i>
Example Value:	<i>dspace.name = DSpace at My University</i>
Informational Note:	Short and sweet site name, used throughout Web UI, e-mails and elsewhere (such as OAI protocol)

5.3.3. DSpace Database Configuration

Many of the database configurations are software-dependent. That is, it will be based on the choice of database software being used. Documentation is below shows PostgreSQL and Oracle examples.

Property:	<i>db.name</i>
Example Value:	<i>db.name = postgres</i>
Informational Note:	In <i>dspace.cfg</i> you choose either <i>postgres</i> or <i>oracle</i> .
Property:	<i>_db.url _</i>
Example Value:	<i>db.url = jdbc:postgresql://localhost:5432/dspace-services</i>
Informational Note:	The above value is the default value when configuring with PostgreSQL. When using Oracle, use this value: <i>jdbc.oracle.thin:@//host:port/dspace</i>
Property:	<i>db.username</i>
Example Value:	<i>db.username = dspace</i>
Informational Note:	In the installation directions, the administrator is instructed to create the user "dspace" who will own the database "dspace".
Property:	<i>password</i>
Example Value:	<i>password = dspace5</i>
Informational Note:	This is the password that was prompted during the installation process (cf. 3.2.3. Installation)



Property:	<i>db.schema</i>
Example Value:	<i>db.schema = vra</i>
Informational Note:	If your database contains multiple schemas, you can avoid problems with retrieving the definitions of duplicate objects by specifying the schema name here that is used for DSpace by uncommenting the entry. This is commented out
Property:	<i>db.maxconnections</i>
Example Value:	<i>db.maxconnections = 30</i>
Informational Note:	Maximum number of DB connections in pool
Property:	<i>db.maxwait</i>
Example Value:	<i>db.maxwait = 5000</i>
Informational Note:	Maximum time to wait before giving up if all connections in pool are busy (in milliseconds).
Property:	<i>db.maxidle</i>
Example Value:	<i>db.maxidle = -1</i>
Informational Note:	Maximum number of idle connections in pool. (-1 = unlimited)
Property:	<i>db.statementpool</i>
Example Value:	<i>db.statementpool = true</i>
Informational Note:	Determines if prepared statement should be cached. (Default is set to true)
Property:	<i>db.poolname</i>
Example Value:	<i>db.poolname = dspacepool</i>
Informational Note:	Specify a name for the connection pool. This is useful if you have multiple applications sharing Tomcat's database connection pool. If nothing is specified, it will default to 'dspacepool'

5.3.4. DSpace Email Settings

The configuration of email is simple and provides a mechanism to alert the person(s) responsible for different features of the DSpace software.

Property:	<i>mail.server</i>
Example Value:	<i>mail.server = smtp.my.edu</i>
Informational Note:	SMTP Mail Server
Property:	<i>mail.server.username</i> <i>mail.server.password</i>



Example Value:	<code>mail.server.username = myusername mypassword</code>
Informational Note:	SMTP mail server authentication username and password, if required.
Property:	<i>mail.server.port</i>
Example Value:	<i>mail.server.port = 25</i>
Informational Note:	SMTP mail server alternate port (Defaults to port 25 when commented out).
Property:	<i>mail.from.address</i>
Example Value:	<i>mail.from.address = dspace-noreply@myu.edu</i>
Informational Note:	The "From" address for email. Change the 'myu.edu' to the site's host name.
Property:	<i>feedback.recipient</i>
Example Value:	<i>feedback.recipient = dspace-help@myu.edu</i>
Informational Note:	When a user clicks on the feedback link/feature, the information will be send to the email address of choice. This configuration is currently limited to only one recipient.
Property:	<i>mail.admin</i>
Example Value:	<i>mail.admin = dspace-help@myu.edu</i>
Informational Note:	Email address of the general site administrator (Webmaster)
Property:	<i>alert.recipient</i>
Example Value:	<i>alert.recipient = john.doe@myu.edu</i>
Informational Note:	Enter the recipient for server errors and alerts.
Property:	<i>registration.notify</i>
Example Value:	<i>registration.notify = mike.smith@myu.edu</i>
Informational Note:	Enter the recipient that will be notified when a new user registers on DSpace.
Property:	<i>mail.charset</i>
Example Value:	<i>mail.charset = UTF8</i>
Informational Note:	Set the default mail character set. This may be overridden by providing a line inside the email template " <i>charset: <encoding></i> ", otherwise this default is used.
Property:	<i>mail.allowed.referrers</i>
Example Value:	<i>mail.allowed.referrers = localhost</i>



Informational Note:	A comma separated list of hostnames that are allowed to refer browsers to email forms. Default behaviour is to accept referrals only from dspace.hostname
Property:	<i>mail.extraproperties</i>
Example Value:	<pre>mail.extraproperties = mail.smtp.socketFactory.port=465, \ mail.smtp.socketFactory.class=javax.net.ssl.SSLSocketFactory \ mail.smtp.socketFactory.fallback=false</pre>
Informational Note:	If you need to pass extra settings to the Java mail library. Comma separated, equals sign between the key and the value. By default, commented out.
Property:	<i>mail.server.disabled</i>
Example Value:	<i>mail.server.disabled = false</i>
Informational Note:	An option is added to disable the mailserver. By default, this property is set to false. By setting value to 'true', DSpace will not send out emails. It will instead log the subject of the email which should have been sent. This is especially useful for development and test environments where production data is used when testing functionality.

Wording of E-mail Messages

Sometimes DSpace automatically sends e-mail messages to users, for example to inform them of a new work flow task, or as a subscription e-mail alert. The wording of emails can be changed by editing the relevant file in *[dspace]/config/emails*. Each file is commented. Be careful to keep the right number 'placeholders' (e.g. {2}).

Note: You should replace the contact-information "*dspace-help@myu.edu or call us at xxx-555-xxxx*" with your own contact details in:
config/emails/change_password
config/emails/register

5.3.5. File Storage

This is the default "technique" that is used by DSpace to store the bitstreams. DSpace can also use SRB (Storage Resource Brokerage) as an alternative. See section 5.2.6 for details regarding SRB.

Property:	<i>assetstore.dir</i>
Example Value:	<i>assetstore.dir = \${dspace.dir}/assetstore</i>
Informational Note:	This is Asset (bitstream) store number 0 (Zero). You need not place your assetstore under the <i>/dspace</i> directory, but may want to place it on a different logical volume on the server that DSpace resides. So, you might have something like this: <i>_ assetstore.dir = /storevgm/assestore_.</i>
Property:	<i>assetstore.dir.1</i>



	<code>assetstore.dir.2</code>
Example Value:	<code>assetstore.dir.1 = /second/assetstore</code> <code>assetstore.dir.2 = /third/assetstore</code>
Informational Note:	This property specifies extra asset stores like the one above, counting from one (1) upwards. This property is commented out (#) until it is needed.
Property:	<code>assetstore.incoming</code>
Example Value:	<code>assetstore.incoming = 1</code>
Informational Note:	Specify the number of the store to use for new bitstreams with this property. The default is 0 [zero] which corresponds to the 'assestore.dir' above.

In the examples above, you can see that your storage does not have to be under the `/dspace` directory. For the default installation it needs to reside on the same server (unless you plan to configure SRB (cf. below)). So, if you added storage space to your server, and it has a different logical volume/name/directory, you could have the following as an example:

```
assetstore.dir = /storevgm/assetstore
assetstore.dir.1 = /storevgm2/assetstore
assetstore.incoming = 1
```

Please Note: When adding additional storage configuration, you will then need to uncomment and declare `assestore.incoming = 1`

5.3.6. SRB (Storage Resource Brokerage) File Storage

An alternate to using the default storage framework is to use Storage Resource Brokerage (SRB). This can provide a different level of storage and disaster recovery. (Storage can take place on storage that is off-site.) Refer to http://www.sdsc.edu/srb/index.php/Main_Page for complete details regarding SRB.

The same framework is used to configure SRB storage. That is, the asset store number (0..n) can reference a file system directory as above or it can reference a set of SRB account parameters. But any particular asset store number can reference one or the other but not both. This way traditional and SRB storage can both be used but with different asset store numbers. The same cautions mentioned above apply to SRB asset stores as well. The particular asset store a bitstream is stored in is held in the database, so don't move bitstreams between asset stores, and do not renumber them.

Property:	<code>srb.hosts.1</code>
Example value:	<code>srb.hosts.1 = mysrbmcahost.myu.edu</code>
Property:	<code>srb.port.1</code>
Example value:	<code>srb.port.1 = 5544</code>
Property:	<code>srb.mcatzone.1</code>
Example value:	<code>srb.mcatzone.1 = mysrbzone</code>
Property:	<code>srb.mdasdomainname.1</code>
Example value:	<code>srb.mdasdomainname.1 = mysrbdomain</code>
Property:	<code>srb.defaultstorageresource.1</code>
Example value:	<code>srb.defaultstorageresource.1 = mydefaultsrbrsource</code>
Property:	<code>srb.username.1</code>
Example value:	<code>srb.username.1 = mysrbuser</code>
Property:	<code>srb.password.1</code>
Example value:	<code>srb.password.1 = mysrbpassword</code>
Property:	<code>srb.homedirectory.1</code>



Example value:	<code>srb.homedirectory.1 = /mysrbzone/home/ mysrbuser.mysrbdomain</code>
Property:	<code>srb.parentdir.1</code>
Example value:	<code>srb.parentdir.1 = mysrbdspaceassetstore</code>
Informational Note:	Several of the terms, such as <i>mcatzone</i> , have meaning only in the SRB context and will be familiar to SRB users. The last, <i>srb.parentdir.n</i> , can be used for additional (SRB) upper directory structure within an SRB account. This property value could be blank as well.

The 'assetstore.incoming' property is an integer that references where **new** bitstreams will be stored. The default (say the starting reference) is zero. The value will be used to identify the storage where all new bitstreams will be stored until this number is changed. This number is stored in the Bitstream table (store_number column) in the DSpace database, so older bitstreams that may have been stored when 'asset.incoming' had a different value can be found.

In the simple case in which DSpace uses local (or mounted) storage the number can refer to different directories (or partitions). This gives DSpace some level of scalability. The number links to another set of properties 'assetstore.dir', 'assetstore.dir.1' (remember zero is default), 'assetstore.dir.2', etc., where the values are directories.

To support the use of SRB DSpace uses the same scheme but broaden to support:

- using SRB instead of the local file system
- using the local file system (native DSpace)
- using a mix of SRB and local file system
in this broadened use of the 'asset.incoming' integer will refer to one of the following storage locations:
- a local file system directory (native DSpace)
- a set of SRB account parameters (host, port, zone, domain, username, password, home directory, and resource
Should there be any conflict, like '2' referring to a local directory and to a set of SRB parameters, the program will select the local directory.

If SRB is chosen from the first install of DSpace, it is suggested that 'assetstore.dir' (no integer appended) be retained to reference a local directory (as above under File Storage) because build.xml uses this value to do a mkdir. In this case, 'assetstore.incoming' can be set to 1 (i.e. uncomment the line in File Storage above) and the 'assetstore.dir' will not be used.

5.3.7. Handle Server Configuration

Property:	<code>handle.prefix</code>
Example Value	<code>handle.prefix = 1234.56789</code>
Informational Note:	The default installed by DSpace is <i>123456789</i> but you will replace this upon receiving a handle from CNRI.
Property:	<code>handle.dir</code>
Example Value:	<code>handle.dir = \${dspace.dir}/handle-server</code>
Informational Note:	The default files, as shown in the Example Value is where DSpace will install the files used for the Handle Server.



For complete information regarding the Handle server, the user should consult 3.3.4. The Handle Server section of Installing DSpace.

5.3.8. Stackable Authentication Method(s)

(formally Custom Authentication)

Since many institutions and organizations have existing authentication systems, DSpace has been designed to allow these to be easily integrated into an existing authentication infrastructure. It keeps a series, or "stack", of *authentication methods*, so each one can be tried in turn. This makes it easy to add new authentication methods or rearrange the order without changing any existing code. You can also share authentication code with other sites.

Property:	<code>plugin.sequence.org.dspace.authenticate.AuthenticationMethod</code>
Example Value:	<code>plugin.sequence.org.dspace.authenticate.AuthenticationMethod = \ org.dspace.authenticate.PasswordAuthentication</code>

The configuration property `plugin.sequence.org.dspace.authenticate.AuthenticationMethod` defines the authentication stack. It is a comma-separated list of class names. Each of these classes implements a different *authentication method*, or way of determining the identity of the user. They are invoked in the order specified until one succeeds.

An authentication method is a class that implements the interface `org.dspace.authenticate.AuthenticationMethod`. It *authenticates* a user by evaluating the *credentials* (e.g. username and password) he or she presents and checking that they are valid.

The basic authentication procedure in the DSpace Web UI is this:

1. A request is received from an end-user's browser that, if fulfilled, would lead to an action requiring authorization taking place.
2. If the end-user is already authenticated:
 - If the end-user is allowed to perform the action, the action proceeds
 - If the end-user is NOT allowed to perform the action, an authorization error is displayed.
 - If the end-user is NOT authenticated, i.e. is accessing DSpace anonymously:
3. The parameters etc. of the request are stored
4. The Web UI's `startAuthentication` method is invoked.
5. First it tries all the authentication methods which do *implicit* authentication (i.e. they work with just the information already in the Web request, such as an X.509 client certificate). If one of these succeeds, it proceeds from Step 2 above.
6. If none of the implicit methods succeed, the UI responds by putting up a "login" page to collect credentials for one of the *explicit* authentication methods in the stack. The servlet processing that page then gives the proffered credentials to each authentication method in turn until one succeeds, at which point it retries the original operation from Step 2 above.
Please see the source files `AuthenticationManager.java` and `AuthenticationMethod.java` for more details about this mechanism.

Authentication by Password

The default method `org.dspace.authenticate.PasswordAuthentication` has the following properties:

- Use of inbuilt e-mail address/password-based log-in. This is achieved by forwarding a request that is attempting an action requiring authorization to the password log-in servlet, `/password-login`. The password



log-in servlet (*org.dspace.app.webui.servlet.PasswordServlet* contains code that will resume the original request if authentication is successful, as per step 3. described above.

- Users can register themselves (i.e. add themselves as e-people without needing approval from the administrators), and can set their own passwords when they do this
- Users are not members of any special (dynamic) e-person groups
- You can restrict the domains from which new users are able to register. To enable this feature, uncomment the following line from dspace.cfg: *authentication.password.domain.valid = example.com* Example options might be '@example.com' to restrict registration to users with addresses ending in @example.com, or '@example.com, .ac.uk' to restrict registration to users with addresses ending in @example.com or with addresses in the .ac.uk domain.

X.509 Certificate Authentication

The X.509 authentication method uses an X.509 certificate sent by the client to establish his/her identity. It requires the client to have a personal Web certificate installed on their browser (or other client software) which is issued by a Certifying Authority (CA) recognized by the web server.

1. See the HTTPS installation instructions to configure your Web server. If you are using HTTPS with Tomcat, note that the *<Connector>* tag *must* include the attribute *clientAuth="true"* so the server requests a personal Web certificate from the client.
2. Add the *org.dspace.authenticate.X509Authentication* plugin *first* to the list of stackable authentication methods in the value of the configuration key *plugin.sequence.org.dspace.authenticate.AuthenticationMethod_e.g.:*

```
plugin.sequence.org.dspace.authenticate.AuthenticationMethod = \
    org.dspace.authenticate.X509Authentication, \
    org.dspace.authenticate.PasswordAuthentication
```

3. You must also configure DSpace with the same CA certificates as the web server, so it can accept and interpret the clients' certificates. It can share the same keystore file as the web server, or a separate one, or a CA certificate in a file by itself. Configure it by *one* of these methods, either the Java keystore

```
authentication.x509.keystore.path = path to Java keystore file
authentication.x509.keystore.password = password to access the keystore
```

...or the separate CA certificate file (in PEM or DER format):

```
authentication.x509.ca.cert = path to certificate file for CA
                             whose client certs to accept.
```

4. Choose whether to enable auto-registration: If you want users who authenticate successfully to be automatically registered as new E-Persons if they are not already, set the *authentication.x509.autoregister* configuration property to *true*. This lets you automatically accept all users with valid personal certificates. The default is *false*.

Example of a Custom Authentication Method

Also included in the source is an implementation of an authentication method used at MIT, *edu.mit.dspace.MITSpecialGroup*. This does not actually authenticate a user, it *only* adds the current user to a special (dynamic) group called 'MIT Users' (which must be present in the system!). This allows us to create authorization policies for MIT users without having to manually maintain membership of the MIT users group.

By keeping this code in a separate method, we can customize the authentication process for MIT by simply adding it to the stack in the DSpace configuration. None of the code has to be touched.



You can create your own custom authentication method and add it to the stack. Use the most similar existing method as a model, e.g. *org.dspace.authenticate.PasswordAuthentication* for an "explicit" method (with credentials entered interactively) or *org.dspace.authenticate.X509Authentication* for an implicit method.

Configuring IP Authentication

You can enable IP authentication by adding its method to the stack in the DSpace configuration, e.g.:

```
plugin.sequence.org.dspace.authenticate.AuthenticationMethod =
    org.dspace.authenticate.IPAAuthentication
```

You are then able to map DSpace groups to IP's in dspace.cfg by setting *authentication.ip.GROUPNAME = iprange[, iprange ...]*, e.g:

```
authentication.ip.MY_UNIVERSITY = 10.1.2.3, \           # Full IP
                                   13.5, \             # Partial IP
                                   11.3.4.5/24, \      # with CIDR
                                   12.7.8.9/255.255.128.0 # with netmask
```

Negative matches can be set by prepending the entry with a '-'. For example if you want to include all of a class B network except for users of a contained class c network, you could use: 111.222,-111.222.333.

Note: if the Groupname contains blanks you must escape the, e.g. Department\ of \ Statistics

Configuring LDAP Authentication

You can enable LDAP authentication by adding its method to the stack in the DSpace configuration, e.g.

```
plugin.sequence.org.dspace.authenticate.AuthenticationMethod =
    org.dspace.authenticate.LDAPAuthentication
```

If LDAP is enabled in the dspace.cfg file, then new users will be able to register by entering their username and password without being sent the registration token. If users do not have a username and password, then they can still register and login with just their email address the same way they do now.

If you want to give any special privileges to LDAP users, create a stackable authentication method to automatically put people who have a netid into a special group. You might also want to give certain email addresses special privileges. Refer to the Custom Authentication Code section above for more information about how to do this.

Here is an explanation of what each of the different configuration parameters are for:

Standard LDAP Configuration	
Property:	<i>ldap.enable</i>
Example Value:	<i>ldap.enable = false</i>
Informational Note:	This setting will enable or disable LDAP authentication in DSpace. With the setting off, users will be required to register and login with their email address. With this setting on, users will be able to login and register with their LDAP user ids and passwords.
Property:	<i>ldap.provider_url</i>
Example Value:	<i>ldap.provider_url = ldap://ldap.myu.edu/o=myu.edu</i>
Informational Note:	This is the url to your institution's LDAP server. You may or may not need the /o=myu.edu part at the end. Your server may also require the ldaps:// protocol.
Property:	<i>ldap.id_field</i>



Example Value:	<i>ldap.id_field = uid</i>
Explanation:	This is the unique identifier field in the LDAP directory where the username is stored.
Property:	<i>ldap.object_context</i>
Example Value:	<i>ldap.object_context = ou=people, o=myu.edu</i>
Informational Note:	This is the object context used when authenticating the user. It is appended to the <i>ldap.id_field</i> and username. For example <i>uid=username,ou=people,o=myu.edu</i> . You will need to modify this to match your LDAP configuration.
Property:	<i>ldap.search_context</i>
Example Value:	<i>ldap.search_context = ou=people</i>
Informational Note:	This is the search context used when looking up a user's LDAP object to retrieve their data for autoregistering. With <i>ldap.autoregister</i> turned on, when a user authenticates without an EPerson object we search the LDAP directory to get their name and email address so that we can create one for them. So after we have authenticated against <i>uid=username,ou=people,o=byu.edu</i> we now search in <i>ou=people</i> for filtering on <i>[uid=username]</i> . Often the <i>ldap.search_context</i> is the same as the <i>ldap.object_context</i> parameter. But again this depends on your LDAP server configuration.
Property:	<i>ldap.email_field</i>
Example Value:	<i>ldap.email_field = mail</i>
Informational Note:	This is the LDAP object field where the user's email address is stored. "mail" is the default and the most common for ldap servers. If the mail field is not found the username will be used as the email address when creating the eperson object.
Property:	<i>ldap.surname_field</i>
Example Value:	<i>ldap.surname_field = sn</i>
Informational Note:	This is the LDAP object field where the user's last name is stored. "sn" is the default and is the most common for LDAP servers. If the field is not found the field will be left blank in the new eperson object.
Property:	<i>ldap.givenname_field</i>
Example Value:	<i>ldap.givenname_field = givenName</i>
Informational Note:	This is the LDAP object field where the user's given names are stored. I'm not sure how common the givenName field is in different LDAP instances. If



	the field is not found the field will be left blank in the new eperson object.
Property:	<i>ldap.phone_field</i>
Example Value:	<i>ldap.phone_field = telephoneNumber</i>
Informational Note:	This is the field where the user's phone number is stored in the LDAP directory. If the field is not found the field will be left blank in the new eperson object.
Property:	<i>webui.ldap.autoregister</i>
Example Value:	<i>webui.ldap.autoregister = true</i>
Informational Note:	This will turn LDAP autoregistration on or off. With this on, a new EPerson object will be created for any user who successfully authenticates against the LDAP server when they first login. With this setting off, the user must first register to get an EPerson object by entering their ldap username and password and filling out the forms.
LDAP Users Group	
Property:	<i>ldap.login.specialgroup</i>
Example Value:	<i>ldap.login.specialgroup = group-name</i>
Informational Note:	If required, a group name can be given here, and all users who log into LDAP will automatically become members of this group. This is useful if you want a group made up of all internal authenticated users. (Remember to log on as the administrator, add this to the "Groups" with read rights).

Hierarchical LDAP Settings. # If your users are spread out across a hierarchical tree on your LDAP server, you will need to use the following stackable authentication class:

```
plugin.sequence.org.dspace.authenticate.AuthenticationMethod = \
org.dspace.authenticate.LDAPHierarchicalAuthentication
```

You can optionally specify the search scope. If anonymous access is not enabled on your LDAP server, you will need to specify the full DN and password of a user that is allowed to bind in order to search for the users.

Property:	<i>ldap.search_scope</i>
Example Value:	<i>ldap.search_scope = 2</i>
Informational Note:	This is the search scope value for the LDAP search during autoregistering. This will depend on your LDAP server setup. This value must be one of the following integers corresponding to the following values: <pre>object scope : 0 one level scope : 1 subtree scope : 2</pre>
Property:	<i>ldap.search.user</i> <i>ldap.search.password</i>
Example Value:	<i>ldap.search.user =</i> <i>cn=admin,ou=people,o=myu.edu</i>



	<code>ldap.search.password = password</code>
Informational Note:	The full DN and password of a user allowed to connect to the LDAP server and search for the DN of the user trying to log in. If these are not specified, the initial bind will be performed anonymously.
Property:	<code>_ldap.netid_email_domain = _</code>
Example Value:	<code>ldap.netid_email_domain = @example.com</code>
Informational Note:	If your LDAP server does not hold an email address for a user, you can use the following field to specify your email domain. This value is appended to the netid in order to make an email address. E.g. a netid of 'user' and <code>ldap.netid_email_domain</code> as <code>@example.com</code> would set the email of the user to be <code>user@example.com</code>

5.3.9. Shibboleth Authentication Configuration Settings

Detailed instructions for installing Shibboleth on DSpace may be found at <https://mams.melcoe.mq.edu.au/zope/mams/pub/Installation/dspace15>.

DSpace requires email as the user's credentials. There are two ways of providing email to DSpace:

1. By explicitly specifying to the user which attribute (header) carries the email address.
2. By turning on the `user-email-using-tomcat=true` which means the software will attempt to acquire the user's email from Tomcat.

The first option takes **Precedence** when specified. both options can be enabled to allow for fallback.

Property:	<code>authentication.shib.email-header</code>
Example Value:	<code>authentication.shib.email-header = MAIL</code>
Informational Note:	The option specifies that the email comes from the mentioned header. This value is CASE-Sensitive.
Property:	<code>authentication.shib.firstname-header</code>
Example Value:	<code>authentication.shib.firstname-header = SHIB-EP-GIVENNAME</code>
Informational Note:	Optional. Specify the header that carries the user's first name. This is going to be used for the creation of new-user.
Property:	<code>authentication.shib.lastname-header</code>
Example Value:	<code>authentication.shib.lastname-header = SHIB-EP-SURNAME</code>
Informational Note:	Optional. Specify the header that carries user's last name. This is used for creation of new user.
Property:	<code>authentication.shib.email-use-tomcat-remote-user</code>
Example Value:	<code>authentication.shib.email-use-tomcat-remote-user = true</code>



Informational Note:	This option forces the software to acquire the email from Tomcat.
Property:	<i>authentication.shib.autoregister</i>
Example Value:	<i>authentication.shib.autoregister = true</i>
Informational Note:	Option will allow new users to be registered automatically if the IdP provides sufficient information (and the user does not exist in DSpace)
Property:	authentication.shib.role-header authentication.shib.role.header.ignore-scope
Example Value:	authentication.shib.role-header = Shib-EP-ScopedAffiliation authentication.shib.role.header.ignore-scope = true or authentication.shib.role-header = Shib-EP-UnscopedAffiliation authentication.shib.role.header.ignore-scope = false
Informational Note:	These two options specify which attribute that is responsible for providing user's roles to DSpace and unscope the attributes if needed. When not specified, it is defaulted to 'Shib-EP-UnscopedAffiliation', and ignore-scope is defaulted to 'false'. The value is specified in AAP.xml (Shib 1.3.x) or attribute-filter.xml (Shib 2.x). The value is CASE-Sensitive. The values provided in this header are separated by semi-colon or comma. If your sp only provides scoped role header, you need to set authentication.shib.role-header.ignore-scope as true. For example if you only get Shib-EP-ScopedAffiliation instead of Shib-EP-ScopedAffiliation, you name to make your settings as in the example value above.
Property:	<i>authentication.shib.default-roles</i>
Example Value:	<i>authentication.shib.default-roles = Staff, Walk-ins</i>
Informational Note:	When user is fully authN or IdP but would not like to release his/her roles to DSpace (for privacy reasons?), what should the default roles be given to such user. The values are separated by semi-colon or comma.
Property:	authentication.shib.role.Senior\ Researcher authentication.shib.role.Librarian
Example Value:	authentication.shib.role.Senior\ Researcher = Researcher, Staff



	<code>authentication.shib.role.Librarian = Administrator</code>
Informational Note:	The following mappings specify role mapping between IdP and Dspace. The left side of the entry is IdP's role (prefixed with "authentication.shib.role.") which will be mapped to the right entry from DSpace. DSpace's group as indicated on the right entry has to EXIST in DSpace, otherwise user will be identified as 'anonymous'. Multiple values on the right entry should be separated by comma. The values are CASE-Sensitive. Heuristic one-to-one mapping will be done when the IdP groups entry are not listed below (i.e. if "X" group in IdP is not specified here, then it will be mapped to "X" group in DSpace if it exists, otherwise it will be mapped to simply 'anonymous'). Given sufficient demand, future release could support regex for the mapping special characters need to be escaped by '\'

5.3.10. Logging Configuration

The following settings are currently **not** used by XMLUI. XMLUI writes its logs to `[dspace-xmlui]/WEB-INF/logs/` in the actual XMLUI web application.

Property:	<code>log.init.config</code>
Example Value:	<code>log.init.config = \${dspace.dir}/config/log4j.properties</code>
Informational Note:	This is where your configure file is located. You may override default log4j configuration by provided your own configurations. Existing alternatives are: <code>log.init.config = \${dspace.dir}/config/log4j.xml</code> <code>log.init.config = \${dspace.dir}/config/log4j-console.properties</code>
Property:	<code>log.dir</code>
Example value:	<code>log.dir = \${dspace.dir}/log</code>
Informational Note:	This is where to put the logs. (This is used for initial configuration only)

5.3.11. Configuring Lucene Search Indexes

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

Property:	<code>search.dir</code>
Example Value:	<code>search.dir = \${dspace.dir}/search</code>
Informational Note	Where to put the search index files



Property:	<i>search.max-clauses</i>
Example Value:	<i>search.max-clauses = 2048</i>
Informational Note	By setting higher values of <i>search.max-clauses</i> will enable prefix searches to work on larger repositories.
Property:	<i>search.analyzer</i>
Example Value:	<i>search.analyzer = org.dspace.search.DSAnalyzer</i>
Informational Note	Which Lucene Analyzer implementation to use. If this is omitted or commented out, the standard DSpace analyzer (designed for English) is used by default.
Property:	<i>search.analyzer</i>
Example Value:	<pre>search.analyzer = \ org.apache.lucene.analysis.cn.ChineseAnalyzer</pre>
Informational Note	Instead of the standard English analyzer, the Chinese analyzer is used.
Property:	<i>search.operator</i>
Example Value:	<i>search.operator = OR</i>
Informational Note	Boolean search operator to use. The currently supported values are OR and AND. If this configuration item is missing or commented out, OR is used. AND requires all the search terms to be present. OR requires one or more search terms to be present.
Property:	<i>search.maxfieldlength</i>
Example Value:	<i>search.maxfieldlength = 10000</i>
Informational Note	This is the maximum number of terms indexed for a single field in Lucene. The default is 10,000 words—often not enough for full-text indexing. If you change this, you will need to re-index for the change to take effect on previously added items. <i>-1</i> = unlimited (Integer.MAG_VALUE)
Property:	<i>search.index.n</i>
Example Value:	<i>search.index.1 = author:dc.contributor.*</i>
Informational Note	See the details of this particular entry below.

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

```
search.index.1 = author:dc.contributor.*
search.index.2 = author:dc.creator.*
search.index.3 = title:dc.title.*
search.index.4 = keyword:dc.subject.*
search.index.5 = abstract:dc.description.abstract
search.index.6 = author:dc.description.statementsofresponsibility
```



```

search.index.7 = series:dc.relation.ispartofseries
search.index.8 = abstract:dc.description.tableofcontents
search.index.9 = mime:dc.format.mimetype
search.index.10 = sponsor:dc.description.sponsorship
search.index.11 = id:dc.identifier.*
search.index.11 = language:dc.language.iso

```


The format of each entry is *search.index.<id> = <search label> : <schema> . <metadata field>* where:

<i><id></i>	is an incremental number to distinguish each search index entry
<i><search label></i>	is the identifier for the search field this index will correspond to
<i><schema></i>	is the schema used. Dublin Core (DC) is the default. Others are possible.
<i><metadata field></i>	is the DSpace metadata field to be indexed.

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

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

While the indexes are created, this only affects the search results and has no effect on the search components of the user interface. One will need to customize the user interface to reflect the changes, for example, to add the a new search category to the Advanced Search.

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

5.3.12. Proxy Settings

These settings for proxy are commented out by default. Uncomment and specify both properties if proxy server is required for external http requests. Use regular host name without port number.

Property:	<code>http.proxy.host</code>
Example Value	<code>http.proxy.host = proxy.myu.edu</code>
Informational Note	Enter the host name without the port number.
Property:	<code>http.proxy.port</code>
Example Value	<code>http.proxy.port = 2048</code>
Informational Note	Enter the port number for the proxy server.

5.3.13. Configuring Media Filters

Media or Format Filters are classes used to generate derivative or alternative versions of content or bitstreams within DSpace. For example, the PDF Media Filter will extract textual content from PDF bitstreams, the JPEG Media Filter can create thumbnails from image bitstreams.



Media Filters are configured as Named Plugins, with each filter also having a separate configuration setting (in *dspace.cfg*) indicating which formats it can process. The default configuration is shown below.

Property:	<i>filter.plugins</i>
Example Value:	(See example below)
<pre>filter.plugins = PDF Text Extractor, Html Text Extractor, \ Word Text Extractor, JPEG Thumbnail</pre>	
Informational Note:	Place the names of the enabled MediaFilter or FormatFilter plugins. To enable Branded Preview, comment out the previous one line and then uncomment the two lines in found in <i>dspace.cfg</i> :
	<pre>Word Text Extractor, JPEG Thumbnail,\ Branded Preview JPEG</pre>
Property:	<i>plugin.named.org.dspace.app.mediafilter.FormatFilter</i>
Example Value:	(See example below)
<pre>plugin.named.org.dspace.app.mediafilter.FormatFilter = \ org.dspace.app.mediafilter.PDFFilter = PDF Text Extractor, \ org.dspace.app.mediafilter.HTMLFilter = HTML Text Extractor, \ org.dspace.app.mediafilter.WordFilter = Word Text Extractor, \ org.dspace.app.mediafilter.JPEGFilter = JPEG Thumbnail, \ org.dspace.app.mediafilter.BrandedPreviewJPEGFilter = Branded Preview JPEG</pre>	
Informational Note:	Assign "human-understandable" names to each filter
Property:	(See key below)
<pre>filter.org.dspace.app.mediafilter.PDFFilter.inputFormats filter.org.dspace.app.mediafilter.HTMLFilter.inputFormats filter.org.dspace.app.mediafilter.WordFilter.inputFormats filter.org.dspace.app.mediafilter.JPEGFilter.inputFormats filter.org.dspace.app.mediafilter.BrandedPreviewJPEGFilter.inputFormats</pre>	
Example Value:	(See example below)
<pre>filter.org.dspace.app.mediafilter.PDFFilter.inputFormats = Adobe PDF filter.org.dspace.app.mediafilter.HTMLFilter.inputFormats = HTML, Text filter.org.dspace.app.mediafilter.WordFilter.inputFormats = Microsoft Word filter.org.dspace.app.mediafilter.JPEGFilter.inputFormats = BMP, GIF, JPEG, \ image/png filter.org.dspace.app.mediafilter.BrandedPreviewJPEGFilter.inputFormats = BMP, \ GIF, JPEG, image/png</pre>	



Informational Note:	Configure each filter's input format(s)
Property:	<code>pdffilter.largepdfs</code>
Example Value:	<code>pdffilter.largepdfs = true</code>
Informational Note:	If this value is set for "true", all PDF extractions are written to temp files as they are indexed. This is slower, but helps to ensure that PDFBox software DSpace uses does not eat up all your memory.
Property:	<code>pdffilter.skiponmemoryexception</code>
Example Value:	<code>pdffilter.skiponmemoryexception = true</code>
Informational Note:	If this value is set for "true", PDFs which still result in an "Out of Memory" error from PDFBox are skipped over. These problematic PDFs will never be indexed until memory usage can be decreased in the PDFBox software.

Names are assigned to each filter using the `plugin.named.org.dspace.app.mediafilter.FormatFilter` field (e.g. by default the PDFFilter is named "PDF Text Extractor").

Finally, the appropriate `filter.<class path>.inputFormats` defines the valid input formats which each filter can be applied. These format names **must match** the `short description` field of the Bitstream Format Registry.

You can also implement more dynamic or configurable Media/Format Filters which extend `SelfNamedPlugin`

5.3.14. Configurable MODS Dissemination Crosswalk

The MODS crosswalk is a self-named plugin. To configure an instance of the MODS crosswalk, add a property to the DSpace configuration starting with `"crosswalk.mods.properties."`; the final word of the property name becomes the plugin's name. For example, a property name `crosswalk.mods.properties.MODS` defines a crosswalk plugin named `"MODS"`.

The value of this property is a path to a separate properties file containing the configuration for this crosswalk. The pathname is relative to the DSpace configuration directory, i.e. the `config` subdirectory of the DSpace install directory. Example from the `_dspace.cfg_file`:

Properties:	<code>crosswalk.mods.properties.MODS</code> <code>crosswalk.mods.properties.mods</code>
Example Values:	<code>crosswalk.mods.properties.MODS =</code> <code>crosswalks/mods.properties</code> <code>crosswalk.mods.properties.mods =</code> <code>crosswalks/mods.properties</code>
Informational Note:	This defines a crosswalk named MODS whose configuration comes from the file <code>[dspace]/config/crosswalks/mods.properties</code> . (In the above example, the lower-case name was added for OAI-PMH)

The MODS crosswalk properties file is a list of properties describing how DSpace metadata elements are to be turned into elements of the MODS XML output document. The property name is a concatenation of the metadata schema, element name, and optionally the qualifier. For example, the `contributor.author` element in the native Dublin Core schema would be: `dc.contributor.author`. The value of the property is a line containing two segments separated by the vertical bar ("`|`"): The first part is an XML fragment which is copied into the output document. The second is an XPath expression describing where in that fragment to put the value of the metadata element. For example, in this property:



```
dc.contributor.author = <mods:name><mods:role><mods:roleTerm
                        type="text">author</mods:roleTerm>
                        </mods:role><mods:namePart>%s</mods:;
                        <mods:namePart>%s</mods:namePart></mods:name> |
                        mods:namePart/text()
```

Some of the examples include the string "%s" in the prototype XML where the text value is to be inserted, but don't pay any attention to it, it is an artifact that the crosswalk ignores. For example, given an author named *Jack Florey*, the crosswalk will insert

```
<mods:name>
  <mods:role>
    <mods:roleTerm type="text">author</mods:roleTerm>
  </mods:role>
  <mods:namePart> Jack Florey</mods:namePart>
</mods:name>
```

into the output document. Read the example configuration file for more details.

5.3.15. XSLT-based Crosswalks

The XSLT crosswalks use XSL stylesheet transformation (XSLT) to transform an XML-based external metadata format to or from DSpace's internal metadata. XSLT crosswalks are much more powerful and flexible than the configurable MODS and QDC crosswalks, but they demand some esoteric knowledge (XSL stylesheets). Given that, you can create all the crosswalks you need just by adding stylesheets and configuration lines, without touching any of the Java code.

The default settings in the *dspace.cfg* file for submission crosswalk:

Properties:	<i>crosswalk.submission.MODS.stylesheet</i>
Example Value:	<i>crosswalk.submission.MODS.stylesheet = crosswalks/mods-submission.xsl</i>
Informational Note:	Configuration XSLT-driven submission crosswalk for MODS

As shown above, there are three (3) parts that make up the properties "key":

```
crosswalk.submissionPluginName.stylesheet =
  1         2         3         4
```


crosswalk first part of the property key. *submission* second part of the property key. *PluginName* is the name of the plugin. The *path* value is the path to the file containing the crosswalk stylesheet (relative to *[dspace]/config*).

Here is an example that configures a crosswalk named "LOM" using a stylesheet in *[dspace]/config/crosswalks/d-lom.xsl*:

```
crosswalk.submission.LOM.stylesheet = crosswalks/d-lom.xsl
```

A dissemination crosswalk can be configured by starting with the property key *crosswalk.dissemination*. Example:

```
crosswalk.dissemination.PluginName.stylesheet = path
```

The *PluginName* is the name of the plugin . The *path* value is the path to the file containing the crosswalk stylesheet (relative to *[dspace]/config*).

You can make two different plugin names point to the same crosswalk, by adding two configuration entries with the same path:



```
crosswalk.submission.MyFormat.stylesheet = crosswalks/myformat.xslt
crosswalk.submission.almost_DC.stylesheet = crosswalks/myformat.xslt
```

The dissemination crosswalk must also be configured with an XML Namespace (including prefix and URI) and an XML schema for its output format. This is configured on additional properties in the DSpace configuration:

```
crosswalk.dissemination.PluginName.namespace.Prefix = namespace-URI
crosswalk.dissemination.PluginName.schemaLocation = schemaLocation value
```

For example:

```
crosswalk.dissemination.qdc.namespace.dc = http://purl.org/dc/elements/1.1/
crosswalk.dissemination.qdc.namespace.dcterms = http://purl.org/dc/terms/
crosswalk.dissemination.qdc.schemaLocation = http://purl.org/dc/elements/1.1/ \
http://dublincore.org/schemas/xmls/qdc/2003/04/02/qualifieddc.xsd
```

Testing XSLT Crosswalks

The XSLT crosswalks will automatically reload an XSL stylesheet that has been modified, so you can edit and test stylesheets without restarting DSpace. You can test a dissemination crosswalk by hooking it up to an OAI-PMH crosswalk and using an OAI request to get the metadata for a known item.

Testing the submission crosswalk is more difficult, so we have supplied a command-line utility to help. It calls the crosswalk plugin to translate an XML document you submit, and displays the resulting intermediate XML (DIM). Invoke it with:

```
[dspace]/bin/dsrun
org.dspace.content.crosswalk.XSLTIngestionCrosswalk [-l] plugin input-file
```

where *plugin* is the name of the crosswalk plugin to test (e.g. "LOM"), and *input-file* is a file containing an XML document of metadata in the appropriate format.

Add the *-l* option to pass the ingestion crosswalk a list of elements instead of a whole document, as if the List form of the ingest() method had been called. This is needed to test ingesters for formats like DC that get called with lists of elements instead of a root element.

5.3.16. Configurable Qualified Dublin Core (QDC) dissemination crosswalk

The QDC crosswalk is a self-named plugin. To configure an instance of the QDC crosswalk, add a property to the DSpace configuration starting with "*crosswalk.qdc.properties.*"; the final word of the property name becomes the plugin's name. For example, a property name *crosswalk.qdc.properties.QDC* defines a crosswalk plugin named "*QDC*".

The following is from *dspace.cfg* file:

Properties:	<i>crosswalk.qdc.namespace.qdc.dc</i>
Example Value:	<code>_crosswalk.qdc.namespace.qdc.dc = http://purl.org/dc/elements/1.1/</code>
Properties:	<i>crosswalk.qdc.namespace.qdc.dcterms</i>
Example Value:	<code>_crosswalk.qdc.namespace.qdc.dc = http://purl.org/dc/terms/</code>
Properties:	<i>crosswalk.qdc.schemaLocation.QDC</i>
Example Value:	<code>crosswalk.qdc.schemaLocation.QDC = http://www.purl.org/dc/terms \</code>



	<pre>http://dublincore.org/schemas/xmls/ qdc/2006/01/06/dcterms.xsd \ http://purl.org/dc/elements/1.1 \ http://dublincore.org/schemas/xmls/ qdc/2006/01/06/dc.xsd</pre>
Properties:	<code>crosswalk.qdc.properties.QDC</code>
Example Value:	<code>crosswalk.qdc.properties.QDC = crosswalks/QDC.properties</code>
Informational Note:	Configuration of the QDC Crosswalk dissemination plugin for Qualified DC. (Add lower-case name for OAI-PMH. That is, change QDC to qdc.)

In the property key "`crosswalk.qdc.properties.QDC`" the value of this property is a path to a separate properties file containing the configuration for this crosswalk. The pathname is relative to the DSpace configuration directory `_[dspace]/config`. Referring back to the "Example Value" for this property key, one has `_crosswalks/qdc.properties` which defines a crosswalk named `QDC` whose configuration comes from the file `[dspace]/config/crosswalks/qdc.properties`.

You will also need to configure the namespaces and schema location strings for the XML output generated by this crosswalk. The namespaces properties names are formatted:

```
crosswalk.qdc.namespace.prefix = uri
```

where `prefix` is the namespace prefix and `uri` is the namespace URI. See the above Property and Example Value keys as the default dspace.cfg has been configured.

The QDC crosswalk properties file is a list of properties describing how DSpace metadata elements are to be turned into elements of the Qualified DC XML output document. The property name is a concatenation of the metadata schema, element name, and optionally the qualifier. For example, the `contributor.author` element in the native Dublin Core schema would be: `dc.contributor.author`. The value of the property is an XML fragment, the element whose value will be set to the value of the metadata field in the property key.

For example, in this property:

```
dc.coverage.temporal = <dcterms:temporal />
```

the generated XML in the output document would look like, e.g.:

```
<dcterms:temporal>Fall, 2005</dcterms:temporal>
```

5.3.17. Configuring Crosswalk Plugins

Ingestion crosswalk plugins are configured as named or self-named plugins for the interface `org.dspace.content.crosswalk.IngestionCrosswalk`. Dissemination crosswalk plugins are configured as named or self-named plugins for the interface `org.dspace.content.crosswalk.DisseminationCrosswalk`.

You can add names for existing crosswalks, add new plugin classes, and add new configurations for the configurable crosswalks as noted below.

5.3.18. Configuring Packager Plugins

Package ingester plugins are configured as named or self-named plugins for the interface `org.dspace.content.packager.PackageIngester`. Package disseminator plugins are configured as named or self-named plugins for the interface `org.dspace.content.packager.PackageDisseminator`.

You can add names for the existing plugins, and add new plugins, by altering these configuration properties. See the Plugin Manager architecture for more information about plugins.



5.3.19. Event System Configuration

Properties:	<code>event.dispatcher.default.class</code> <code>event.dispatcher.default.consumers</code>
Example Value:	<code>event.dispatcher.default.class = org.dspace.event.BasicDispatcher</code> <code>event.dispatcher.default.consumers = search, browse, eperson</code>
Informational Note:	This is the default synchronous dispatcher (Same behavior as traditional DSpace).
Properties:	<code>event.dispatcher.noindex.class</code> <code>event.dispatcher.noindex.consumers</code>
Example Value:	<code>event.dispatcher.noindex.class = org.dspace.event.BasicDispatcher</code> <code>event.dispatcher.noindex.consumers = eperson</code>
Informational Note:	The noindex dispatcher will not create search or browse indexes (useful for batch item imports).
Properties:	<code>event.consumer.search.class</code> <code>event.consumer.search.filters</code>
Example Value:	<i>(See example below)</i>
	<code>event.consumer.search.class = org.dspace.search.SearchConsumer</code> <code>event.consumer.search.filters = \</code> <code>Community Collection Item Bundle+Add </code> <code>Create Modify Modify_Metadata Delete Remove</code>
Informational Note:	Consumer to maintain the search index.
Properties:	<code>event.consumer.browse.class</code> <code>event.consumer.browse.filters</code>
Example Value:	<i>(See example below)</i>
	<code>event.consumer.browse.class = org.dspace.browse.BrowseConsumer</code> <code>event.consumer.browse.filters = \</code> <code>Community Collection Item Bundle+Add </code> <code>Create Modify Modify_Metadata Delete Remove</code>
Informational Note:	Consumer to maintain the browse index.
Properties:	<code>event.consumer.eperson.class</code> <code>event.consumer.eperson.filters</code>
Example Value:	<code>event.consumer.eperson.class = org.dspace.eperson.EPersonConsumer</code> <code>event.consumer.eperson.filters = EPerson</code> <code>+Create</code>
Informational Note:	Consumer related to EPerson changes
Properties:	<code>event.consumer.test.class</code> <code>event.consumer.test.filters</code>



Example Value:	<code>event.consumer.test.class = org.dspace.event.TestConsumer event.consumer.test.filters = All+All</code>
Informational Note:	Test consumer for debugging and monitoring. Commented out by default.
Properties:	<code>testConsumer.verbose</code>
Example Value:	<code>testConsumer.verbose = true</code>
Informational Note:	Set this to true to enable testConsumer messages to standard output. Commented out by default.

5.3.20. Checksum Checker Settings

DSpace now comes with a Checksum Checker script (`[dspace]/bin/checker`) which can be scheduled to verify the checksum of every item within DSpace. Since DSpace calculates and records the checksum of every file submitted to it, this script is able to determine whether or not a file has been changed (either manually or by some sort of corruption or virus). The idea being that the earlier you can identify a file has changed, the more likely you'd be able to recover it (assuming it was not a wanted change).

Property:	<code>plugin.single.org.dspace.checker.BitstreamDispatcher</code>
Example Value:	<code>plugin.single.org.dspace.checker.BitstreamDispatcher = org.dspace.checker.SimpleDispatcher</code>
Informational Note:	The Default dispatcher is case non is specified.
Property:	<code>checker.retention.default</code>
Example Value:	<code>checker.retention.default = 10y</code>
Informational Note:	This option specifies the default time frame after which all checksum checks are removed from the database (defaults to 10 years). This means that after 10 years, all successful or unsuccessful matches are removed from the database.
Property:	<code>checker.retention.CHECKSUM_MATCH</code>
Example Value:	<code>checker.retention.CHECKSUM_MATCH = 8w</code>
Informational Note:	This option specifies the time frame after which a successful “match” will be removed from your DSpace database (defaults to 8 weeks). This means that after 8 weeks, all successful matches are automatically deleted from your database (in order to keep that database table from growing too large).

5.3.21. Item Export and Download Settings

It is possible for an authorized user to request a complete export and download of a DSpace item in a compressed zip file. This zip file may contain the following:

`dublin_core.xmllicense.txtcontents (listing of the contents)_handle_file itself and the extract file if available`

The configuration settings control several aspects of this feature:

Property:	<code>org.dspace.app.itemexport.work.dir</code>
Example Value:	<code>org.dspace.app.itemexport.work.dir = \${dspace.dir}/exports</code>



Informational Note:	The directory where the exports will be done and compressed.
Property:	<i>org.dspace.app.itemexport.download.dir</i>
Example Value:	<i>org.dspace.app.itemexport.download.dir = \${dspace.dir}/exports/download</i>
Informational Note	The directory where the compressed files will reside and be read by the downloader.
Property:	<i>org.dspace.app.itemexport.life.span.hours</i>
Example Value:	<i>org.dspace.app.itemexport.life.span.hours = 48</i>
Informational Note	The length of time in hours each archive should live for. When new archives are created this entry is used to delete old ones.
Property:	<i>org.dspace.app.itemexport.max.size</i>
Example Value:	<i>org.dspace.app.itemexport.max.size = 200</i>
Informational Note	The maximum size in Megabytes (Mb) that the export should be. This is enforced before the compression. Each bitstream's size in each item being exported is added up, if their cumulative sizes are more than this entry the export is not kicked off.

5.3.22. Subscription Emails

DSpace, through some advanced installation and setup, is able to send out an email to collections that a user has subscribed. The user who is subscribed to a collection is emailed each time an item id added or modified. The following property key controls whether or not a user should be notified of a modification.

Property:	<i>eperson.subscription.onlynew</i>
Example Value:	<i>eperson.subscription.onlynew = true</i>
Informational Note:	For backwards compatibility, the subscription emails by default include any modified items. The property key is COMMENTED OUT by default.

5.3.23. Settings for the Submission Process

These settings control two aspects of the submission process: thesis submission permission and whether or not a bitstream file is required when submitting to a collection.

Property:	<i>webui.submit.blocktheses</i>
Example Value:	<i>webui.submit.blocktheses = false</i>
Informational Note:	Controls whether or not that the submission should be marked as a thesis.
Property:	<i>webui.submit.upload.required</i>
Example Value:	<i>webui.submit.upload.required = true</i>
Informational Note:	Whether or not a file is required to be uploaded during the "Upload" step in the submission process. The



	default is true. If set to "false", then the submitter (human being) has the option to skip the uploading of a file.
--	--

5.3.24. Configuring Creative Commons License

This enables the Creative Commons license step in the submission process of the JSP User Interface (JSPUI). Submitters are given an opportunity to select a Creative Commons license to accompany the item. Creative Commons license govern the use of the content. For further details, refer to the Creative Commons website at <http://creativecommons.org>.

Property:	<code>webui.submit.enable-cc</code>
Example Value:	<code>webui.submit.enable-cc = false</code>
Informational Note:	<i>Set key to "false" if you are not using CC License. Set key to "true" if you are using CC License.</i>

5.3.25. WEB User Interface Configurations

General Web User Interface Configurations

In this section of Configuration, we address the agnostic WEB User Interface that is used for JSP UI and XML UI. Some of the configurations will give information towards customization or refer you to the appropriate documentation.

Property:	<code>webui.browse.thumbnail.show</code>
Example Value:	<code>webui.browse.thumbnail.show = true</code>
Informational Note:	Controls whether to display thumbnails on browse and search result pages. If you have customized the Browse columnlist, then you must also include a "thumbnail" column in your configuration. (<i>This configuration property key is not used by XMLUI. To show thumbnails using XMLUI, you need to create a theme which displays them.</i>)
Property:	<code>webui.browse.thumbnail.maxheight</code> <code>webui.browse.thumbnail.maxwidth</code>
Example Value:	<code>webui.browse.thumbnail.maxheight = 80</code> <code>webui.browse.thumbnail.maxwidth = 80</code>
Informational Note:	This determines the maximum height and maximum width of the browse/search thumbnails in pixels (px). This only needs to be set if the thumbnails are required to be smaller than the dimensions of thumbnails generated by MediaFilter.
Property:	<code>webui.itme.thumbnail.show</code>
Example Value:	<code>webui.itme.thumbnail.show = true</code>
Informational Note:	This determines whether or not to display the thumbnail against each bitstream. (<i>This configuration property key is not used by XMLUI. To show thumbnails using XMLUI, you need to create a theme which displays them.</i>)
Property:	<code>webui.browse.thumbnail.linkbehavior</code>



Example Value:	<code>webui.browse.thumbnail.linkbehavior = item</code>
Informational Note:	This determines where should the clicking on the thumbnail from browse/search screens take the user. The only values currently supported are "item" or "bitstream".
Property:	<code>thumbnail.maxwidth</code> <code>thumbnail.maxheight</code>
Example Value:	<code>thumbnail.maxwidth 80</code> <code>thumbnail.maxheight 80</code>
Informational Note:	This is the where one sets the maximum height and width of generated thumbnails.
Property:	<code>webui.preview.enabled</code>
Example Value:	<code>webui.preview.enabled = false</code>
Informational Note:	Whether or not the user can "preview" the image.
Property:	<code>webui.preview.maxwidth</code> <code>webui.preview.maxheight</code>
Example Value:	<code>webui.preview.maxwidth = 600</code> <code>webui.preview.maxheight = 600</code>
Informational Note:	This sets the maximum dimensions for the preview image.
Property:	<code>webui.preview.brand</code>
Example Value:	<code>webui.preview.brand = My Institution Name</code>
Informational Note:	This is the brand text that will appear with the image.
Property:	<code>webui.preview.brand.abbrev</code>
Example Value:	<code>webui.preview.brand.abbrev = MyOrg</code>
Informational Note:	An abbreviated form of the full Branded Name. This will be used when the preview image cannot fit the normal text.
Property:	<code>webui.preview.brand.height</code>
Example Value:	<code>webui.preview.brand.height = 20</code>
Informational Note:	The height (in px) of the brand
Property:	<code>webui.preview.brand.font</code> <code>webui.preview.brand.fontpoint</code>
Example Value:	<code>webui.preview.brand.font = SanSerif</code> <code>webui.preview.brand.fontpoint = 12</code>
Informational Note:	The font settings for the brand text.
Property:	<code>webui.preview.dc</code>



Example Value:	<i>webui.preview.dc = rights</i>
Informational Note:	The Dublin Core field that will display along with the preview.
Property:	<i>webui.strengths.show</i>
Example Value:	<i>webui.strengths.show = false</i>
Informational Note:	Determines if communities and collections should display item counts when listed. The default behavior if omitted, is true. (<i>This configuration property key is not used by XMLUI. To show thumbnails using XMLUI, you need to create a theme which displays them</i>).
Property:	<i>webui.strengths.cache</i>
Example Value:	<i>webui.strengths.cache = false</i>
Informational Note:	When showing the strengths, should they be counted in real time, or fetched from the cache. Counts fetched in real time will perform an actual count of the database contents every time a page with this feature is requested, which will not scale. If you set the property key is set to cache ("true") you must run the following command periodically to update the count: <i>/[dspace]/bin/dsrun org.dspace.browse.ItemCounter</i> . The default is to count in real time (set to "false").

5.3.26. Browse Index Configuration

The browse indexes for DSpace can be extensively configured. This section of the configuration allows you to take control of the indexes you wish to browse, and how you wish to present the results. The configuration is broken into several parts: defining the indexes, defining the fields upon which users can sort results, defining truncation for potentially long fields (e.g. authors), setting cross-links between different browse contexts (e.g. from an author's name to a complete list of their items), how many recent submissions to display, and configuration for item mapping browse.

Property:	<i>webui.browse.index.<n></i>
Example Value:	<i>_webui.browse.index.1 = dateissued:metadata:dc.date.issued:date:full _</i>
Informational Note:	This is an example of how one "Defines the Indexes". See Defining the Indexes in the next sub-section.
Property:	<i>webui.itemlist.sort-option.<n></i>
Example Value:	<i>webui.itemlist.sort-option.1 = title:dc.title:title</i>
Informational Note:	This is an example of how one "Defines the Sort Options". See Defining Sort Options in the following sub-section.



Defining the Indexes.

DSpace arrives with four default indexes already defined: author, title, date issued, and subjects. Users may also define additional indexes or re-configure the current indexes for different levels of specificity. For example, the default entries that appear in the *dspace.cfg* as default installation:

```
webui.browse.index.1 = dateissued:metadata:dc.date.issued:date:full
webui.browse.index.2 = author:metadata:dc.contributor.*:text
webui.browse.index.3 = title:metadata:dc.title:title:full
webui.browse.index.4 = subject:metadata:dc.subject.*:text
#webui.browse.index.5 = dateaccessioned:item:dateaccessioned
```

The format of each entry is *webui.browse.index.<n> = <index name>:<metadata>:<schema prefix>.<element>.<qualifier>:<data-type field>:<sort option>*. Please notice that the punctuation is paramount in typing this property key in the *dspace.cfg* file. The following table explains each element:

Element	Definition and Options (if available)
<i>webui.browse.index.n</i>	<i>n</i> is the index number. The index numbers must start from 1 and increment continuously by 1 thereafter. Deviation from this will cause an error during install or a configuration update. So anytime you add a new browse index, remember to increase the number. (Commented out index numbers may be used over again).
<i><index name></i>	The name by which the index will be identified. You will need to update your Messages.properties file to match this field. (The form used in the Messages.properties file is: <i>browse.type.metadata.<index name></i> .
<i><metadata></i>	Only two options are available: "metadata" or "item"
<i><schema prefix></i>	The schema used for the field to be index. The default is dc (for Dublin Core).
<i><element></i>	The schema element. In Dublin Core, for example, the author element is referred to as "Contributor". The user should consult the default Dublin Core Metadata Registry table in Appendix A.
<i><qualifier></i>	This is the qualifier to the <i><element></i> component. The user has two choices: an asterisk "*" or a proper qualifier of the element . The asterisk is a wildcard and causes DSpace to index all types of the schema element. For example, if you have the element "contributor" and the qualifier "*" then you would index all contributor data regardless of the qualifier. Another example, you have the element "subject" and the qualifier "lsh" would cause the indexing of only those fields that have the qualifier "lsh". (This means you would only index Library of Congress Subject Headings and not all data elements that are subjects.
<i><datatype field></i>	This refers to the datatype of the field: <i>_date_</i> the index type will be treated as a date object <i>_title_</i> the index type will be treated like a title, which will include a link to the item page <i>_text_</i> the index type will be treated as plain text. If single mode is specified then this will link to the full mode list



Element	Definition and Options (if available)
<code><index display></code>	Choose <i>full</i> or <i>single</i> . This refers to the way that the index will be displayed in the browse listing. "Full" will be the full item list as specified by <i>webui.itemlist.columns</i> ; "single" will be a single list of only the indexed term.

If you are customizing this list beyond the default, you will need to insert the text you wish to appear in the navigation and on link and buttons. You need to edit the *Messages.properties* file. The form of the parameter(s) in the file:

```
browse.type.<index name>
```

Defining Sort Options

Sort options will be available when browsing a list of items (i.e. only in "full" mode, not "single" mode). You can define an arbitrary number of fields to sort on, irrespective of which fields you display using *web.itemlist.columns*. For example, the default entries that appear in the *dspace.cfg* as default installation:

```
webui.itemlist.sort-option.1 = title:dc.title:title
webui.itemlist.sort-option.2 = dateissued:dc.date.issued:date
webui.itemlist.sort-option.3 = dateaccessioned:dc.date.accessioned:date
```

The format of each entry is `web.browse.sort-option.<n> = <option name>:<schema prefix>.<element>.<qualifier>:<datatype>_`. Please notice the punctuation used between the different elements. The following table explains the each element:

Element	Definition and Options (if available)
<code>webui.browse.index.n</code>	<i>n</i> is an arbitrary number you choose.
<code><option name></code>	The name by which the sort option will be identified. This may be used in later configuration or to locate the message key (found in <i>Messages.properties</i> file) for this index.
<code><schema prefix></code>	The schema used for the field to be index. The default is dc (for Dublin Core).
<code><element></code>	The schema element. In Dublin Core, for example, the author element is referred to as "Contributor". The user should consult the default Dublin Core Metadata Registry table in Appendix A.
<code><qualifier></code>	This is the qualifier to the <code><element></code> component. The user has two choices: an asterisk "*" or a proper qualifier of the element.
<code><datatype field></code>	This refers to the datatype of the field: <code>_date_</code> the sort type will be treated as a date object <code>_text_</code> the sort type will be treated as plain text.

Browse Index Normalization Rule Configuration

Normalization Rules are those rules that make it possible for the indexes to intermix entries without regard to case sensitivity. By default, the display of metadata in the browse indexes are case-sensitive. In the example below, you retrieve separate entries:

Twain, Marktwain, markTWAIN, MARK



However, clicking through from either of these will result in the same set of items (i.e., any item that contains either representation in the correct field).

Property:	<i>webui.browse.metadata.case-insensitive</i>
Example Value:	<i>webui.browse.metadata.case-insensitive = true</i>
Informational Note:	This controls the normalization of the index entry. Uncommenting the option (which is commented out by default) will make the metadata items case-insensitive. This will result in a single entry in the example above. However, the value displayed may be any one of the above—depending on what representation was present in the first item indexed.

At the present time, you would need to edit your metadata to clean up the index presentation.

Other Browse Options

We set other browse values in the following section.

Property:	<i>webui.browse.value_columns.max</i>
Example Value:	<i>webui.browse.value_columns.max = 500</i>
Informational Note:	This sets the options for the size (number of characters) of the fields stored in the database. The default is 0, which is unlimited size for fields holding indexed data. Some database implementations (e.g. Oracle) will enforce their own limit on this field size. Reducing the field size will decrease the potential size of your database and increase the speed of the browse, but it will also increase the chance of mis-ordering of similar fields. The values are commented out, but proposed values for reasonably performance versus result quality. This affects the size of field for the browse value (this will affect display, and value sorting)
Property:	<i>webui.browse.sort_columns.max</i>
Example Value:	<i>webui.browse.sort_columns.max = 200</i>
Informational Note:	Size of field for hidden sort columns (this will affect only sorting, not display). Commented out as default.
Property:	<i>webui.browse.value_columns.omission_mark</i>
Example Value:	<i>webui.browse.value_columns.omission_mark = ...</i>
Informational Note:	Omission mark to be placed after truncated strings in display. The default is "...".
Property:	<i>plugin.named.org.dspace.sort.OrderFormatDelegate</i>
Example Value:	<i>plugin.named.org.dspace.sort.OrderFormatDelegate = \</i> <i>org.dspace.sort.OrderFormatTitleMarc21=title</i>
Informational Note:	This sets the option for how the indexes are sorted. All sort normalizations are carried out by the Order-



	<p>FormatDelegate. The plugin manager can be used to specify your own delegates for each datatype. The default datatypes (and delegates) are:</p> <pre>author = org.dspace.sort.OrderFormatAuthor title = org.dspace.sort.OrderFormatTitle text = org.dspace.sort.OrderFormatText</pre> <p>If you redefine a default datatype here, the configuration will be used in preferences to the default. However, if you do not explicitly redefine a datatype, then the default will still be used in addition to the datatypes you do specify. As of DSpace release 1.5.2, the multi-lingual MARC21 title ordering is configured as default, as shown in the example above. To use the previous title ordering (before release 1.5.2), comment out the configuration in your <i>dspace.cfg</i> file.</p>
--	--

5.3.27. Author (Multiple metadata value) Display

This section actually applies to any field with multiple values, but authors are the define case and example here.

Property:	<i>webui.browse.author-field</i>
Example Value:	<i>webui.browse.author-field = dc.contributor.*</i>
Informational Note:	This defines which field is the author/editor, etc. listing.

Replace *dc.contributor.** with another field if appropriate. The field should be listed in the configuration for *webui.itemlist.columns*, otherwise you will not see its effect. It must also be defined in *webui.itemlist.columns* as being of the datatype *text* otherwise the functionality will be overridden by the specific data type feature. (This setting is not used by the XMLUI as it is controlled by your theme).

Now that we know which field is our author or other multiple metadata value field we can provide the option to truncate the number of values displayed by default. We replace the remaining list of values with "et al" or the language pack specific alternative. Note that this is just for the default, and users will have the option of changing the number displayed when they browse the results. See the following table:

Property:	<i>webui.browse.author-limit</i>
Example Value:	<i>webui.browse.author-limit = <n></i>
Informational Note:	Where <i><n></i> is an integer number of values to be displayed. Use <i>-1</i> for unlimited (the default value).

5.3.28. Links to Other Browse Contexts

We can define which fields link to other browse listings. This is useful, for example, to link an author's name to a list of just that author's items. The effect this has is to create links to browse views for the item clicked on. If it is a "single" type, it will link to a view of all the items which share that metadata element in common (i.e. all the papers by a single author). If it is a "full" type, it will link to a view of the standard full browse page, starting with the value of the link clicked on.

Property:	<i>webui.browse.link.<n></i>
-----------	------------------------------------



Example Value:	<i>webui.browse.link.1 = author:dc.contributor.*</i>
Informational Note:	This is used to configure which fields should link to other browse listings. This should be associated with the name of one of the browse indexes (<i>webui.browse.index.n</i>) with a metadata field listed in <i>webui.itemlist.columns</i> above. If this condition is not fulfilled, cross-linking will not work. Note also that crosslinking only works for metadata fields not tagged as <i>title</i> in <i>webui.itemlist.columns</i> .

The format of the property key is *webui.browse.link.<n> = <index name>:<display column metadata>*
Please notice the punctuation used between the elements.

Element	Definition and Options (if available)
<i>webui.browse.link.n</i>	<i>n</i> is an arbitrary number you choose
<i><index name></i>	This need to match your entry for the index name from <i>webui.browse.index</i> property key.
<i><display column metadata></i>	Use the DC element (and qualifier)

Examples of some browse links used in a real DSpace installation instance:

*_webui.browse.link.1 = author:dc.contributor.** _Creates a link for all types of contributors (authors, editors, illustrators, others, etc.)
_webui.browse.link.2 = subject:dc.subject.lcsh _Creates a link to subjects that are Library of Congress only. In this case, you have a browse index that contains only LC Subject Headings
_webui.browse.link.3 = series:dc.relation.ispartofseries _Creates a link for the browse index "Series". Please note this is again, a customized browse index and not part of the DSpace distributed release.

5.3.29. Recent Submissions

This allows us to define which index to base Recent Submission display on, and how many we should show at any one time. This uses the PluginManager to automatically load the relevant plugin for the Community and Collection home pages. Values given in examples are the defaults supplied in *dspace.cfg*

Property:	<i>recent.submission.sort-option</i>
Example Value:	<i>recent.submission.sort-option = dateaccessioned</i>
Informational Note:	First is to define the sort name (from <i>webui.browse.sort-options</i>) to use for displaying recent submissions.
Property:	<i>recent.submissions.count</i>
Example Value:	<i>recent.submissions.count = 5</i>
Informational Note:	Defines how many recent submissions should be displayed at any one time.

There will be the need to set up the processors that the PluginManager will load to actually perform the recent submissions query on the relevant pages. This is already configured by default *dspace.cfg* so there should be no need for the administrator/programmer to worry about this.


```
plugin.sequence.org.dspace.plugin.CommunityHomeProcessor = \
    org.dspace.app.webui.components.RecentCommunitySubmissions

plugin.sequence.org.dspace.plugin.CollectionHomeProcessor = \
    org.dspace.app.webui.components.RecentCollectionSubmissions
```



5.3.30. Syndication Feed (RSS) Settings

This will enable syndication feeds—links display on community and collection home pages. This setting is not used by the XMLUI, as you enable feeds in your theme.

Property:	<i>webui.feed.enable</i>
Example Value:	<i>webui.feed.enable = false</i>
Informational Note:	By default, RSS feeds are set to false  . Change key to "true" to enable.
Property:	<i>webui.feed.items</i>
Example Value:	<i>webui.feed.items = 4</i>
Informational Note:	Defines the number of DSpace items per feed (the most recent submissions)
Property:	<i>webui.feed.cache.size</i>
Example Value:	<i>webui.feed.cache.size = 100</i>
Informational Note:	Defines the maximum number of feeds in memory cache. Value of "0" will disable caching.
Property:	<i>webui.feed.cache.age</i>
Example Value:	<i>webui.feed.cache.age = 48</i>
Informational Note:	Defines the number of hours to keep cached feeds before checking currency. The value of "0" will force a check with each request.
Property:	<i>webui.feed.formats</i>
Example Value:	<i>webui.feed.formats = rss_1.0,rss_2.0,atom_1.0</i>
Informational Note:	Defines which syndication formats to offer. You can use more than one; use a comma-separated list. The following list are the available values: rss_0.90, rss_0.91, rss_0.92, rss_0.93, rss_0.94, rss_1.0, rss_2.0, atom_1.0.
Property:	<i>webui.feed.localresolve</i>
Example Value:	<i>webui.feed.localresolve = false</i>
Informational Note:	By default, (set to false), URLs returned by the feed will point at the global handle resolver (e.g. http://hdl.handle.net/123456789/1). If set to true the local server URLs are used (e.g. http://myserver.myorg/handle/123456789/1).
Property:	<i>webui.feed.item.title</i> <i>webui.feed.item.date</i>
Example Value:	<i>webui.feed.item.title = dc.title</i> <i>webui.feed.item.date = dc.date.issued</i>



Informational Note:	This set of keys customize each single-value field displayed in the feed information for each item. Each of the fields takes a single metadata field. The form of the key is <code><scheme prefix>.<element>.<qualifier></code> In place of the qualifier, one may leave it blank to exclude any qualifiers or use the wildcard "*" to include all qualifiers for a particular element.
Property:	<code>webui.feed.item.description</code>
Example Value:	<i>(See example below)</i>
<pre>webui.feed.item.description = dc.title, dc.contributor.author, \ dc.contributor.editor, dc.description.abstract, \ dc.description</pre>	
Informational Note:	One can customize the metadata fields to show in the feed for each item's description. Elements are displayed in the order they are specified in <i>dspace.cfg</i> . Like other property keys, the format of this property key is: <code>webui.feed.item.description = <scheme prefix>.<element>.<qualifier></code> . In place of the qualifier, one may leave it blank to exclude any qualifiers or use the wildcard "*" to include all qualifiers for a particular element.
Property:	<code>webui.feed.item.author</code>
Example Value:	<code>webui.feed.item.author = dc.contributor.author</code>
Informational Note:	The name of field to use for authors (Atom only); repeatable.
Property:	<code>webui.feed.logo.url</code>
Example Value:	<code>webui.feed.logo.url = \${dspace.url}/themes/mysite/images/mysite-logo.png</code>
Informational Note:	Customize the image icon included with the site-wide feeds. This must be an absolute URL
Property:	<pre>webui.feed.item.dc.creator webui.feed.item.dc.date webui.feed.item.dc.description</pre>
Example Value:	<pre>webui.feed.item.dc.creator = dc.contributor.author webui.feed.item.dc.date = dc.date.issued webui.feed.item.dc.description = dc.description.abstract</pre>
Informational Note:	These optional properties add <i>structured</i> DC elements as XML elements to the feed description. They are not the same thing as, for example, <code>webui.feed.item.description</code> . Useful when a program or stylesheet will be transforming a feed and wants separate author, description, date, etc.



5.3.31. Content Inline Disposition Threshold

The following configuration is used to change the disposition behavior of the browser. That is, when the browser will attempt to open the file or download it to the user's specified location. For example, the default size is 8Mb. When an item being viewed is larger than 8MB, the browser will download the file to the desktop (or wherever you have it set to download) and the user will have to open it manually.

Property:	<i>webui.content_disposition_threshold</i>
Example value:	<i>webui.content_disposition_threshold = 8388608</i>
Informational Note:	The default value is set to 8Mb. This property key applies to the JSPUI interface.
Property:	<i>xmlui.content_disposition_threshold</i>
Example Value:	<i>xmlui.content_disposition_threshold = 8388608</i>
Informational Note:	The default value is set to 8Mb. This property key applies to the XMLUI (Manakin) interface.

Other values are possible:

4 MB = 4194304 MB = 838860816 MB = 1677216

5.3.32. Multi-file HTML Document/Site Settings

The setting is used to configure the "depth" of request for html documents bearing the same name.

Property:	<i>webui.html.max-depth-guess</i> <i>xmlui.html.max-depth-guess</i>
Example Value:	<i>webui.html.max-depth-guess = 3</i> <i>xmlui.html.max-depth-guess = 3</i>
Informational Note:	When serving up composite HTML items, how deep can the request be for us to serve up a file with the same name? For example, if one receives a request for " <i>foo/bar/index.html</i> " and one has a bitstream called just " <i>index.html</i> ", DSpace will serve up the former bitstream (<i>foo/bar/index.html</i>) for the request if <i>webui.html.max-depth-guess</i> is 2 or greater. If <i>webui.html.max-depth-guess</i> is 1 or less, then DSpace would not serve that bitstream, as the depth of the file is greater. If <i>webui.html.max-depth-guess</i> is zero, the request filename and path must always exactly match the bitstream name. The default is set to 3.

5.3.33. Sitemap Settings

To aid web crawlers index the content within your repository, you can make use of sitemaps.

Property:	<i>sitemap.dir</i>
Example Value:	<i>sitemap.dir = \${dspace.dir}/sitemaps</i>
Informational Note:	The directory where the generate sitemaps are stored.
Property:	<i>sitemap.engineurls</i>



Example Value:	<code>_sitemap.engineurls = http://www.google.com/webmasters/sitemaps/ping?sitemap=</code>
Informational Note:	Comma-separated list of search engine URLs to 'ping' when a new Sitemap has been created. Include everything except the Sitemap UL itself (which will be URL-encoded and appended to form the actual URL 'pinged'). Add the following to the above parameter if you have an application ID with Yahoo: http://search.yahooapis.com/SiteExplorerService/V1/updateNotification?appid=REPLACE_ME?url=. (Replace the component <code>_REPLACE_ME</code> with your application ID). There is no known 'ping' URL for MSN/Live search.

5.3.34. Upload File Settings

Property:	<code>upload.temp.dir</code>
Example Value:	<code>upload.temp.dir = \${dspace.dir}/upload</code>
Informational Note:	Where to temporarily store uploaded files.
Property:	<code>upload.max</code>
Example Value:	<code>upload.max = 536870912</code>
Informational Note:	Maximum size of uploaded files in bytes. A negative setting will result in no limit being set. The default is set for 512Mb.

5.3.35. Statistical Report Configuration Setting

Property:	<code>report.public</code>
Example Value:	<code>report.public = false</code>
Informational Note:	Controls whether or not the stats can be publicly available. Set it to false (the default) if you only want administrators to access the stats, or you do not intend to generate any statistics.
Property:	<code>report.dir</code>
Example Value:	<code>report.dir = \${dspace.dir}/reports</code>
Informational Note:	Directory where the live reports are stored.

5.3.36. JSP Web Interface (JSPUI) Settings

The following section is limited to JSPUI. If the user wishes to use XMLUI settings, please refer to Chapter 7: XMLUI Configuration and Customization.

Property:	<code>webui.licence_bundle.show</code>	
Example Value:	<code>webui.licence_bundle.show = false</code>	
Informational Note:	Sets whether to display the contents of the license bundle (often	



	just the deposit license in the standard DSpace installation).	
Property:	<i>webui.itemdisplay.default</i>	
Example Value:	(See example below)	
<pre>webui.itemdisplay.default = dc.title, dc.title.alternative, \ dc.contributor.*, dc.subject, dc.data.issued(date), \ dc.publisher, dc.identifier.citation, \ dc.relation.ispartofseries, dc.description.abstract, \ dc.description, dc.identifier.govdoc, \ dc.identifier.uri(link), dc.identifier.isbn, \ dc.identifier.issn, dc.identifier.ismn, dc.identifier</pre>		
Informational Note:	<p>This is used to customize the DC metadata fields that display in the itemdisplay (the brief display) when pulling up a record. The format is: <code><schema>.<element>.<optionalqualifier></code></p> <p>In place of the qualifier, one can use the wildcard "" to include all fields of the same element, or, leave it blank for unqualified elements. Additionally, two additional options are available for behavior/rendering: (date) and (link). See the following examples: <code>dc.title = Dublin Core element 'title'(unqualified)</code><code>dc.title.alternative = DC element 'title', qualifier 'alternative'</code><code>dc.title. = All fields with Dublin Core element 'title' (any or no qualifier)</code><code>dc.identifier.uri(link) = DC identifier.uri, rendered as a link_dc.date.issued(date)_ = DC date.issued, rendered as a date</code></p> <p>The <i>Messages.properties</i> file controls how the fields defined above will display to the user. If the field is missing from the <i>Messages.properties_file</i>, it will not be display. Look in <i>_Messages.properties</i> under <i>metadata.dc.<field></i>. Example:</p>	



	<pre>metadata.dc.contributor.other = Authors metadata.dc.contributor.author = Authors metadata.dc.title.* = Title</pre> <p><i>*Please note:</i> The order in which you place the values to the property key control the order in which they will display to the user on the outside world. (See the Example Value above).</p>	
Property:	<pre>webui.resolver.1.urn webui.resolver.1.baseurl webui.resolver.2.urn webui.resolver.2.baseurl</pre>	
Example Value:	<pre>webui.resolver.1.urn = doi webui.resolver.1.baseurl = http://dx.doi.org/ webui.resolver.2.urn = hdl webui.resolver.2.baseurl = http://hdl.handle.net/</pre>	
Informational Note:	<p>When using "resolver" in <i>webui.itemdisplay</i> to render identifiers as resolvable links, the base URL is take from <code><code>webui.resolver.<n>.baseurl</code></code> where <code><code>webui.resolver.<n>.baseurl</code></code> matches the urn specified in the metadata value. The value is appended to the "baseurl" as is, so the baseurl needs to end with the forward slash almost in any case. If no urn is specified in the value it will be displayed as simple text. For the doi and hdl urn defaults values are provided, respectively http://dc.doi.org and http://hdl.handle.net are used. If a metadata value with style "doi", "handle" or "resolver" matches a URL already, it is simply rendered as a link with no other manipulation.</p>	
Property:	<i>plugin.single.org.dspace.app.webui.util.StyleSelection</i>	
Example Value:	<pre>plugin.single.org.dspace.app.webui.util.StyleSelection = \ org.dspace.app.web.util.CollectionStyleSelection #org.dspace.app.web.util.MetadataStyleSelection</pre>	
Informational Note:	Specify which strategy to use for select the style for an item.	



Property:	<i>webui.itemdisplay.thesis.collections</i>	
Example Value:	<i>webui.itemdisplay.thesis.collections</i> = 123456789/24, 123456789/35	
Informational Note:	Specify which collections use which views by Handle.	
Property:	<i>webui.itemdisplay.metadata-style</i> <i>webui.itemdisplay.metadata-syle</i>	
Example Value:	<i>webui.itemdisplay.metadata-style</i> = schema.element[.qualifier . *] <i>webui.itemdisplay.metadata-syle</i> = dc.type	
Informational Note:	Specify which metadata to use as name of the style	
Property:	<i>webui.itemlist.columns</i>	
Example Value:	<i>webui.itemlist.columns</i> = thumbnail, dc.date.issued(date), dc.title, \ dc.contributor.*	
Informational Note:	Customize the DC fields to use in the item listing page. Elements will be displayed left to right in the order they are specified here. The form is <schema prefix>.<element>[.<qualifier> .][<(date)>], ... Although not a requirement, it would make sense to include among the listed fields at least the date and title fields as specified by the webui.browse.index. configuration options in the next section mentioned. (cf.)If you have enabled thumbnails (<i>webui.browse.thumbnail.show</i>), you must also include a 'thumbnail' entry in your columns—this is where the thumbnail will be displayed.	
Property:	<i>webui.itemlist.width</i>	
Example Value:	<i>webui.itemlist.width</i> = *, 130, 60%, 40%	
Informational Note:	You can customize the width of each column with the following line—you can have numbers (pixels) or percentages. For the 'thumbnail' column, a setting of '*' will use the max width specified for browse thumbnails (cf. <i>webui.browse.thumbnail.maxwidth</i> , <i>thumbnail.maxwidth</i>)	



Property:	<pre>webui.itemlist.browse.<index name>.sort.<sort name>.columns webui.itemlist.sort.<sort name>.columns webui.itemlist.browse.<browse name>.columns webui.itemlist.<sort or index name>.columns</pre>	
Example Value:	—	
Informational Note:	<p>You can override the DC fields used on the listing page for a given browse index and/or sort option. As a sort option or index may be defined on a field that isn't normally included in the list, this allows you to display the fields that have been indexed/sorted on. There are a number of forms the configuration can take, and the order in which they are listed below is the priority in which they will be used (so a combination of an index name and sort name will take precedence over just the browse name). In the last case, a sort option name will always take precedence over a browse index name. Note also, that for any additional columns you list, you will need to ensure there is an <i>itemlist.<field name></i> entry in the messages file.</p>	
Property:	<i>webui.itemlist.dateaccessioned.columns</i>	
Example Value:	<pre>webui.itemlist.dateaccessioned.columns = thumbnail, dc.date.accessioned(date), dc.title, dc.contributor.*</pre>	
Informational Note:	<p>This would display the date of the accession in place of the issue date whenever the dateaccessioned browsed index or sort option is selected. Just like <i>webui.itemlist.columns</i>, you will need to include a 'thumbnail' entry to display the thumbnails in the item list.</p>	
Property:	<i>webui.itemlist.dateaccessioned.widths</i>	
Example Value:	<pre>webui.itemlist.dateaccessioned.widths = *, 130, 60%, 40%</pre>	
Informational Note:	<p>As in the aforementioned property key, you can customize the</p>	



	width of the columns for each configured column list, substituting ' <i>widths</i> ' for ' <i>columns</i> ' in the property name. See the setting for <code>_webui.itemlist.widths_for</code> for more information.	
Property:	<code>webui.itemlist.tablewidth</code>	
Example Value:	<code>webui.itemlist.tablewidth = 100%</code>	
Informational Note:	You can also set the overall size of the item list table with the following setting. It can lead to faster table rendering when used with the column widths above, but not generally recommended.	

5.3.37. Configuring Multilingual Support

Setting the Default Language for the Application

Property:	<code>default.locale</code>
Example Value:	<code>default.locale = en</code>
Informational Note:	The default language for the application is set with this property key. This is a locale according to i18n and might consist of country, country_language or country_language_variant. If no default locale is defined, then the server default locale will be used. The format of a local specifier is described here: http://java.sun.com/j2se/1.4.2/docs/api/java/util/Locale.html

Supporting More Than One Language

Changes in dspace.cfg

Property:	<code>webui.supported.locale</code>
Example Value:	<code>webui.supported.locale = en, de</code> or perhaps <code>_webui.supported.locals = en, en_ca, de_</code>
Informational Note:	All the locales that are supported by this instance of DSpace. Comma separated list.

The table above, if needed and is used will result in:

- a language switch in the default header
- the user will be enabled to choose his/her preferred language, this will be part of his/her profile
- wording of emails
 - mails to registered users, e.g. alerting service will use the preferred language of the user
 - mails to unregistered users, e.g. suggest an item will use the language of the session



- according to the language selected for the session, using dspace-admin Edit News will edit the news file of the language according to session

Related Files

If you set `webui.supported.locales` make sure that all the related additional files for each language are available. *LOCALE* should correspond to the locale set in `webui.supported.locales`, e. g.: for `webui.supported.locales = en, de, fr`, there should be:

- `[dspace-source]/dspace/modules/jspui/src/main/resources/Messages.properties`
 - `[dspace-source]/dspace/modules/jspui/src/main/resources/Messages_en.properties`
 - `[dspace-source]/dspace/modules/jspui/src/main/resources/Messages_de.properties`
 - `[dspace-source]/dspace/modules/jspui/src/main/resources/Messages_fr.properties`
- Files to be localized:
- `[dspace-source]/dspace/modules/jspui/src/main/resources/Messages_LOCALE.properties`
 - `[dspace-source]/dspace/config/input-forms_LOCALE.xml`
 - `[dspace-source]/dspace/config/default_LOCALE.licenses` should be pure ascii
 - `[dspace-source]/dspace/config/news-top_LOCALE.html`
 - `[dspace-source]/dspace/config/news-side_LOCALE.html`
 - `[dspace-source]/dspace/config/emails/change_password_LOCALE`
 - `[dspace-source]/dspace/config/emails/feedback_LOCALE`
 - `[dspace-source]/dspace/config/emails/internal_error_LOCALE`
 - `[dspace-source]/dspace/config/emails/register_LOCALE`
 - `[dspace-source]/dspace/config/emails/submit_archive_LOCALE`
 - `[dspace-source]/dspace/config/emails/submit_reject_LOCALE`
 - `[dspace-source]/dspace/config/emails/submit_task_LOCALE`
 - `[dspace-source]/dspace/config/emails/subscription_LOCALE`
 - `[dspace-source]/dspace/config/emails/suggest_LOCALE`
 - `[dspace]/webapps/jspui/help/collection-admin_LOCALE.html` in html keep the jump link as original; must be copied to `[dspace-source]/dspace/modules/jspui/src/main/webapp/help`
 - `[dspace]/webapps/jspui/help/index_LOCALE.html` must be copied to `[dspace-source]/dspace/modules/jspui/src/main/webapp/help`
 - `[dspace]/webapps/jspui/help/site-admin_LOCALE.html` must be copied to `[dspace-source]/dspace/modules/jspui/src/main/webapp/help`

5.3.38. Item Mapper

Because the item mapper requires a primitive implementation of the browse system to be present, we simply need to tell that system which of our indexes defines the author browse (or equivalent) so that the mapper can list authors' items for mapping



Define the index name (from *webui.browse.index*) to use for displaying items by author.

Property:	<i>itemmap.author.index</i>
Example Value:	<i>itemmap.author.index = author</i>
Informational Note:	If you change the name of your author browse field, you will also need to update this property key.

5.3.39. Display of Group Membership

Property:	<i>webui.mydspace.showgroupmembership</i>
Example Value:	<i>webui.mydspace.showgroupmembership = false</i>
Informational Note:	To display group membership set to "true". If omitted, the default behavior is false.

5.3.40. SFX Server

SFX Server is an OpenURL Resolver.

Property:	<i>sfx.server.url</i>
Example Value:	<i>_sfx.server.url = http://sfx.myu.edu:8888/sfx?</i>
Informational Note:	SFX query is appended to this URL. If this property is commented out or omitted, SFX support is switched off.

H3. Item Recommendation Setting

Property:	<i>webui.suggest.enable</i>
Example Value:	<i>webui.suggest.enable = true</i>
Informational Note:	Show a link to the item recommendation page from item display page.
Property:	<i>webui.suggest.loggedinusers.only</i>
Example Value:	<i>webui.suggest.loggedinusers.only = true</i>
Informational Note:	Enable only if the user is logged in. If this key commented out, the default value is false.

H3. Controlled Vocabulary Settings

DSpace now supports controlled vocabularies to confine the set of keywords that users can use while describing items.

Property:	<i>webui.controlledvocabulary.enable</i>
Example Value:	<i>webui.controlledvocabulary.enable = true</i>
Informational Note:	Enable or disable the controlled vocabulary add-on. WARNING: This feature is not compatible with WAI (it requires javascript to function).

The need for a limited set of keywords is important since it eliminates the ambiguity of a free description system, consequently simplifying the task of finding specific items of information.



The controlled vocabulary add-on allows the user to choose from a defined set of keywords organized in a tree (taxonomy) and then use these keywords to describe items while they are being submitted.

We have also developed a small search engine that displays the classification tree (or taxonomy) allowing the user to select the branches that best describe the information that he/she seeks.

The taxonomies are described in XML following this (very simple) structure:

```
<node id="acmccs98" label="ACMCCS98">
  <isComposedBy>
    <node id="A." label="General Literature">
      <isComposedBy>
        <node id="A.0" label="GENERAL"/>
        <node id="A.1" label="INTRODUCTORY AND SURVEY"/>
      </isComposedBy>
    </node>
  </isComposedBy>
</node>
```

You are free to use any application you want to create your controlled vocabularies. A simple text editor should be enough for small projects. Bigger projects will require more complex tools. You may use Protegé to create your taxonomies, save them as OWL and then use a XML Stylesheet (XSLT) to transform your documents to the appropriate format. Future enhancements to this add-on should make it compatible with standard schemas such as OWL or RDF.

In order to make DSpace compatible with WAI 2.0, the add-on is **turned off** by default (the add-on relies strongly on Javascript to function). It can be activated by setting the following property in *dspace.cfg*:

```
webui.controlledvocabulary.enable = true
```

New vocabularies should be placed in *[dspace]/config/controlled-vocabularies/* and must be according to the structure described. A validation XML Schema can be downloaded *controlledvocabulary.xsd*.

Vocabularies need to be associated with the correspondent DC metadata fields. Edit the file *[dspace]/config/input-forms.xml* and place a "vocabulary" tag under the "field" element that you want to control. Set value of the "vocabulary" element to the name of the file that contains the vocabulary, leaving out the extension (the add-on will only load files with extension "*.xml"). For example:

```
<field>
  <dc-schema>dc</dc-schema>
  <dc-element>subject</dc-element>
  <dc-qualifier></dc-qualifier>
  <!-- An input-type of twobox MUST be marked as repeatable -->
  <repeatable>true</repeatable>
  <label>Subject Keywords</label>
  <input-type>twobox</input-type>
  <hint> Enter appropriate subject keywords or phrases below. </hint>
  <required></required>
  <vocabulary [closed="false"]>nsi</vocabulary>
</field>
```

The vocabulary element has an optional boolean attribute **closed** that can be used to force input only with the javascript of controlled-vocabulary add-on. The default behavior (i.e. without this attribute) is as set **closed="false"**. This allow the user also to enter the value in free way.

The following vocabularies are currently available by default:

- **nsi** - *nsi.xml* - The Norwegian Science Index
- **srsc** - *srsc.xml* - Swedish Research Subject Categories

5.3.41. XMLUI Specific Configuration

The DSpace digital repository supports two user interfaces: one based upon JSP technologies and the other based upon the Apache Cocoon framework. This section describes those configurations settings which are



specific to the XMLUI interface based upon the Cocoon framework. (Prior to DSpace Release 1.5.1 XMLUI was referred to Manakin. You may still see references to "Manakin")

Property:	<i>xmlui.supported.locales</i>
Example Value:	<i>xmlui.supported.locales = en, de</i>
Informational Note:	A list of supported locales for Manakin. Manakin will look at a user's browser configuration for the first language that appears in this list to make available to in the interface. This parameter is a comma separated list of Locales. All types of Locales country, country_language, country_language_variant. Note that if the appropriate files are not present (i.e. Messages_XX_XX.xml) then Manakin will fall back through to a more general language.
Property:	<i>xmlui.force.ssl</i>
Example Value:	<i>xmlui.force.ssl = true</i>
Informational Note:	Force all authenticated connections to use SSL, only non-authenticated connections are allowed over plain http. If set to true, then you need to ensure that the 'dspace.hostname' parameter is set to the correctly.
Property:	<i>xmlui.user.registration</i>
Example Value:	<i>xmlui.user.registration = true</i>
Informational Note:	Determine if new users should be allowed to register. This parameter is useful in conjunction with Shibboleth where you want to disallow registration because Shibboleth will automatically register the user. Default value is true.
Property:	<i>xmlui.user.editmetadata</i>
Example Value:	<i>xmlui.user.editmetadata = true</i>
Informational Note:	Determines if users should be able to edit their own metadata. This parameter is useful in conjunction with Shibboleth where you want to disable the user's ability to edit their metadata because it came from Shibboleth. Default value is true.
Property:	<i>xmlui.user.assumelogon</i>
Example Value:	<i>xmlui.user.assumelogon = true</i>
Informational Note:	Determine if super administrators (those whom are in the Administrators group) can login as another user from the "edit eperson" page. This is useful for debugging problems in a running dspace instance, especially in the workflow process. The default value is false, i.e., no one may assume the login of another user.
Property:	<i>xmlui.user.loginredirect</i>



Example Value:	<i>xmlui.user.loginredirect = /profile</i>
Informational Note:	After a user has logged into the system, which url should they be directed? Leave this parameter blank or undefined to direct users to the homepage, or <i>/profile</i> for the user's profile, or another reasonable choice is <i>/submissions</i> to see if the user has any tasks awaiting their attention. The default is the repository home page.
Property:	<i>xmlui.theme.allowoverrides</i>
Example Value:	<i>xmlui.theme.allowoverrides = false</i>
Informational Note:	Allow the user to override which theme is used to display a particular page. When submitting a request add the HTTP parameter "themepath" which corresponds to a particular theme, that specified theme will be used instead of the any other configured theme. Note that this is a potential security hole allowing execution of unintended code on the server, this option is only for development and debugging it should be turned off for any production repository. The default value unless otherwise specified is "false".
Property:	<i>xmlui.bundle.upload</i>
Example Value:	<i>xmlui.bundle.upload = ORIGINAL, METADATA, THUMBNAIL, LICENSE, CC_LICENSE</i>
Informational Note:	Determine which bundles administrators and collection administrators may upload into an existing item through the administrative interface. If the user does not have the appropriate privileges (add and write) on the bundle then that bundle will not be shown to the user as an option.
Property:	<i>xmlui.community-list.render.full</i>
Example Value:	<i>xmlui.community-list.render.full = true</i>
Informational Note:	On the community-list page should all the metadata about a community/collection be available to the theme. This parameter defaults to true, but if you are experiencing performance problems on the community-list page you should experiment with turning this option off.
Property:	<i>xmlui.community-list.cache</i>
Example Value:	<i>xmlui.community-list.cache = 12 hours</i>
Informational Note:	Normally, Manakin will fully verify any cache pages before using a cache copy. This means that when the community-list page is viewed the database is queried for each community/collection to see if their metadata has been modified. This can be expensive for repositories with a large community tree. To help solve this problem you can set the cache to be assumed valued for a specific set of time. The downside



	of this is that new or editing communities/collections may not show up the website for a period of time.
Property:	<i>xmlui.bitstream.mods</i>
Example Value:	<i>xmlui.bitstream.mods = true</i>
Informational Note:	Optionally, you may configure Manakin to take advantage of metadata stored as a bitstream. The MODS metadata file must be inside the "METADATA" bundle and named MODS.xml. If this option is set to 'true' and the bitstream is present then it is made available to the theme for display.
Property:	<i>xmlui.bitstream.mets</i>
Example Value:	<i>xmlui.bitstream.mets = true</i>
Informational Note:	Optionally, you may configure Manakin to take advantage of metadata stored as a bitstream. The METS metadata file must be inside the "METADATA" bundle and named METS.xml. If this option is set to "true" and the bitstream is present then it is made available to the theme for display.
Property:	<i>xmlui.google.analytics.key</i>
Example Value:	<i>xmlui.google.analytics.key = UA-XXXXXX-X</i>
Informational Note:	If you would like to use google analytics to track general website statistics then use the following parameter to provide your analytics key. First sign up for an account at http://analytics.google.com , then create an entry for your repositories website. Google Analytics will give you a snippet of javascript code to place on your site, inside that snippet it is your google analytics key usually found in the line: <code>_uacct = "UA-XXXXXXXX-X"</code> Take this key (just the UA-XXXXXXXX-X part) and place it here in this parameter.
Property:	<i>xmlui.controlpanel.activity.max</i>
Example Value:	<i>xmlui.controlpanel.activity.max = 250</i>
Informational Note:	Assign how many page views will be recorded and displayed in the control panel's activity viewer. The activity tab allows an administrator to debug problems in a running DSpace by understanding who and how their dspace is currently being used. The default value is 250.
Property:	<i>xmlui.controlpanel.activity.ipheader</i>
Example Value:	<i>xmlui.controlpanel.activity.ipheader = X-Forward-For</i>
Informational Note:	Determine where the control panel's activity viewer receives an events IP address from. If your DSpace is in a load balanced environment or otherwise behind



a context-switch then you will need to set the parameter to the HTTP parameter that records the original IP address.

5.3.42. OAI-PMH Configuration and Activation

In the following sections, you will learn how to configure OAI-PMH and activate additional OAI-PMH crosswalks. The user is also referred to 9.2OAI-PMH Data Provider for greater depth details of the program.

OAI-PMH Configuration

Property:	<i>oai.didl.maxresponse</i>
Example Value:	<i>oai.didl.maxresponse = 0</i>
Informational Note:	Max response size for DIDL. This is the maximum size in bytes of the files you wish to enclose Base64 encoded in your responses, remember that the base64 encoding process uses a lot of memory. We recommend at most 200000 for answers of 30 records each on a 1 Gigabyte machine. Ultimately this will change to a streaming model and remove this restriction. Also please remember to allocate plenty of memory, at least 512 MB to your Tomcat. Optional: DSpace uses 100 records as the limit for the oai responses. You can alter this by changing <code>[/dspace-source]/dspace-oai/dspace-oai-api/src/main/java/org/dspace/app/oai/DSpaceOAIcatalog.java</code> to codify the declaration: <code>private final int MAX_RECORDS = 100</code> to <code>private final int MAX_RECORDS = 30</code>

Activating Additional OAI-PMH Crosswalks

DSpace comes with an unqualified DC Crosswalk used in the default OAI-PMH data provider. There are also other Crosswalks bundled with the DSpace distribution which can be activated by editing one or more configuration files. How to do this for each available Crosswalk is described below. The DSpace source includes the following crosswalk plugins available for use with OAI-PMH:

- **mets** - The manifest document from a DSpace METS SIP.
- **mods** - MODS metadata, produced by the table-driven MODS dissemination crosswalk.
- **qdc** - Qualified Dublin Core, produced by the configurable QDC crosswalk. Note that this QDC does *not* include all of the DSpace "dublin core" metadata fields, since the XML standard for QDC is defined for a different set of elements and qualifiers.

OAI-PMH crosswalks based on Crosswalk Plugins are activated as follows:

1. Ensure the crosswalk plugin has a *lower-case* name (possibly in addition to its upper-case name) in the plugin configuration.
2. Add a line to the file `config/templates/oaicat.properties` of the form: `Crosswalks.plugin_name=org.dspace.app.oai.PluginCrosswalk` substituting the plugin's name, e.g. `"mets"` or `"qdc"` for `_plugin_name`.
3. Run the `bin/install-configs` script
4. Restart your servlet container, e.g. Tomcat, for the change to take effect. DIDL



By activating the DIDL provider, DSpace items are represented as MPEG-21 DIDL objects. These DIDL objects are XML documents that wrap both the Dublin Core metadata that describes the DSpace item and its actual bitstreams. A bitstream is provided inline in the DIDL object in a base64 encoded manner, and/or by means of a pointer to the bitstream. The data provider exposes DIDL objects via the metadataPrefix didl.

The crosswalk does not deal with special characters and purposely skips dissemination of the *license.txt* file awaiting a better understanding on how to map DSpace rights information to MPEG21-DIDL.

The DIDL Crosswalk can be activated as follows:

1. Uncomment the *oai.didl.maxresponse* item in *dspace.cfg*
2. Uncomment the DIDL Crosswalk entry from the *config/templates/oai/properties* file
3. Run the *bin/install-configs* script
4. Restart Tomcat
5. Verify the Crosswalk is activated by accessing a URL such as <http://myspace/oai-request?verb=ListRecords&metadataPrefix=didl>

5.3.43. Delegation Administration

(Authorization System Configuration)|Property: `|core.authorization.community-admin.create-subelement|`

Example Value:	<code>core.authorization.community-admin.create-subelement = true</code>
Informational Note:	It is now possible to delegate the administration of Communities and Collections without the need of the Administrator Superuser. The delegation uses an "inherited" technique. A community admin will be a collection admin for all the collections within the community and a collection admin will be always an item admin for all the item owned by the collection. All the functions that are allowed to user with WRITE permission on an object will always allowed to be the ADMIN of the object (e.g. community/collection/admin will be always allowed to edit metadata of the object). The default will be "on" for all the configurations.
Community Administration: subcommunities and collections	<code>core.authorization.community-admin.create-subelement</code> <code>core.authorization.community-admin.delete-subelement</code>
then administers the following keys:	<code>core.authorization.community-admin.policies</code> <code>core.authorization.community-admin.admin-group</code>
Collections in the above community:	<code>core.authorization.community-admin.collection.policies</code> <code>core.authorization.community-admin.collection.template-item</code> <code>core.authorization.community-admin.collection.submitters</code> <code>core.authorization.community-admin.collection.workflows</code> <code>core.authorization.community-admin.collection.admin-group</code>
Item owned by Collections in the above Community:	<code>core.authorization.community-admin.item.delete</code>



	<pre>core.authorization.community-admin.item.withdraw core.authorization.community-admin.item.reinstatiate core.authorization.community-admin.item.policies</pre> <p>And also these bundles:</p> <pre>core.authorization.community-admin.item.create-bitstream core.authorization.community-admin.item.delete-bitstream core.authorization.community-admin.item-admin.cc-license</pre>
<p>Collection Administration:</p>	<pre>core.authorization.collection-admin.policies core.authorization.collection-admin.template-item core.authorization.collection-admin.submitters core.authorization.collection-admin.workflows core.authorization.collection-admin.admin-group</pre> <p>Item owned by the above Collection</p> <pre>core.authorization.collection-admin.item.delete core.authorization.collection-admin.item.withdraw core.authorization.collection-admin.item.reinstatiate core.authorization.collection-admin.item.policies</pre> <p>And also these bundles:</p> <pre>core.authorization.collection-admin.item.create-bitstream core.authorization.collection-admin.item.delete-bitstream core.authorization.collection-admin.item-admin.cc-license</pre>
<p>Item Administration:</p>	<pre>core.authorization.item-admin.policies</pre> <p>And also these bundles:</p> <pre>core.authorization.item-admin.create-bitstream core.authorization.item-admin.delete-bitstream core.authorization.item-admin.cc-license</pre>

Oracle users should consult *Chapter 4 Updating a DSpace Installation* regarding the necessary database changes that need to take place.

5.3.44. Batch Metadata Editing

The following configurations allow the administrator extract from the DSpace database a set of records for editing by a metadata export. It provides an easier way of editing large collections.



Property:	<i>bulkedit.valueseparator</i>	
Example Value:	<code>__bulkedit.valueseparator =</code>	
Informational note	The delimiter used to separate values within a single field. For example, this will place the double pipe between multiple authors appearing in one record (Smith, William	Johannsen, Susan). This applies to any metadata field that appears more than once in a record. The user can change this to another character.
Property:	<i>bulkedit.fieldseparator</i>	
Example Value:	<code>bulkedit.fieldseparator = ,</code>	
Informational note	The delimiter used to separate fields (defaults to a comma for CSV). Again, the user could change it something like '\$'. If you wish to use a tab, semicolon, or hash (#) sign as the delimiter, set the value to be <i>tab</i> , <i>semicolon</i> or <i>hash</i> .	
	<code>bulkedit.fieldseparator = tab</code>	
Property:	<i>bulkedit.gui-item-limit</i>	
Example Value:	<code>bulkedit.gui-item-limit = 20</code>	
Informational note	When using the WEBUI, this sets the limit of the number of items allowed to be edited in one processing. There is no limit when using the CLI.	
Property:	<i>bulkedit.ignore-on-export</i>	
Example Value:	<code>bulkedit.ignore-on-export = dc.date.accessioned, \ dc.date.available, \ dc.date.updated, dc.description.provenance</code>	
Informational note	Metadata elements to exclude when exporting via the user interfaces, or when using the command line version and not using the -a (all) option.	

5.3.45. Hiding Metadata

It is now possible to hide metadata from public consumption that is only available to the Administrator.

Property:	<i>metadata.hide.dc.description.provenance</i>
Example Value:	<code>metadata.hide.dc.description.provenance = true</code>
Informational Note:	Hides the metadata in the property key above except to the administrator. Fields named here are hidden



in the following places UNLESS the logged-in user is an Administrator:

1. XMLUI metadata XML view, and Item splash pages (long and short views).
2. JSPUI Item splash pages
3. OAI-PMH server, "oai_dc" format. (Note: Other formats are ***not*** affected.) To designate a field as hidden, add a property here in the `metadata.hide.SCHEMA.ELEMENT.QUALIFIER = true`. This default configuration hides the `dc.description.provenance` field, since that usually contains email addresses which ought to be kept private and is mainly of interest to administrators.

5.4. Optional or Advanced Configuration Settings

The following section explains how to configure either optional features or advanced features that are not necessary to make DSpace "out-of-the-box"

5.4.1. The Metadata Format and Bitstream Format Registries

The `[dspace]/config/registries` directory contains three XML files. These are used to load the *initial* contents of the Dublin Core Metadata registry and Bitstream Format registry and SWORD metadata registry. After the initial loading (performed by *ant fresh_install* above), the registries reside in the database; the XML files are not updated.

In order to change the registries, you may adjust the XML files before the first installation of DSpace. On an already running instance it is recommended to change bitstream registries via DSpace admin UI, but the metadata registries can be loaded again at any time from the XML files without difficulty. The changes made via admin UI are not reflected in the XML files.

Metadata Format Registries

The default metadata schema is Dublin Core, so DSpace is distributed with a default Dublin Core Metadata Registry. Currently, the system requires that every item have a Dublin Core record.

There is a set of Dublin Core Elements, which is used by the system and should not be removed or moved to another schema, see Appendix: Default Dublin Core Metadata registry.

Note: altering a Metadata Registry has no effect on corresponding parts, e.g. item submission interface, item display, item import and vice versa. Every metadata element used in submission interface or item import must be registered before using it.

Note also that deleting a metadata element will delete all its corresponding values.

If you wish to add more metadata elements, you can do this in one of two ways. Via the DSpace admin UI you may define new metadata elements in the different available schemas. But you may also modify the XML file (or provide an additional one), and re-import the data as follows:

```
[dspace]/bin/dsrun org.dspace.administer.MetadataImporter -f [xml file]
```

The XML file should be structured as follows:

```
<dspace-dc-types>
  <dc-type>
    <schema>dc</schema>
    <element>contributor</element>
```



```
<qualifier>advisor</qualifier>
<scope_note>Use primarily for thesis advisor.</scope_note>
</dc-type>
</dspace-dc-types>
```

Bitstream Format Registry

The bitstream formats recognized by the system and levels of support are similarly stored in the bitstream format registry. This can also be edited at install-time via `[dspace]/config/registries/bitstream-formats.xml` or by the administration Web UI. The contents of the bitstream format registry are entirely up to you, though the system requires that the following two formats are present:

- *Unknown*
- *License*
Deleting a format will cause any existing bitstreams of this format to be reverted to the unknown bitstream format.

5.4.2. XPDF Filter

This is an alternative suite of MediaFilter plugins that offers faster and more reliable text extraction from PDF Bitstreams, as well as thumbnail image generation. It replaces the built-in default PDF MediaFilter.

If this filter is so much better, why isn't it the default? The answer is that it relies on external executable programs which must be obtained and installed for your server platform. This would add too much complexity to the installation process, so it left out as an optional "extra" step.

Installation Overview

Here are the steps required to install and configure the filters:

1. Install the xpdf tools for your platform, from the downloads at <http://www.foolabs.com/xpdf/>
2. Acquire the Sun Java Advanced Imaging Tools and create a local Maven package.
3. Edit DSpace configuration properties to add location of xpdf executables, reconfigure MediaFilter plugins.
4. Build and install DSpace, adding `-Pxpdf-mediafilter-support` to Maven invocation.

Install XPDF Tools

First, download the XPDF suite found at: <http://www.foolabs.com/xpdf/> and install it on your server. The executables can be located anywhere, but make a note of the full path to each command.

You may be able to download a binary distribution for your platform, which simplifies installation. Xpdf is readily available for Linux, Solaris, MacOSX, Windows, NetBSD, HP-UX, AIX, and OpenVMS, and is reported to work on AIX, OS/2, and many other systems.

The only tools you *really* need are:

- *pdftinfo* - displays properties and Info dict
- *pdftotext* - extracts text from PDF
- *pdftoppm* - images PDF for thumbnails

Fetch and install jai_imageio JAR

Fetch and install the Java™ Advanced Imaging Image I/O Tools.



For AIX, Sun support has the following: "JAI has native acceleration for the above but it also works in pure Java mode. So as long as you have an appropriate JDK for AIX (1.3 or later, I believe), you should be able to use it. You can download any of them, extract just the jars, and put those in your \$CLASSPATH."

Download the *jai_imageio* library version 1.0_01 or 1.1 found at: https://jai-imageio.dev.java.net/binary-builds.html#Stable_builds.

For these filters you do NOT have to worry about the native code, just the JAR, so choose a download for any platform.

```
curl -O http://download.java.net/media/jai-imageio/builds/release/1.1/jai_imageio-1_1-lib-linux-i586.tar.gz
tar xzf jai_imageio-1_1-lib-linux-i586.tar.gz
```

The preceding example leaves the JAR in **jai_imageio-1_1/lib/jai_imageio.jar**. Now install it in your local Maven repository, e.g.: (changing the path after *file=* if necessary)

```
mvn install:install-file \
  -Dfile=jai_imageio-1_1/lib/jai_imageio.jar \
  -DgroupId=com.sun.media \
  -DartifactId=jai_imageio \
  -Dversion=1.0_01 \
  -Dpackaging=jar \
  -DgeneratePom=true
```

You may have to repeat this procedure for the *jai_core.jar* library, as well, if it is not available in any of the public Maven repositories. Once acquired, this command installs it locally:

```
mvn install:install-file -Dfile=jai_core-1.1.2_01.jar \
  -DgroupId=javax.media -DartifactId=jai_core -Dversion=1.1.2_01 -Dpackaging=jar -
  DgeneratePom=true
```

Edit DSpace Configuration

First, be sure there is a value for *thumbnail.maxwidth* and that it corresponds to the size you want for preview images for the UI, e.g.: (*NOTE*: this code doesn't pay any attention to *thumbnail.maxheight* but it's best to set it too so the other thumbnail filters make square images.)

```
# maximum width and height of generated thumbnails
thumbnail.maxwidth 300
thumbnail.maxheight 300
```

Now, add the absolute paths to the XPDF tools you installed. In this example they are installed under */usr/local/bin* (a logical place on Linux and MacOSX), but they may be anywhere.

```
xpdf.path.pdftotext = /usr/local/bin/pdftotext
xpdf.path.pdftoppm = /usr/local/bin/pdftoppm
xpdf.path.pdfinfo = /usr/local/bin/pdfinfo
```

Change the MediaFilter plugin configuration to remove the old *org.dspace.app.mediafilter.PDFFilter* and add the new filters, e.g.: (New sections are in bold)

```
filter.plugins = \
  PDF Text Extractor, \
  PDF Thumbnail, \
  HTML Text Extractor, \
  Word Text Extractor, \
  JPEG Thumbnail
plugin.named.org.dspace.app.mediafilter.FormatFilter = \
  org.dspace.app.mediafilter.XPDF2Text = PDF Text Extractor, \
  org.dspace.app.mediafilter.XPDF2Thumbnail = PDF Thumbnail, \
  org.dspace.app.mediafilter.HTMLFilter = HTML Text Extractor, \
  org.dspace.app.mediafilter.WordFilter = Word Text Extractor, \
  org.dspace.app.mediafilter.JPEGFilter = JPEG Thumbnail, \
  org.dspace.app.mediafilter.BrandedPreviewJPEGFilter = Branded Preview JPEG
```

Then add the input format configuration properties for each of the new filters, e.g.:



```
filter.org.dspace.app.mediafilter.XPDF2Thumbnail.inputFormats = Adobe
PDFfilter.org.dspace.app.mediafilter.XPDF2Text.inputFormats = Adobe PDF
```

Finally, if you want PDF thumbnail images, don't forget to add that filter name to the *filter.plugins* property, e.g.:

```
filter.plugins = PDF Thumbnail, PDF Text Extractor, ...
```

Build and Install

Follow your usual DSpace installation/update procedure, only add *-Pxpdf-mediafilter-support* to the Maven invocation:

```
mvn -Pxpdf-mediafilter-support package
ant -Dconfig=etc. ...
```

5.4.3. Creating a new Media/Format Filter

Creating a simple Media Filter

New Media Filters **must implement** the *org.dspace.app.mediafilter.FormatFilter* interface. More information on the methods you need to implement is provided in the *FormatFilter.java* source file. For example:

```
public class MySimpleMediaFilter implements
    FormatFilter
```

Alternatively, you could extend the *org.dspace.app.mediafilter.MediaFilter* class, which just defaults to performing no pre/post-processing of bitstreams before or after filtering.

```
public class MySimpleMediaFilter extends
    MediaFilter
```

You must give your new filter a "name", by adding it and its name to the *plugin.named.org.dspace.app.mediafilter.FormatFilter* field in *dspace.cfg*. In addition to naming your filter, make sure to specify its input formats in the *filter.<class path>.inputFormats* config item. Note the input formats must match the *short description* field in the Bitstream Format Registry (i.e. *bitstreamformat-registry* table).

```
plugin.named.org.dspace.app.mediafilter.FormatFilter = \
    org.dspace.app.mediafilter.MySimpleMediaFilter = My Simple Text
Filter, \ ...
filter.org.dspace.app.mediafilter.MySimpleMediaFilter.inputFormats =
Text
```

WARNING: If you neglect to define the *_inputFormats* for a particular filter, the *MediaFilterManager* will never call that filter, since it will never find a bitstream which has a format matching that filter's input format(s).

If you have a complex Media Filter class, which actually performs different filtering for different formats (e.g. conversion from Word to PDF **and** conversion from Excel to CSV), you should define this as a [self-namedfilter|Dynamic / Self-Named Format Filter].

Creating a Dynamic or "Self-Named" Format Filter

If you have a more complex Media/Format Filter, which actually performs **multiple** filtering or conversions for different formats (e.g. conversion from Word to PDF **and** conversion from Excel to CSV), you should have define a class which implements the *FormatFilter* interface, while also extending the *business.html#selfnamedplugin* class. For example:

```
public class MyComplexMediaFilter extends
    SelfNamedPlugin implements FormatFilter
```



Since *SelfNamedPlugins* are self-named (as stated), they must provide the various names the plugin uses by defining a `getPluginNames()` method. Generally speaking, each "name" the plugin uses should correspond to a different type of filter it implements (e.g. "Word2PDF" and "Excel2CSV" are two good names for a complex media filter which performs both Word to PDF and Excel to CSV conversions).

Self-Named Media/Format Filters are also configured differently in *dspace.cfg*. Below is a general template for a Self Named Filter (defined by an imaginary *MyComplexMediaFilter* class, which can perform both Word to PDF and Excel to CSV conversions):

```
#Add to a list of all Self Named filters
plugin.selfnamed.org.dspace.app.mediafilter.FormatFilter = \
org.dspace.app.mediafilter.MyComplexMediaFilter #Define input formats
for each "named" plugin this filter implements
filter.org.dspace.app.mediafilter.MyComplexMediaFilter.Word2PDF.inputF
ormats = Microsoft Word
filter.org.dspace.app.mediafilter.MyComplexMediaFilter.Excel2CSV.input
Formats = Microsoft Excel
```

As shown above, each Self-Named Filter class must be listed in the *plugin.selfnamed.org.dspace.app.mediafilter.FormatFilter* item in *dspace.cfg*. In addition, each Self-Named Filter **must** define the input formats for *each named plugin* defined by that filter. In the above example the *MyComplexMediaFilter* class is assumed to have defined two named plugins, *Word2PDF* and *Excel2CSV*. So, these two valid plugin names ("Word2PDF" and "Excel2CSV") **must** be returned by the *getPluginNames()* method of the *MyComplexMediaFilter* class.

These named plugins take different input formats as defined above (see the corresponding *inputFormats* setting). **WARNING:** *If you neglect to define the `_inputFormats` for a particular named plugin, the *MediaFilterManager* will never call that plugin, since it will never find a bitstream which has a format matching that plugin's input format(s).*

For a particular Self-Named Filter, you are also welcome to define additional configuration settings in *dspace.cfg*. To continue with our current example, each of our imaginary plugins actually results in a different output format (Word2PDF creates "Adobe PDF", while Excel2CSV creates "Comma Separated Values"). To allow this complex Media Filter to be even more configurable (especially across institutions, with potential different "Bitstream Format Registries"), you may wish to allow for the output format to be customizable for each named plugin. For example:

```
#Define output formats for each named plugin
filter.org.dspace.app.mediafilter.MyComplexMediaFilter.Word2PDF.output
Format = Adobe PDF
filter.org.dspace.app.mediafilter.MyComplexMediaFilter.Excel2CSV.outpu
tFormat = Comma Separated Values
```

Any custom configuration fields in *dspace.cfg* defined by your filter are ignored by the *MediaFilterManager*, so it is up to your custom media filter class to read those configurations and apply them as necessary. For example, you could use the following sample Java code in your *MyComplexMediaFilter* class to read these custom *outputFormat* configurations from *dspace.cfg* :

```
//get "outputFormat" configuration from dspace.cfg
String outputFormat =
ConfigurationManager.getProperty(MediaFilterManager.FILTER_PREFIX +
"." + MyComplexMediaFilter.class.getName() + "." +
this.getPluginInstanceName() + ".outputFormat");
```

5.4.4. Configuration Files for Other Applications

To ease the hassle of keeping configuration files for other applications involved in running a DSpace site, for example Apache, in sync, the DSpace system can automatically update them for you when the main DSpace configuration is changed. This feature of the DSpace system is entirely optional, but we found it useful.

The way this is done is by placing the configuration files for those applications in *[dspace]/config/templates*, and inserting special values in the configuration file that will be filled out with appropriate DSpace configuration properties. Then, tell DSpace where to put filled-out, 'live' version of the configuration by adding an appropriate property to *dspace.cfg*, and run *[dspace]/bin/install-configs*.



Take the *apache13.conf* file as an example. This contains plenty of Apache-specific stuff, but where it uses a value that should be kept in sync across DSpace and associated applications, a 'placeholder' value is written. For example, the host name:

```
ServerName @@dSPACE.hostname@@
```

The text `@@dSPACE.hostname@@` will be filled out with the value of the *dSPACE.hostname* property in *dSPACE.cfg*. Then we decide where we want the 'live' version, that is, the version actually read in by Apache when it starts up, will go.

Let's say we want the live version to be located at */opt/apache/conf/dSPACE-httpd.conf*. To do this, we add the following property to *dSPACE.cfg* so DSpace knows where to put it:

```
config.template.apache13.conf = /opt/apache/conf/dSPACE-httpd.conf
```

Now, we run *[dSPACE]/bin/install-configs*. This reads in *[dSPACE]/config/templates/apache13.conf*, and places a copy at */opt/apache/conf/dSPACE-httpd.conf* with the placeholders filled out.

So, in */opt/apache/conf/dSPACE-httpd.conf*, there will be a line like:

```
ServerName dSPACE.myu.edu
```

The advantage of this approach is that if a property like the hostname changes, you can just change it in *dSPACE.cfg* and run *install-configs*, and all of your tools' configuration files will be updated.

However, take care to make all your edits to the versions in *[dSPACE]/config/templates*! It's a wise idea to put a big reminder at the top of each file, since someone might unwittingly edit a 'live' configuration file which would later be overwritten.

5.4.5. Configuring Usage Instrumentation Plugins

A usage instrumentation plugin is configured as a singleton plugin for the abstract class *org.dSPACE.app.statistics.AbstractUsageEvent*.

The Passive Plugin

The Passive plugin is provided as the class *org.dSPACE.app.statistics.PassiveUsageEvent*. It absorbs events without effect. Use the Passive plugin when you have no use for usage event postings. This is the default if no plugin is configured.

The Tab File Logger Plugin

The Tab File Logger plugin is provided as the class *org.dSPACE.app.statistics.UsageEventTabFileLogger*. It writes event records to a file in tab-separated column format. If left unconfigured, an error will be noted in the DSpace log and no file will be produced. To specify the file path, provide an absolute path as the value for *usageEvent.tabFileLogger.file* in *dSPACE.cfg*.

The XML Logger Plugin

The XML Logger plugin is provided as the class *org.dSPACE.app.statistics.UsageEventXMLLogger*. It writes event records to a file in a simple XML-like format. If left unconfigured, an error will be noted in the DSpace log and no file will be produced. To specify the file path, provide an absolute path as the value for *usageEvent.xmlLogger.file* in *dSPACE.cfg*.

5.4.6. SWORD Configuration

SWORD (Simple Web-service Offering Repository Deposit) is a protocol that allows the remote deposit of items into repositories. DSpace implements the SWORD protocol via the 'sword' web application. The version of SWORD currently supported by DSpace is 1.3. The specification and further information can be downloaded from <http://swordapp.org/>.



SWORD is based on the Atom Publish Protocol and allows service documents to be requested which describe the structure of the repository, and packages to be deposited.

Properties:	<i>sword.mets-ingester.package-ingester</i>
Example Value:	<i>sword.mets-ingester.package-ingester = METS</i>
Informational Note:	<p>The property key tell the SWORD METS implementation which package ingester to use to install deposited content. This should refer to one of the classes configured for:</p> <pre>plugin.named.org.dspace.content.packager.PackageIngester</pre> <p>The value of <i>sword.mets-ingester.package-ingester</i> tells the system which named plugin for this interface should be used to ingest SWORD METS packages.</p>
Properties:	<i>mets.submission.crosswalk.EPDCX</i>
Example Value:	<i>mets.submission.crosswalk.EPDCX = SWORD</i>
Informational Note:	Define the metadata type EPDCX (EPrints DC XML) to be handled by the SWORD crosswalk configuration.
Properties:	<i>crosswalk.submission.SWORD.stylesheet</i>
Example Value:	<i>crosswalk.submission.SWORD.stylesheet = crosswalks/sword-swap-ingest.xsl</i>
Informational Note:	Define the stylesheet which will be used by the self-named XSLTIngestionCrosswalk class when asked to load the SWORD configuration (as specified above). This will use the specified stylesheet to crosswalk the incoming SWAP metadata to the DIM format for ingestion.
Properties:	<i>sword.deposit.url</i>
Example Value:	<i>_sword.deposit.url = http://www.myu.ac.uk/sword/deposit</i>
Informational Note:	The base URL of the SWORD deposit. This is the URL from which DSpace will construct the deposit location urls for collections. The default is <i>{dspace.url}/sword/deposit</i> . In the event that you are not deploying DSpace as the ROOT application in the servlet container, this will generate incorrect URLs, and you should override the functionality by specifying in full as shown in the example value.
Properties:	<i>_sword.servicedocument.url _</i>
Example Value:	<i>_sword.servicedocument.url = http://www.myu.ac.uk/sword/servicedocument</i>
Informational Note:	The base URL of the SWORD service document. This is the URL from which DSpace will construct the service document location urls for the site, and for individual collections. The default is <i>{dspace.url}/</i>



	<i>sword/servicedocument</i> . In the event that you are not deploying DSpace as the ROOT application in the servlet container, this will generate incorrect URLs, and you should override the functionality by specifying in full as shown in the example value.
Properties:	<i>sword.media-link.url</i>
Example Value:	<code>_sword.media-link.url = http://www.myu.ac.uk/sword/media-link</code>
Informational Note:	The base URL of the SWORD media links. This is the URL which DSpace will use to construct the media link urls for items which are deposited via sword. The default is <i>{dspace.url}/sword/media-link</i> . In the event that you are not deploying DSpace as the ROOT application in the servlet container, this will generate incorrect URLs, and you should override the functionality by specifying in full as shown in the example value.
Properties:	<i>sword.generator.url</i>
Example Value:	<code>_sword.generator.url = http://www.dspace.org/ns/sword/1.3.1</code>
Informational Note:	The URL which identifies the sword software which provides the sword interface. This is the URL which DSpace will use to fill out the atom:generator element of its atom documents. The default is: http://www.dspace.org/ns/sword/1.3.1 . If you have modified your sword software, you should change this URI to identify your own version. If you are using the standard dspace-sword module you will not, in general, need to change this setting.
Properties:	<i>sword.updated.field</i>
Example Value:	<code>sword.updated.field = dc.date.updated</code>
Informational Note:	The metadata field in which to store the updated date for items deposited via SWORD.
Properties:	<i>sword.slug.field</i>
Example Value:	<code>sword.slug.field = dc.identifier.slug</code>
Informational Note:	The metadata field in which to store the value of the slug header if it is supplied.
Properties:	<code>sword.accept-packaging.METSDSpaceSIP.identifier sword.accept-packaging.METSDSpaceSIP.q</code>
Example Value:	<code>_(See example below) _</code>
<code>sword.accept-packaging.METSDSpaceSIP.identifier = http://purl.org/net/sword-types/ METSDSpaceSIP</code>	



<code>sword.accept-packaging.METSDSpaceSIP.q = 1.0</code>	
Informational Note:	The accept packaging properties, along with their associated quality values where appropriate. This is a Global Setting; these will be used on all DSpace collections
Properties:	<code>sword.accept-packaging.[handle].METSDSpaceSIP.identifier</code> <code>sword.accept-packaging.[handle].METSDSpaceSIP.q</code>
Example Value:	(See example below)
<code>sword.accept-packaging.[handle].METSDSpaceSIP.identifier = http://purl.org/net/sword-types/METSDSpaceSIP</code> <code>sword.accept-packaging.[handle].METSDSpaceSIP.q = 1.0</code>	
Informational Note:	Collection Specific settings: these will be used on the collections with the given handles.
Properties:	<code>sword.expose-items</code>
Example Value:	<code>sword.expose-items = false</code>
Informational Note:	Should the server offer up items in collections as sword deposit targets. This will be effected by placing a URI in the collection description which will list all the allowed items for the depositing user in that collection on request. NOTE: this will require an implementation of deposit onto items, which will not be forthcoming for a short while.
Properties:	<code>sword.expose-communities</code>
Example Value:	<code>sword.expose-communities = false</code>
Informational Note:	Should the server offer as the default the list of all Communities to a Service Document request. If false, the server will offer the list of all collections, which is the default and recommended behavior at this stage. NOTE: a service document for Communities will not offer any viable deposit targets, and the client will need to request the list of Collections in the target before deposit can continue.
Properties:	<code>sword.max-upload-size</code>
Example Value:	<code>sword.max-upload-size = 0</code>
Informational Note:	The maximum upload size of a package through the sword interface, in bytes. This will be the combined size of all the files, the metadata and any manifest data. It is NOT the same as the maximum size set for an individual file upload through the user interface. If not set, or set to 0, the sword service will default to no limit.



Properties:	<i>sword.keep-original-package</i>
Example Value:	<i>sword.keep-original-package = true</i>
Informational Note:	Whether or not DSpace should store a copy of the original sword deposit package. NOTE: this will cause the deposit process to run slightly slower, and will accelerate the rate at which the repository consumes disk space. BUT, it will also mean that the deposited packages are recoverable in their original form. It is strongly recommended, therefore, to leave this option turned on. When set to "true", this requires that the configuration option <i>upload.temp.dir</i> above is set to a valid location.
Properties:	<i>sword.bundle.name</i>
Example Value:	<i>sword.bundle.name = SWORD</i>
Informational Note:	The bundle name that SWORD should store incoming packages under if <i>sword.keep-original-package</i> is set to true. The default is "SWORD" if not value is set
Properties:	<i>sword.identify-version</i>
Example Value:	<i>sword.identify-version = true</i>
Informational Note:	Should the server identify the sword version in a deposit response. It is recommended to leave this unchanged.
Properties:	<i>sword.on-behalf-of.enable</i>
Example Value:	<i>sword.on-behalf-of.enable = true</i>
Informational Note:	Should mediated deposit via sword be supported. If enabled, this will allow users to deposit content packages on behalf of other users.
Properties:	<code>plugin.named.org.dspace.sword.SWORDIngestor</code>
Example Value:	<i>(See example below)</i>
<pre> plugin.named.org.dspace.sword.SWORDIngestor = \ org.dspace.sword.SWORDMETSIngestor = http://purl.org/net/sword-types/ METSDSpaceSIP \ org.dspace.sword.SimpleFileIngestor = SimpleFileIngestor </pre>	
Informational Note:	Configure the plugins to process incoming packages. The form of this configuration is as per the Plugin Manager's Named Plugin documentation: <i>plugin.named.[interface] = [implementation] = [package format identifier] _ . Package ingesters should implement the SWORDIngestor interface, and will be loaded when a package of the format specified above in: <i>_sword.accept-packaging.[package format].identifier = [package format identifier]</i> is received. In the event that this is a simple file de-</i>



	posit, with no package format, then the class named by "SimpleFileIngester" will be loaded and executed where appropriate. This case will only occur when a single file is being deposited into an existing DSpace Item.
Properties:	<i>sword.accepts</i>
Example Value:	<i>sword.accepts = application/zip, foo/bar</i>
Informational Note:	A comma separated list of MIME types that SWORD will accept.

5.4.7. OpenSearch Support

OpenSearch is a small set of conventions and documents for describing and using "search engines", meaning any service that returns a set of results for a query. See extensive description in the *Business Layer section* of the documentation.

Please note that for result data formatting, OpenSearch uses Syndication Feed Settings (RSS). So, even if Syndication Feeds **are not** enable, they **must** be configured to enable OpenSearch. OpenSearch uses all the configuration properties for DSpace RSS to determine the mapping of metadata fields to feed fields. Note that a new field for authors has been added (used in Atom format only).

Property:	<i>websvc.opensearch.enable</i>
Example Value:	<i>websvc.opensearch.enable = false</i>
Informational Note:	Whether or not OpenSearch is enabled. By default, the feature is disabled. Change the property key to 'ture' to enable.
Property:	<i>websvc.opensearch.uicontext</i>
Example Value:	<i>websvc.opensearch.uicontext = simple-search</i>
Informational Note:	<i>Context for HTML request URLs. Change only for non-standard servlet mapping.</i>
Property:	<i>websvc.opensearch.svccontext</i>
Example Value:	<i>websvc.opensearch.svccontext = open-search/</i>
Informational Note:	Context for RSS/Atom request URLs. Change only for non-standard servlet mapping.
Property:	<i>websvc.opensearch.autolink</i>
Example Value:	<i>websvc.opensearch.autolink = true</i>
Informational Note:	<i>Present autodiscovery link in every page head.</i>
Property:	<i>websvc.opensearch.validity</i>
Example Value:	<i>websvc.opensearch.validity = 48</i>
Informational Note:	Number of hours to retain results before recalculating. This applies to the Manakin interface only.
Property:	<i>websvc.opensearch.shortname</i>



Example Value:	<i>websvc.opensearch.shortname = DSpace</i>
Informational Note:	A short name used in browsers for search service. It should be sixteen (16) or fewer characters.
Property:	<i>websvc.opensearch.longname</i>
Example Value:	<i>websvc.opensearch.longname = \${dspace.name}</i>
Informational Note:	A longer name up to 48 characters.
Property:	<i>websvc.opensearch.description</i>
Example Value:	<i>websvc.opensearch.description = \${dspace.name} DSpace repository</i>
Informational Note:	<i>Brief service description</i>
Property:	<i>websvc.opensearch.faviconurl</i>
Example Value:	<i>_websvc.opensearch.faviconurl = http://www.dspace.org/images/favicon.ico</i>
Informational Note:	Location of favicon for service, if any. They must be 16 x 16 pixels. You can provide your own local favicon instead of the default.
Property:	<i>websvc.opensearch.samplequery</i>
Example Value:	<i>websvc.opensearch.samplequery = photosynthesis</i>
Informational Note:	Sample query. This should return results. You can replace the sample query with search terms that should actually yield results in your repository.
Property:	<i>websvc.opensearch.tags</i>
Example Value:	<i>websvc.opensearch.tags = IR DSpace</i>
Informational Note:	Tags used to describe search service.
Property:	<i>websvc.opensearch.formats</i>
Example Value:	<i>websvc.opensearch.formats = html,atom,rss</i>
Informational Note:	Result formats offered. Use one or more comma-separated from the list: html, atom, rss. Please note that html is required for autodiscovery in browsers to function, and must be the first in the list if present.

5.4.8. Embargo

It is possible now to configure a DSpace instance to have an "Embargo" feature used for thesis and dissertations.

Property:	<i>embargo.field.terms</i>
Example Value:	<i>embargo.field.terms = SCHEMA.ELEMENT.QUALIFIER</i>
Informational Note:	DC metadata field to hold the user-supplied embargo terms



Property:	<i>embargo.field.lift</i>
Example Value:	<i>embargo.field.lift</i> = <i>SCHEMA.ELEMENT.QUALIFIER</i>
Informational Note:	DC metadata field to hold computed "lift date" of embargo
Property:	<i>embargo.terms.open</i>
Example Value:	<i>embargo.terms.open</i> = <i>forever</i>
Informational Note:	The string in terms field to indicate indefinite embargo
Property:	<i>plugin.single.org.dspace.embargo.EmbargoSetter</i>
Example Value:	<i>plugin.single.org.dspace.embargo.EmbargoSetter</i> = <i>CLASSNAME</i>
Informational Note:	Implementation of embargo setter plugin
Property:	<i>plugin.single.org.dspace.embargo.EmbargoLifter</i>
Example Value:	<i>plugin.single.org.dspace.embargo.EmbargoLifter</i> = <i>org.dspace.embargo.DefaultEm</i>
Informational Note:	Implementation of embargo lifter plugin

Remember that you need to replace *SCHEMA.ELEMENT.QUALIFIER* with a real metadata field. Additionally, you need to replace the *CLASSNAME* with a properly implemented plugin.

5.5. DSpace Services Framework

5.5.1. Implementing Providers

TODO: Provide examples of Implementing and Configuring Services in Spring and Guice

TODO: Provide examples of Implementing and Registering EventListeners in Spring and Guice.

Configuring Event Listeners

Event Listeners can be created by overriding the the EventListener interface:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>

  <bean id="dspace" class="org.dspace.utils.DSpace" />

  <bean id="dspace.eventService"
    factory-bean="dspace"
    factory-method="getEventService" />

  <bean class="org.my.EventListener">
    <property name="eventService" >
      <ref bean="dspace.eventService" />
    </property>
  </bean>
</beans>
```

TODO: Provide examples of Implementing and Registering Configurations in Spring and Guice.



5.5.2. Architectural Overview

DSpace 2 Kernel

The DS2 ([DSpace 2.0](#)) kernel manages the start up and access to the [core services](#) in DS2. It is meant to allow for a simple way to control the core parts of DSpace and allow for flexible ways to startup the kernel. For example, the kernel can be run inside a single webapp along with a frontend piece (like JSPUI) or it can be started as part of the servlet container so that multiple webapps can use a single kernel (this increases speed and efficiency). The kernel is also designed to happily allow multiple kernels to run in a single servlet container using identifier keys.

Kernel registration

The kernel will automatically register itself as an MBean in when it starts up so that it can be managed. It allows startup and shutdown and provides direct access to the ServiceManager and the ConfigurationService. All the other core services can be retrieved from the ServiceManager by their APIs.

Kernel Startup and Access

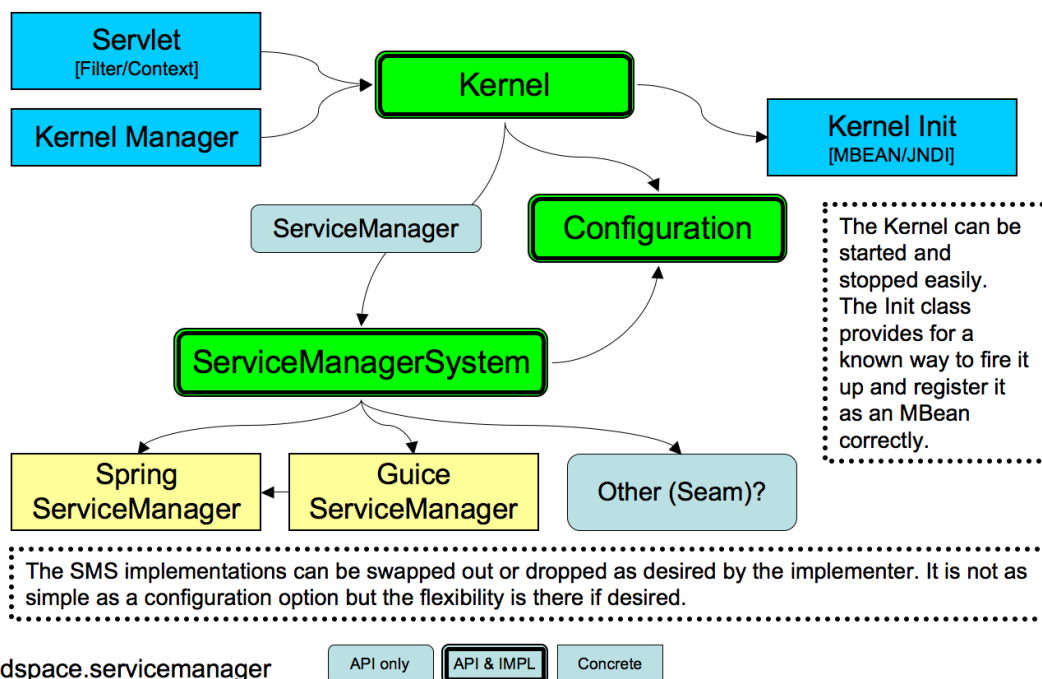
The kernel can be started and accessed through the use of Servlet Filter/ContextListeners which are provided as part of the DSpace 2 utilities. Developers don't need to understand what is going on behind the scenes and can simply write their applications and package them as webapps and take advantage of the services which are offered by DSpace 2. Access to the kernel is provided via the Kernel Manager and the DSpace object which will locate the kernel object and allow it to be used.

Service Manager

The ServiceManager abstracts the concepts of service lookups and lifecycle control. It also manages the configuration of services by allowing properties to be pushed into the services as they start up (mostly from the ConfigurationService). The ServiceManagerSystem abstraction allows the DSpace ServiceManager to use different systems to manage it's services. The current implementations include Spring and Guice. This allows DSpace 2 to have very little service management code but still be flexible and not tied to specific technology. Developers who are comfortable with those technologies can consume the services from a parent Spring ApplicationContext or a parent Guice Module. The abstraction also means that we can replace Spring/ Guice or add other dependency injection systems later without requiring developers to change their code. The interface provides simple methods for looking up services by interface type for developers who do not want to have to use or learn a dependency injection system or are using one which is not currently supported.



DSpace 2 Service Manager (DS2 Kernel / Service Manager)



5.5.3.

The DS2 kernel is compact so it can be completely started up in a unit test (technically integration test) environment (this is who we test the kernel and core services currently). This allows developers to execute code against a fully functional kernel while developing and then deploy their code with high confidence.

5.5.4. Providers and Plugins

For developers (how we are trying to make your lives easier): The DS2 ServiceManager supports a plugin/provider system which is runtime hot-swappable. The implementor can register any service/provider bean or class with the DS2 kernel ServiceManager. The ServiceManager will manage the lifecycle of beans (if desired) and will instantiate and manage the lifecycle of any classes it is given. This can be done at any time and does not have to be done during Kernel startup. This allows providers to be swapped out at runtime without disrupting the service if desired. The goal of this system is to allow DS2 to be extended without requiring any changes to the core codebase or a rebuild of the code code.

Activators

Developers can use an activator to allow the system to startup their service or provider. It is a simple interface with 2 methods which are called to startup the provider(s) and later to shut them down. These simply allow a developer to run some arbitrary code in order to create and register services if desired. It is the method provided to add plugins directly to the system via configuration as the activators are just listed in the configuration file and the system starts them up in the order it finds them.

Provider Stacks

Utilities are provided to assist with stacking and ordering providers. The priority is handled via a priority number such that 1 is the highest priority and something like 10 would be lower. 0 indicates that priority



is not important for this service and can be used to ensure the provider is placed at or near the end without having to set some arbitrarily high number.

The DSpace Services Framework is a backporting of the DSpace 2.0 Development Groups work in creating a reasonable and abstractable "Core Services" layer for DSpace components to operate within. The Services Framework represents a "best practices" for new DSpace architecture and implementation of extensions to the DSpace application. DSpace Services are best described as a "Simple Registry" where plugins. The DS2 ([DSpace 2.0](#)) core services are the main services that make up a DS2 system. These includes services for things like user and permissions management and storage and caching. These services can be used by any developer writing DS2 plugins (e.g. statistics), providers (e.g. Authn), or user interfaces (e.g. JSPUI).

5.5.5. Core Services

The core services are all behind APIs so that they can be reimplemented without affecting developers who are using the services. Most of the services have plugin/provider points so that customizations can be added into the system without touching the core services code. For example, let's say a deployer has a specialized authentication system and wants to manage the authentication calls which come into the system. The implementor can simply implement an AuthenticationProvider and then register it with the DS2 kernel Service-Manager. This can be done at any time and does not have to be done during Kernel startup. This allows providers to be swapped out at runtime without disrupting the DS2 service if desired. It can also speed up development by allowing quick hot redeploys of code during development.

Caching Service

Provides for a centralized way to handle caching in the system and thus a single point for configuration and control over all caches in the system. Provider and plugin developers are strongly encouraged to use this rather than implementing their own caching. The caching service has the concept of scopes so even storing data in maps or lists is discouraged unless there are good reasons to do so.

Configuration Service

The ConfigurationService controls the external and internal configuration of DSpace 2. It reads in properties files when the kernel starts up and merges them with any dynamic configuration data which is available from the services. The service allows settings to be updated as the system is running and also provides listeners which allow services to know when their configuration settings have changed and take action if desired. It is the central point to access and manage all the configuration settings in DSpace 2.

Manages the configuration of the DSpace 2 system. Can be used to manage configuration for providers and plugins also.

EventService

Handles events and provides access to listeners for consumption of events.

RequestService

In DS2 a request is the concept of a request (HTTP) or an atomic transaction in the system. It is likely to be an HTTP request in many cases but it does not have to be. This service provides the core services with a way to manage atomic transactions so that when a request comes in which requires multiple things to happen they can either all succeed or all fail without each service attempting to manage this independently. In a nutshell this simply allows identification of the current request and the ability to discover if it succeeded or failed when it ends. Nothing in the system will enforce usage of the service but we encourage developers who are interacting with the system to make use of this service so they know if the request they are participating in with has succeeded or failed and take appropriate actions.



SessionService

In DS2 a session is like an HttpSession (and generally is actually one) so this service is here to allow developers to find information about the current session and to access information in it. The session identifies the current user (if authenticated) so it also serves as a way to track user sessions. Since we use HttpSession directly it is easy to mirror sessions across multiple servers in order to allow for no-interruption failover for users when servers go offline.

5.6. DSpace Statistics

DSpace uses the Apache Solr application underlying the statistics. There is no need to download any separate software. All the necessary software is included.

5.6.1. Usage Event Logging and Usage Statistics Gathering

The DSpace Statistics Implementation is a Client/Server architecture based on Solr for collecting usage events in the JSPUI and XMLUI user interface applications of DSpace. Solr runs as a separate webapplication and an instance of Apache Http Client is utilized to allow parallel requests to log statistics events into this Solr instance. The Usage Event framework has a couple EventListeners installed which assist in

5.6.2. Configuration settings for Statistics

In the dspace.cfg file review the following fields to make sure they are uncommented:

Property Name	Default Value	Type	Description
solr.log.server	\${dspace.baseUrl}/solr/statistics	String	Is used by the SolrLogger Client class to connect to the Solr server over http and perform updates and queries. Access to this
solr.spidersfile	\${dspace.dir}/config/spiders.txt	String	Spiders file is utilized by the SolrLogger, this will be populated by running the following command: <pre>dsrun org.dspace.statistics.util.SpiderDet -i <httpd log file></pre>
solr.dbfile	\${dspace.dir}/config/GeoLiteCity.dat	String	The following refers to the GeoLiteCity database file utilized by the LocationUtils to calculate the location of client requests based on IP address. During the Ant build process (both fresh_install and update) this file will be downloaded from http://www.maxmind.com/app/geolitecity if a new version has been published or it is absent from your [dspace]/config directory.
useProxies	true	boolean	Will cause Statistics logging to look for X-



Property Name	Default Value	Type	Description
			Forward URI to detect clients IP that have accessed it through a Proxy service. Allows detection of client IP when accessing DSpace.
statistics.item.authorization	true	boolean	Enables access control restriction on DSpace Statistics pages, Restrictions are based on access rights to Community, Collection and Item Pages. This will require the user to sign on to see that statistics. Setting the statistics to "false" will make them publicly available.

Upgrade Process for Statistics.

Example of rebuild and redeploy DSpace (only if you have configured your distribution in this manner)

First approach the traditional DSpace build process for updating

```
cd [dspace-source]/dspace
mvn package
cd [dspace-source]/dspace/target/dspace-<version>-build.dir
ant -Dconfig=[dspace]/config/dspace.cfg update
cp -R [dspace]/webapps/* [TOMCAT]/webapps
```

The last step is only used if you are not mounting `[~mdiggory:dspace]/webapps` directly into your Tomcat, Resin or Jetty host (the recommended practice) If you only need to build the statistics, and don't make any changes to other web applications, you can replace the copy step above with:

```
cp -R dspace/webapps/solr TOMCAT/webapps
```

Again, only if you are not mounting `[~mdiggory:dspace]/webapps` directly into your Tomcat, Resin or Jetty host (the recommended practice)

Restart your webapps (Tomcat/Jetty/Resin)

5.6.3. Older setting that are no currently utilized in the reports

Are the following Dspace.cfg fields still used by the new 1.6 Statistics? If not, we need to either document this well or remove them altogether:

```
##### Statistical Report Configuration Settings #####

# should the stats be publicly available? should be set to false if you only
# want administrators to access the stats, or you do not intend to generate
# any
report.public = false

# directory where live reports are stored
report.dir = ${dspace.dir}/reports/
```

These fields are not used by the new 1.6 Statistics, but are only related to the Statistics from previous DSpace releases



5.7. JSPUI Configuration and Customization

5.7.1. Configuration

The user will need to refer to the extensive WebUI/JSPUI configurations that are contained in 5.2.36 JSP Web Interface Settings.

5.7.2. Customizing the JSP pages

The JSPUI interface is implemented using Java Servlets which handle the business logic, and JavaServer Pages (JSPs) which produce the HTML pages sent to an end-user. Since the JSPs are much closer to HTML than Java code, altering the look and feel of DSpace is relatively easy.

To make it even easier, DSpace allows you to 'override' the JSPs included in the source distribution with modified versions, that are stored in a separate place, so when it comes to updating your site with a new DSpace release, your modified versions will not be overwritten. It should be possible to dramatically change the look of DSpace to suit your organization by just changing the CSS style file and the site 'skin' or 'layout' JSPs in *jsp/layout*; if possible, it is recommended you limit local customizations to these files to make future upgrades easier.

You can also easily edit the text that appears on each JSP page by editing the *Messages.properties* file. However, note that unless you change the entry in all of the different language message files, users of other languages will still see the default text for their language. See Internationalization in Application Layer.

Note that the data (attributes) passed from an underlying Servlet to the JSP may change between versions, so you may have to modify your customized JSP to deal with the new data.

Thus, if possible, it is recommended you limit your changes to the 'layout' JSPs and the stylesheet.

The JSPs are available in one of two places:

- *[dspace-source]/dspace-jspui/dspace-jspui-webapp/src/main/webapp/* - Only exists if you downloaded the full Source Release of DSpace
- *[dspace-source]/dspace/target/dspace-[version].dir/webapps/dspace-jspui-webapp/* - The location where they are copied after first building DSpace.

If you wish to modify a particular JSP, place your edited version in the **[dspace-source]/dspace/modules/jspui/src/main/webapp/** directory (*this is the replacement for the pre-1.5 `_jsp/local` directory*), with the same path as the original. If they exist, these will be used in preference to the default JSPs. For example:

DSpace default	Locally-modified version
<i>[jsp.dir]/community-list.jsp</i>	<i>[jsp.custom-dir]/dspace/modules/jspui/src/main/webapp/community-list.jsp</i>
<i>[jsp.dir]/myspace/main.jsp</i>	<i>[jsp.custom-dir]/dspace/modules/jspui/src/main/webapp/myspace/main.jsp</i>

Heavy use is made of a style sheet, *styles.css.jsp*. If you make edits, copy the local version to *[jsp.custom-dir]/dspace/modules/jspui/src/main/webapp/styles.css.jsp*, and it will be used automatically in preference to the default, as described above.

Fonts and colors can be easily changed using the stylesheet. The stylesheet is a JSP so that the user's browser version can be detected and the stylesheet tweaked accordingly.

The 'layout' of each page, that is, the top and bottom banners and the navigation bar, are determined by the JSPs */layout/header-**jspand**/layout/footer-.jsp*. You can provide modified versions of these (in *[jsp.custom-dir]/dspace/modules/jspui/src/main/webapp/layout*), or define more styles and apply them to pages by using the "style" attribute of the *dspace:layout* tag.



1. Rebuild the DSpace installation package by running the following command from your `[dspace-source]/dspace/` directory:

```
mvn package
```

2. Update all DSpace webapps to `[dspace]/webapps` by running the following command from your `[dspace-source]/dspace/target/dspace-[version]-build.dir` directory:

```
ant -Dconfig=[dspace]/config/dspace.cfg update
```

3. Deploy the the new webapps:

```
cp -R /[dspace]/webapps/* /[tomcat]/webapps
```

4. Restart Tomcat

When you restart the web server you should see your customized JSPs.

5.8. XMLUI Configuration and Customization

5.8.1. Manakin Configuration Property Keys

In an effort to save the programmer/administrator some time, the configuration table below is taken from 5.3.43. *XMLUI Specific Configuration*.

Property:	<code>xmlui.supportedLocales</code>
Example Value:	<code>xmlui.supportedLocales = en, de</code>
Informational Note:	A list of supported locales for Manakin. Manakin will look at a user's browser configuration for the first language that appears in this list to make available to in the interface. This parameter is a comma separated list of Locales. All types of Locales country, country_language, country_language_variant. Note that if the appropriate files are not present (i.e. Messages_XX_XX.xml) then Manakin will fall back through to a more general language.
Property:	<code>xmlui.force.ssl</code>
Example Value:	<code>xmlui.force.ssl = true</code>
Informational Note:	Force all authenticated connections to use SSL, only non-authenticated connections are allowed over plain http. If set to true, then you need to ensure that the ' <code>dspace.hostname</code> ' parameter is set to the correctly.
Property:	<code>xmlui.user.registration</code>
Example Value:	<code>xmlui.user.registration = true</code>
Informational Note:	Determine if new users should be allowed to register. This parameter is useful in conjunction with Shibboleth where you want to disallow registration because Shibboleth will automatically register the user. Default value is true.



Property:	<i>xmlui.user.editmetadata</i>
Example Value:	<i>xmlui.user.editmetadata = true</i>
Informational Note:	Determines if users should be able to edit their own metadata. This parameter is useful in conjunction with Shibboleth where you want to disable the user's ability to edit their metadata because it came from Shibboleth. Default value is true.
Property:	<i>xmlui.user.assumelogon</i>
Example Value:	<i>xmlui.user.assumelogon = true</i>
Informational Note:	Determine if super administrators (those whom are in the Administrators group) can login as another user from the "edit eperson" page. This is useful for debugging problems in a running dspace instance, especially in the workflow process. The default value is false, i.e., no one may assume the login of another user.
Property:	<i>xmlui.user.loginredirect</i>
Example Value:	<i>xmlui.user.loginredirect = /profile</i>
Informational Note:	After a user has logged into the system, which url should they be directed? Leave this parameter blank or undefined to direct users to the homepage, or <i>/profile</i> for the user's profile, or another reasonable choice is <i>/submissions</i> to see if the user has any tasks awaiting their attention. The default is the repository home page.
Property:	<i>xmlui.theme.allowoverrides</i>
Example Value:	<i>xmlui.theme.allowoverrides = false</i>
Informational Note:	Allow the user to override which theme is used to display a particular page. When submitting a request add the HTTP parameter "themepath" which corresponds to a particular theme, that specified theme will be used instead of the any other configured theme. Note that this is a potential security hole allowing execution of unintended code on the server, this option is only for development and debugging it should be turned off for any production repository. The default value unless otherwise specified is "false".
Property:	<i>xmlui.bundle.upload</i>
Example Value:	<i>xmlui.bundle.upload = ORIGINAL, METADATA, THUMBNAIL, LICENSE, CC_LICENSE</i>
Informational Note:	Determine which bundles administrators and collection administrators may upload into an existing item through the administrative interface. If the user does not have the appropriate privileges (add and write) on the bundle then that bundle will not be shown to the user as an option.



Property:	<i>xmlui.community-list.render.full</i>
Example Value:	<i>xmlui.community-list.render.full = true</i>
Informational Note:	On the community-list page should all the metadata about a community/collection be available to the theme. This parameter defaults to true, but if you are experiencing performance problems on the community-list page you should experiment with turning this option off.
Property:	<i>xmlui.community-list.cache</i>
Example Value:	<i>xmlui.community-list.cache = 12 hours</i>
Informational Note:	Normally, Manakin will fully verify any cache pages before using a cache copy. This means that when the community-list page is viewed the database is queried for each community/collection to see if their metadata has been modified. This can be expensive for repositories with a large community tree. To help solve this problem you can set the cache to be assumed valued for a specific set of time. The downside of this is that new or editing communities/collections may not show up the website for a period of time.
Property:	<i>xmlui.bitstream.mods</i>
Example Value:	<i>xmlui.bitstream.mods = true</i>
Informational Note:	Optionally, you may configure Manakin to take advantage of metadata stored as a bitstream. The MODS metadata file must be inside the "METADATA" bundle and named MODS.xml. If this option is set to 'true' and the bitstream is present then it is made available to the theme for display.
Property:	<i>xmlui.bitstream.mets</i>
Example Value:	<i>xmlui.bitstream.mets = true</i>
Informational Note:	Optionally, you may configure Manakin to take advantage of metadata stored as a bitstream. The METS metadata file must be inside the "METADATA" bundle and named METS.xml. If this option is set to "true" and the bitstream is present then it is made available to the theme for display.
Property:	<i>xmlui.google.analytics.key</i>
Example Value:	<i>xmlui.google.analytics.key = UA-XXXXXX-X</i>
Informational Note:	If you would like to use google analytics to track general website statistics then use the following parameter to provide your analytics key. First sign up for an account at http://analytics.google.com , then create an entry for your repositories website. Google Analytics will give you a snippet of javascript code to place on your site, inside that snippet it is your google



	analytics key usually found in the line: <code>_uacct = "UA-XXXXXXX-X"</code> Take this key (just the UA-XXXXXXX-X part) and place it here in this parameter.
Property:	<code>xmlui.controlpanel.activity.max</code>
Example Value:	<code>xmlui.controlpanel.activity.max = 250</code>
Informational Note:	Assign how many page views will be recorded and displayed in the control panel's activity viewer. The activity tab allows an administrator to debug problems in a running DSpace by understanding who and how their dspace is currently being used. The default value is 250.
Property:	<code>xmlui.controlpanel.activity.ipheader</code>
Example Value:	<code>xmlui.controlpanel.activity.ipheader = X-Forward-For</code>
Informational Note:	Determine where the control panel's activity viewer receives an events IP address from. If your DSpace is in a load balanced environment or otherwise behind a context-switch then you will need to set the parameter to the HTTP parameter that records the original IP address.

5.8.2. Configuring Themes and Aspects

The Manakin user interface is composed of two distinct components: *aspects* and *themes*. Manakin aspects are like extensions or plugins for Manakin; they are interactive components that modify existing features or provide new features for the digital repository. Manakin themes stylize the look-and-feel of the repository, community, or collection.

The repository administrator is able to define which aspects and themes are installed for the particular repository by editing the `[dspace]/config/xmlui.xconf` configuration file. The `xmlui.xconf` file consists of two major sections: Aspects and Themes.

Aspects

The `<aspects>` section defines the "Aspect Chain", or the linear set of aspects that are installed in the repository. For each aspect that is installed in the repository, the aspect makes available new features to the interface. For example, if the "submission" aspect were to be commented out or removed from the `xmlui.xconf`, then users would not be able to submit new items into the repository (even the links and language prompting users to submit items are removed). Each `<aspect>` element has two attributes, *name* and *path*. The name is used to identify the Aspect, while the path determines the directory where the aspect's code is located. Here is the default aspect configuration:

```
<aspects>
  <aspect name="Artifact Browser" path="resource://aspects/ArtifactBrowser/" />
  <aspect name="Administration" path="resource://aspects/Administrative/" />
  <aspect name="E-Person" path="resource://aspects/EPerson/" />
  <aspect name="Submission and Workflow" path="resource://aspects/Submission/" />
</aspects>
```

A standard distribution of Manakin/DSpace includes four "core" aspects:

- ***Artifact Browser***The Artifact Browser Aspect is responsible for browsing communities, collections, items and bitstreams, viewing an individual item and searching the repository.



- ***E-Person***The E-Person Aspect is responsible for logging in, logging out, registering new users, dealing with forgotten passwords, editing profiles and changing passwords.
- ***Submission***The Submission Aspect is responsible for submitting new items to DSpace, determining the workflow process and ingesting the new items into the DSpace repository.
- ***Administrative***The Administrative Aspect is responsible for administrating DSpace, such as creating, modifying and removing all communities, collections, e-persons, groups, registries and authorizations.

Themes

The `<themes>` section defines a set of "rules" that determine where themes are installed in the repository. Each rule is processed in the order that it appears, and the first rule that matches determines the theme that is applied (so order is important). Each rule consists of a `<theme>` element with several possible attributes:

- **name** (*always required*)The name attribute is used to document the theme's name.
- **path** (*always required*)The path attribute determines where the theme is located relative to the `themes/` directory and must either contain a trailing slash or point directly to the theme's `sitemap.xml` file.
- **regex** (*either regex and/or handle is required*)The regex attribute determines which URLs the theme should apply to.
- **handle** (*either regex and/or handle is required*)The handle attribute determines which community, collection, or item the theme should apply to.

If you use the "handle" attribute, the effect is cascading, meaning if a rule is established for a community then all collections and items within that community will also have this theme apply to them as well. Here is an example configuration:

```
<themes>
  <theme name="Theme 1" handle="123456789/23" path="theme1/" />
  <theme name="Theme 2" regex="community-list" path="theme2/" />
  <theme name="Reference Theme" regex=".*" path="Reference/" />
</themes>
```

In the example above three themes are configured: "Theme 1", "Theme 2", and the "Reference Theme". The first rule specifies that "Theme 1" will apply to all communities, collections, or items that are contained under the parent community "123456789/23". The next rule specifies any URL containing the string "community-list" will get "Theme 2". The final rule, using the regular expression ".*", **will match anything**, so all pages which have not matched one of the preceding rules will be matched to the Reference Theme.

5.8.3. Multilingual Support

The XMLUI user interface supports multiple languages through the use of internationalization catalogues as defined by the <http://cocoon.apache.org/2.1/userdocs/i18nTransformer.html>. Each catalog contains the translation of all user-displayed strings into a particular language or variant. Each catalog is a single xml file whose name is based upon the language it is designated for, thus:

`messages_language_country_variant.xml`

`messages_language_country.xml`

`messages_language.xml`

`messages.xml`

The interface will automatically determine which file to select based upon the user's browser and system configuration. For example, if the user's browser is set to Australian English then first the system will check if `messages_en_au.xml` is available. If this translation is not available it will fall back to `messages_en.xml`, and finally if that is not available, `messages.xml`.



Manakin supplies an English only translation of the interface. In order to add other translations to the system, locate the `[dspace-source]/dspace/modules/xmlui/src/main/webapp/i18n/` directory. By default this directory will be empty; to add additional translations add alternative versions of the `messages.xml` file in specific language and country variants as needed for your installation.

To set a language other than English as the default language for the repository's interface, simply name the translation catalogue for the new default language "`messages.xml`"

5.8.4. Creating a New Theme

Manakin themes stylize the look-and-feel of the repository, community, or collection and are distributed as self-contained packages. A Manakin/DSpace installation may have multiple themes installed and available to be used in different parts of the repository. The central component of a theme is the `sitemap.xmap`, which defines what resources are available to the theme such as XSL stylesheets, CSS stylesheets, images, or multimedia files.

1) Create theme skeleton

Most theme developers do not create a new theme from scratch; instead they start from the standard theme template, which defines a skeleton structure for a theme. The template is located at: `[dspace-source]/dspace-xmlui/dspace-xmlui-webbapp/src/main/webbapp/themes/template`. To start your new theme simply copy the theme template into your locally defined modules directory, `[dspace-source]/dspace/modules/xmlui/src/main/webbapp/themes/[your theme's directory]/`.

2) Modify theme variables

The next step is to modify the theme's parameters so that the theme knows where it is located. Open the `[your theme's directory]/sitemap.xmap` and look for `<global-variables>`

```
<global-variables>
  <theme-path>[your theme's directory]</theme-path>
  <theme-name>[your theme's name]</theme-name>
</global-variables>
```

Update both the theme's path to the directory name you created in step one. The theme's name is used only for documentation.

3) Add your CSS stylesheets

The base theme template will produce a repository interface without any style - just plain XHTML with no color or formatting. To make your theme useful you will need to supply a CSS Stylesheet that creates your desired look-and-feel. Add your new CSS stylesheets:

`[your theme's directory]/lib/style.css` (The base style sheet used for all browsers)

`[your theme's directory]/lib/style-ie.css` (Specific stylesheet used for internet explorer)

4) Install theme and rebuild DSpace

Next rebuild and deploy DSpace (replace `<version>` with the your current release):

1. Rebuild the DSpace installation package by running the following command from your `[dspace-source]/dspace/` directory:

```
mvn package
```

2. Update all DSpace webapps to `[dspace]/webapps` by running the following command from your `[dspace-source]/dspace/target/dspace-[version]-build.dir` directory:

```
ant -Dconfig=[dspace]/config/dspace.cfg update
```

3. Deploy the the new webapps:

```
cp -R /[dspace]/webapps/* /[tomcat]/webapps
```

4. Restart Tomcat



This will ensure the theme has been installed as described in the previous section "Configuring Themes and Aspects".

5.8.5. Adding Static Content

The XMLUI user interface supports the addition of globally static content (as well as static content within individual themes).

Globally static content can be placed in the `[dspace-source]/dspace/modules/xmlui/src/main/webapp/static/` directory. By default this directory only contains the default `robots.txt` file, which provides helpful site information to web spiders/crawlers. However, you may also add static HTML (`*.html`) content to this directory, as needed for your installation.

Any static HTML content you add to this directory may also reference static content (e.g. CSS, Javascript, Images, etc.) from the same `[dspace-source]/dspace/modules/xmlui/src/main/webapp/static/` directory. You may reference other static content from your static HTML files similar to the following:

```
<link href="./static/mystyle.css" rel="stylesheet" type="text/css"/>


```

6. System Administration

6.1. 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:	<code>[dspace]/bin/dspace structure-builder</code>
Java class:	<code>org.dspace.administer.StructBuilder</code>
Argument: short and long (if available) forms:	Description of the argument
<code>-f</code>	Source xml file.
<code>-o</code>	Output xml file.
<code>-e</code>	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.

6.1.1. Limitation

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

6.2. 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 Item, or *disseminate* an Item as a package.

To see all the options, invoke it as:

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

This mode also displays a list of the names of package ingesters and disseminators that are available.

6.2.1. Ingesting

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



```
[dspace]/bin/packager -e user -c handle -t packagerpath
```

Where *_user_* is the e-mail address of the E-Person under whose authority this runs; *_handle_* is the Handle of the collection into which the Item is added, *_packager_* is the plugin name of the package ingester to use, and *_path_* 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/packager -e florey@mit.edu -c 1721.2/13 -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/packager -e florey@mit.edu -c 1721.2/13 -t pdf -
```

6.2.2. Disseminating

To disseminate an Item as a package, give the command:

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

Where *_user_* is the e-mail address of the E-Person under whose authority this runs; *_handle_* is the Handle of the Item to disseminate; *_packager_* is the plugin name of the package disseminator to use; and *_path_* is the path to the file to create (or "-" to write to the standard output). This example writes an Item out as a METS package in the file "454.zip":

```
/dspace/bin/packager -e florey@mit.edu -d -i 1721.2/454 -t METS 454.zip
```

6.2.3. 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 <http://www.loc.gov/standards/mets/>, <http://www.loc.gov/standards/mods/>, and <http://www.loc.gov/standards/premis/>. The plugin name is *METS* by default, and it uses MODS for descriptive metadata.

The DSpace METS SIP profile is available at: <http://www.dspace.org/standards/METS/SIP/profilev1p0/metsipv1p0.pdf>.

6.3. 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.

6.3.1. 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 its 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 the sequence:

```
\tbundle:bundlename
```

where `\t` is the tab character and `'bundlename'` is replaced by the name of the bundle to which the bitstream should be added. If no bundle is specified, the bitstream will be added to the 'ORIGINAL' bundle.

6.3.2. 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:	<code>[_dspace]_bin/dspace import</code>
Java class:	<code>org.dspace.app.itemimport.ItemImport</code>
Arguments short and (long) forms:	Description
<code>-a</code> or <code>--add</code>	Add items to DSpace †
<code>-r</code> or <code>--replace</code>	Replace items listed in mapfile †



<i>d-or-delete</i>	Delete items listed in mapfile †
<i>s-or-source</i>	Source of the items (directory)
<i>c-or-collection</i>	Destination Collection by their Handle or database ID
<i>m-or-mapfile</i>	Where the mapfile for items can be found (name and directory)
<i>e-or-eperson</i>	Email of eperson doing the importing
<i>w-or-workflow</i>	Send submission through collection' workflow
<i>n-or-notify</i>	Kicks off the email alerting of the item(s) has(have) been imported
<i>t-or-test</i>	Test run—do not actually import items
<i>p-or-template</i>	Apply the collection template
<i>R-or-resume</i>	Resume a failed import (Used on Add only)
<i>h-or-help</i>	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/import --add --eperson=joe@user.com --collection=CollectionID --source=items_dir --mapfile=mapfile
```

or by using the short form:

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

Long form:



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

In long form:

```
[dspace]/bin/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.

6.3.3. 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:	<code>_[dspace]_/bin/dspace export</code>
Java class:	<code>org.dspace.app.itemexport.ItemExport</code>
Arguments short and (long) forms:	Description
<code>t-or-type</code>	Type of export. <i>COLLECTION</i> will inform the program you want the whole collection. <i>ITEM</i> will be only the specific item. (You will actually key in the keywords in all caps. See examples below.)
<code>i-or-id</code>	The ID or Handle of the Collection or Item to export.
<code>d-or-dest</code>	The destination of where you want the file of items to be placed. You place the path if necessary.
<code>n-or-number</code>	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.
<code>m-or-migrate</code>	Export the item/collection for migration. This will remove the handle and metadata that will be re-created in the new instance of DSpace.
<code>h-or-help</code>	Brief Help.

Table 4. 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_unumber
```

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.

6.4. 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_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.

6.5. 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.

6.5.1. 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.

6.5.2. ItemUpdate Commands

Command used:	<code>[_dspace]_bin/dspace itemupdate</code>
Java class:	<code>org.dspace.app.itemimport.ItemUpdate</code>
Arguments short and (long) forms:	Description
<code>a-or--addmetadata [metadata element]</code>	Repeatable for multiple elements. The metadata element should be in the form <code>dc.x</code> or <code>dc.x.y</code> . The mandatory argument indicates the metadata fields in the <code>dublin_core.xml</code> file to be added unless already present. However, duplicate fields will not be added to the item metadata without warning or error.
<code>d-or--deletemetadata [metadata element]</code>	Repeatable for multiple elements. All metadata fields matching the element will be deleted.
<code>A-or--addbitstream</code>	Adds bitstreams listed in the contents file with the bitstream metadata cited there.
<code>D-or--deletebitstream [filter plug classname or alis]</code>	Not repeatable. With no argument, this operation deletes bitstreams listed in the <code>deletes_contents</code> file. Only bitstream ids are recognized identifiers for this operation. The optional filter argument is the classname of an implementation of



	<i>org.dspace.app.itemupdate.BitstreamFilter</i> 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 <i>delete_contents</i> file is not required for any item. The filter properties file will contains properties pertinent to the particular filter used. Multiple filters are not allowed.
<i>h-or-help</i>	Displays brief command line help.
<i>e-or-eperson</i>	Email address of the person or the user's database ID (Required)
<i>s-or-source</i>	Directory archive to process (Required)
<i>i-or-itemidentifier</i>	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 <i>dc.identifier.uri</i> field. (Optional)
<i>t-or-test</i>	Runs the process in test mode with logging but no changes applied to the DSpace instance. (Optional)
<i>P-or-alterprovenance</i>	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, un keepin with the practice of MediaFilterManager. (Optional)
<i>F-or-filterproperties</i>	The filter properties files to be used by the delete bitstreams action (Optional)

. CLI Examples

Adding Metadata:

```
[dspace]/bin/dspace updateitem -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).

6.6. 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.

6.6.1. 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.

6.6.2. 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 it 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*.



6.6.3. 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 registered 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.

6.6.4. 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 be registered in DSpace instead of being uploaded and ingested normally.

6.6.5. 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.

6.6.6. 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 its 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.

6.7. METS Tools

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

6.7.1. The Export Tool

This tool is obsolete, and does not export a complete AIP. Its use is strongly deprecated.

Command used:	<code>_[dspace]_/bin/dspace mets-export</code>
Java class:	<code>org.dspace.app.mets.METSExport</code>



Arguments short and (long) forms:	Description
<i>a-or-all</i>	Export all items in the archive.
<i>c-or-collection</i>	Handle of the collection to export.
<i>d-or-destination</i>	Destination directory.
<i>i-or-item</i>	Handle of the item to export.
<i>h-or-help</i>	Help

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

Exporting an individual item. From the CLI:

```
[dspace]/bin/dspace mets-export -i [handle] -d /path/to/destination
```

Exporting a collection. From the CLI:

```
[dspace]/bin/dspace mets-export -c [handle] -d /path/to/destination
```

Exporting all the items in DSpace. From the CLI:

```
[dspace]/bin/dspace mets-export -a -d /path/to/destination
```

6.7.2. The AIP Format

Note that this tool is deprecated, and the output format is not a true AIP

Each exported item is written to a separate directory, created under the base directory specified in the command-line arguments, or in the current directory if *--destination* is omitted. The name of each directory is the Handle, URL-encoded so that the directory name is 'legal'.

Within each item directory is a *mets.xml* file which contains the METS-encoded metadata for the item. Bitstreams in the item are also stored in the directory. Their filenames are their MD5 checksums, firstly for easy integrity checking, and also to avoid any problems with 'special characters' in the filenames that were legal on the original filing system they came from but are illegal in the server filing system. The *mets.xml* file includes XLink pointers to these bitstream files.

An example AIP might look like this:

- *hdl%3A123456789%2F8/*

- *mets.xml* – METS metadata

- *184BE84F293342* – bitstream

- *3F9AD0389CB821*

- *135FB82113C32D*

The contents of the METS in the *mets.xml* file are as follows:

- A *dmdSec* (descriptive metadata section) containing the item's metadata in <http://www.loc.gov/standards/mods/XML>. The Dublin Core descriptive metadata is mapped to MODS since there is no official qualified Dublin Core XML schema in existence as of yet, and the Library Application Profile of DC that DSpace uses includes some qualifiers that are not part of the <http://dublincore.org/documents/dcmi-terms/>.
- An *amdSec* (administrative metadata section), which contains the a rights metadata element, which in turn contains the base64-encoded deposit license (the license the submitter granted as part of the submission process).
- A *fileSec* containing a list of the bitstreams in the item. Each bundle constitutes a *fileGrp*. Each bitstream is represented by a *file* element, which contains an *FLocat* element with a simple XLink to the bitstream in



the same directory as the *mets.xml* file. The *file* attributes consist of most of the basic technical metadata for the bitstream. Additionally, for those bitstreams that are thumbnails or text extracted from another bitstream in the item, those 'derived' bitstreams have the same *GROUPID* as the bitstream they were derived from, in order that clients understand that there is a relationship. The *OWNERID* of each *file* is the 'persistent' bitstream identifier assigned by the DSpace instance. The *ID* and *GROUPID* attributes consist of the item's Handle, together with the bitstream's sequence ID, which underscores used in place of dots and slashes. For example, a bitstream with sequence ID 24, in the item *hdl:123.456/789* will have the *ID123_456_789_24*. This is because *ID* and *GROUPID* attributes must be of type *xsd:id*.

6.7.3. 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

6.8. 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 *FormatFilter* 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/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.

6.9. 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:	<code>_[dspace]_/bin/dspace community-filiator</code>
Java class:	<code>org.dspace.administer.CommunityFiliator</code>
Arguments short and (long) forms:	Description
<code>s-o-set</code>	Set a parent/child relationship
<code>r-o-remove</code>	Remove a parent/child relationship
<code>e-o-child</code>	Child community (Handle or database ID)
<code>p-o-parent</code>	Parent community (Handle or database ID)



h-or-help	Online help.
----------------------	--------------

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

```
dssrun org.dspace.administer.CommunityFiliator --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.

6.10. 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:	<i>_[dspace]_/bin/dspace metadata-export</i>
Java class:	org.dspace.app.bulkedit.MetadataExport



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

6.10.1. 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.csv' found in the '/batch_export' directory.

6.10.2. Import Function

The following table summarizes the basics.

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

6.10.3. 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 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 cvs 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).

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.

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

6.11.1. 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 Bistream 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 bistreams 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. The Check
- **Looping mode:** `[dSPACE]/bin/dSPACE checker -l` or `_[dSPACE]/bin/dSPACE checker -L_` There are two modes. The lowercase 'l' (-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!

6.11.2. 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>
```

6.11.3. 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]/conf/defaults/log4j.properties` file and run `[dspace]/bin/install_configs`.

6.11.4. 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/dsrun.bat org.dspace.app.checker.ChecksumChecker -d2h -p` (This command should appear on a single line).

6.11.5. 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:	<code>_[dspace]_/bin/dspace checker</code>
Java class:	<code>org.dspace.checker.DailyReportEmailer</code>
Arguments short and (long) forms):	Description
<code>a-or-All</code>	Send all the results (everything specified below)
<code>d-or-Deleted</code>	Send E-mail report for all bitstreams set as deleted for today.
<code>m-or-Missing</code>	Send E-mail report for all bitstreams not found in assetstore for today.
<code>c-or-Changed</code>	Send E-mail report for all bitstreams where checksum has been changed for today.
<code>u-or-Unchanged</code>	Send the Unchecked bitstream report.
<code>n-or-Not Processed</code>	Send E-mail report for all bitstreams set to longer be processed for today.
<code>h-or-help</code>	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:	<code>_[dspace]_/bin/dspace embargo-lifter</code>
Java class:	<code>org.dspace.embargo.EmbargoManager</code>
Arguments short and (long) forms):	Description
<code>c-or-check</code>	ONLY check the state of embargoed Items, do NOT lift any embargoes
<code>i-or-identifier</code>	Process ONLY this handle identifier(s), which must be an Item. Can be repeated.
<code>t-or-lift</code>	Only lift embargoes, do NOT check the state of any embargoed items.
<code>n-or-dryrun</code>	Do no change anything in the data model, print message instead.
<code>v-or-verbose</code>	Print a line describing the action taken for each embargoed item found.
<code>q-or-quiet</code>	No output except upon error.
<code>h-or-help</code>	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:	<code>_[dspace]_/bin/dspace index-init</code>
Java class:	<code>org.dspace.browse.IndexBrowse</code>
Arguments short and long forms):	Description
<code>r-or-rebuild</code>	Should we rebuild all the indexes, which removes old tables and creates new ones. For use with <code>-f</code> . Mutually exclusive with <code>-d</code>
<code>s-or-start</code>	<code>[-s <int>] _start from this index number and work upwards (mostly only useful for debugging). For use with <code>_-t</code> and <code>-f</code></code>
<code>x-or-execute</code>	Execute all the remove and create SQL against the database. For use with <code>-t</code> <code>_and</code> <code>_-f</code>
<code>i-or-index</code>	Actually do the indexing. Mutually exclusive with <code>-t</code> and <code>-f</code> .
<code>o-or-out</code>	<code>[-o<filename>]</code> write the remove and create SQL to the given file. For use with <code>-t</code> and <code>-f</code>
<code>p-or-print</code>	Write the remove and create SQL to the stdout. For use with <code>-t</code> and <code>-f</code> .
<code>t-or-tables</code>	Create the tables only, do no attempt to index. Mutually exclusive with <code>-f</code> and <code>-i</code>
<code>f-or-full</code>	Make the tables, and do the indexing. This forces <code>-x</code> . Mutually exclusive with <code>-f</code> and <code>-i</code> .



v or <i>-verbose</i>	Print extra information to the stdout. If used in conjunction with <i>-p</i> , you cannot use the stdout to generate your database structure.
d or <i>-delete</i>	Delete all the indexes, but do not create new ones. For use with <i>-f</i> . This is mutually exclusive with <i>-r</i> .
h or <i>-help</i>	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. Running this is the same as:

```
[dSPACE]/bin/dSRUN org.dSPACE.browse.IndexBrowse -f -r
```

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). Running this is the same as:

```
[dSPACE]/bin/dSRUN org.dSPACE.browse.IndexBrowse -i
```

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/dSRUN org.dSPACE.browse.IndexBrowse -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 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 thesauris.)*)
 - 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`



- PostgreSQL schemas are in `[dSPACE-source]/dSPACE/etc/postgres/`
- Oracle schemas are in `[dSPACE-source]/dSPACE/etc/oracle/`. The SQL (DDL) statements to create the tables for the current release, starting with an empty database, are in `database_schema.sql`. The schema SQL file also creates the two required e-person groups (*Anonymous* and *Administrator*) that are required for the system to function properly.

Also in `[dSPACE-source]/dSPACE/etc/[database]` are various SQL files called `database_schema_1x_1y`. These contain the necessary SQL commands to update a live DSpace database from version 1.x to 1.y. Note that this might not be the only part of an upgrade process: see [Updating a DSpace Installation](#) for details.

The DSpace database code uses an SQL function `getnextid` to assign primary keys to newly created rows. This SQL function must be safe to use if several JVMs are accessing the database at once; for example, the Web UI might be creating new rows in the database at the same time as the batch item importer. The PostgreSQL-specific implementation of the method uses *SEQUENCES* for each table in order to create new IDs. If an alternative database backend were to be used, the implementation of `getnextid` could be updated to operate with that specific DBMS.

The `etc` directory in the source distribution contains two further SQL files. `clean-database.sql` contains the SQL necessary to completely clean out the database, so use with caution! The Ant target `clean_database` can be used to execute this. `update-sequences.sql` contains SQL to reset the primary key generation sequences to appropriate values. You'd need to do this if, for example, you're restoring a backup database dump which creates rows with specific primary keys already defined. In such a case, the sequences would allocate primary keys that were already used.

Versions of the `.sql` files for Oracle are stored in `[dSPACE-source]/dSPACE/etc/oracle`. These need to be copied over their PostgreSQL counterparts in `[dSPACE-source]/dSPACE/etc` prior to installation.

7.1.1. Maintenance and Backup

When using PostgreSQL, it's a good idea to perform regular 'vacuuming' of the database to optimize performance. This is performed by the `vacuumdb` command which can be executed via a 'cron' job, for example by putting this in the system `crontab`:

```
# clean up the database nightly
40 2 * * * /usr/local/pgsql/bin/vacuumdb --analyze dSPACE > /dev/null
    2>&1
```

The DSpace database can be backed up and restored using usual methods, for example with `pg_dump` and `psql`. However when restoring a database, you will need to perform these additional steps:

- The `fresh_install` target loads up the initial contents of the Dublin Core type and bitstream format registries, as well as two entries in the `epersongroup` table for the system anonymous and administrator groups. Before you restore a raw backup of your database you will need to remove these, since they will already exist in your backup, possibly having been modified. For example, use:

```
DELETE FROM dctyperegistry;
DELETE FROM bitstreamformatregistry;
DELETE FROM epersongroup;
```

- After restoring a backup, you will need to reset the primary key generation sequences so that they do not produce already-used primary keys. Do this by executing the SQL in `[dSPACE-source]/dSPACE/etc/update-sequences.sql`, for example with:

```
psql -U dSPACE -f
    [dSPACE-source]/dSPACE/etc/update-sequences.sql
```

Future updates of DSpace may involve minor changes to the database schema. Specific instructions on how to update the schema whilst keeping live data will be included. The current schema also contains a



few currently unused database columns, to be used for extra functionality in future releases. These unused columns have been added in advance to minimize the effort required to upgrade.

7.1.2. Configuring the RDBMS Component

The database manager is configured with the following properties in *dspace.cfg*:

<i>db.url</i>	The JDBC URL to use for accessing the database. This should not point to a connection pool, since DSpace already implements a connection pool.
<i>db.driver</i>	JDBC driver class name. Since presently, DSpace uses PostgreSQL-specific features, this should be <i>org.postgresql.Driver</i> .
<i>db.username</i>	Username to use when accessing the database.
<i>db.password</i>	Corresponding password to use when accessing the database.

7.2. Bitstream Store

DSpace offers two means for storing content. The first is in the file system on the server. The second is using <http://www.sdsc.edu/srb>. Both are achieved using a simple, lightweight API.

SRB is purely an option but may be used in lieu of the server's file system or in addition to the file system. Without going into a full description, SRB is a very robust, sophisticated storage manager that offers essentially unlimited storage and straightforward means to replicate (in simple terms, backup) the content on other local or remote storage resources.

The terms "store", "retrieve", "in the system", "storage", and so forth, used below can refer to storage in the file system on the server ("traditional") or in SRB.

The *BitstreamStorageManager* provides low-level access to bitstreams stored in the system. In general, it should not be used directly; instead, use the *Bitstream* object in the content management API since that encapsulated authorization and other metadata to do with a bitstream that are not maintained by the *BitstreamStorageManager*.

The bitstream storage manager provides three methods that store, retrieve and delete bitstreams. Bitstreams are referred to by their 'ID'; that is the primary key *bitstream_id* column of the corresponding row in the database.

As of DSpace version 1.1, there can be multiple bitstream stores. Each of these bitstream stores can be traditional storage or SRB storage. This means that the potential storage of a DSpace system is not bound by the maximum size of a single disk or file system and also that traditional and SRB storage can be combined in one DSpace installation. Both traditional and SRB storage are specified by configuration parameters. Also see Configuring the Bitstream Store below.

Stores are numbered, starting with zero, then counting upwards. Each bitstream entry in the database has a store number, used to retrieve the bitstream when required.

At the moment, the store in which new bitstreams are placed is decided using a configuration parameter, and there is no provision for moving bitstreams between stores. Administrative tools for manipulating bitstreams and stores will be provided in future releases. Right now you can move a whole store (e.g. you could move store number 1 from */localdisk/store* to */fs/anotherdisk/store* but it would still have to be store number 1 and have the exact same contents.

Bitstreams also have an 38-digit internal ID, different from the primary key ID of the bitstream table row. This is not visible or used outside of the bitstream storage manager. It is used to determine the exact location (relative to the relevant store directory) that the bitstream is stored in traditional or SRB storage. The first



three pairs of digits are the directory path that the bitstream is stored under. The bitstream is stored in a file with the internal ID as the filename.

For example, a bitstream with the internal ID `12345678901234567890123456789012345678` is stored in the directory:

```
(assetstore dir)/12/34/56/12345678901234567890123456789012345678
```

The reasons for storing files this way are:

- Using a randomly-generated 38-digit number means that the 'number space' is less cluttered than simply using the primary keys, which are allocated sequentially and are thus close together. This means that the bitstreams in the store are distributed around the directory structure, improving access efficiency.
- The internal ID is used as the filename partly to avoid requiring an extra lookup of the filename of the bitstream, and partly because bitstreams may be received from a variety of operating systems. The original name of a bitstream may be an illegal UNIX filename.
When storing a bitstream, the *BitstreamStorageManager* DOES set the following fields in the corresponding database table row:

- *bitstream_id*
- *size*
- *checksum*
- *checksum_algorithm*
- *internal_id*
- *deleted*
- *store_number*The remaining fields are the responsibility of the *Bitstream* content management API class.

The bitstream storage manager is fully transaction-safe. In order to implement transaction-safety, the following algorithm is used to store bitstreams:

1. A database connection is created, separately from the currently active connection in the current DSpace context.
2. An unique internal identifier (separate from the database primary key) is generated.
3. The bitstream DB table row is created using this new connection, with the *deleted* column set to *true*.
4. The new connection is `_commit_ted`, so the 'deleted' bitstream row is written to the database
5. The bitstream itself is stored in a file in the configured 'asset store directory', with a directory path and filename derived from the internal ID
6. The *deleted* flag in the bitstream row is set to *false*. This will occur (or not) as part of the current DSpace *Context*.
This means that should anything go wrong before, during or after the bitstream storage, only one of the following can be true:
 - No bitstream table row was created, and no file was stored
 - A bitstream table row with *deleted=true* was created, no file was stored
 - A bitstream table row with *deleted=true* was created, and a file was storedNone of these affect the integrity of the data in the database or bitstream store.



Similarly, when a bitstream is deleted for some reason, its *deleted* flag is set to true as part of the overall transaction, and the corresponding file in storage is *not* deleted.

The above techniques mean that the bitstream storage manager is transaction-safe. Over time, the bitstream database table and file store may contain a number of 'deleted' bitstreams. The *cleanup* method of *BitstreamStorageManager* goes through these deleted rows, and actually deletes them along with any corresponding files left in the storage. It only removes 'deleted' bitstreams that are more than one hour old, just in case cleanup is happening in the middle of a storage operation.

This cleanup can be invoked from the command line via the *Cleanup* class, which can in turn be easily executed from a shell on the server machine using */dSPACE/bin/cleanup*. You might like to have this run regularly by *cron*, though since DSpace is read-lots, write-not-so-much it doesn't need to be run very often.

7.2.1. Backup

The bitstreams (files) in traditional storage may be backed up very easily by simply 'tarring' or 'zipping' the *assetstore* directory (or whichever directory is configured in *dSPACE.cfg*). Restoring is as simple as extracting the backed-up compressed file in the appropriate location.

Similar means could be used for SRB, but SRB offers many more options for managing backup.

It is important to note that since the bitstream storage manager holds the bitstreams in storage, and information about them in the database, that a database backup and a backup of the files in the bitstream store must be made at the same time; the bitstream data in the database must correspond to the stored files.

Of course, it isn't really ideal to 'freeze' the system while backing up to ensure that the database and files match up. Since DSpace uses the bitstream data in the database as the authoritative record, it's best to back up the database before the files. This is because it's better to have a bitstream in storage but not the database (effectively non-existent to DSpace) than a bitstream record in the database but not storage, since people would be able to find the bitstream but not actually get the contents.

7.2.2. Configuring the Bitstream Store

Both traditional and SRB bitstream stores are configured in *dSPACE.cfg*.

Configuring Traditional Storage

Bitstream stores in the file system on the server are configured like this:

```
assetstore.dir = [dSPACE]/assetstore
```

(Remember that *[dSPACE]* is a placeholder for the actual name of your DSpace install directory).

The above example specifies a single asset store.

```
assetstore.dir = [dSPACE]/assetstore_0
assetstore.dir.1 = /mnt/other_filesystem/assetstore_1
```

The above example specifies two asset stores. *assetstore.dir* specifies the asset store number 0 (zero); after that use *assetstore.dir.1*, *assetstore.dir.2* and so on. The particular asset store a bitstream is stored in is held in the database, so don't move bitstreams between asset stores, and don't renumber them.

By default, newly created bitstreams are put in asset store 0 (i.e. the one specified by the *assetstore.dir* property.) This allows backwards compatibility with pre-DSpace 1.1 configurations. To change this, for example when asset store 0 is getting full, add a line to *dSPACE.cfg* like:

```
assetstore.incoming = 1
```



Then restart DSpace (Tomcat). New bitstreams will be written to the asset store specified by *assetstore.dir.1*, which is */mnt/other_filesystem/assetstore_1* in the above example.

Configuring SRB Storage

The same framework is used to configure SRB storage. That is, the asset store number (0..n) can reference a file system directory as above or it can reference a set of SRB account parameters. But any particular asset store number can reference one or the other but not both. This way traditional and SRB storage can both be used but with different asset store numbers. The same cautions mentioned above apply to SRB asset stores as well: The particular asset store a bitstream is stored in is held in the database, so don't move bitstreams between asset stores, and don't renumber them.

For example, let's say asset store number 1 will refer to SRB. Then there will be a set of SRB account parameters like this:

```
srb.host.1 = mysrbmcahost.myu.edu
srb.port.1 = 5544
srb.mcatzone.1 = mysrbzone
srb.mdasdomainname.1 = mysrbdomain
srb.defaultstorageresource.1 = mydefaultsrbresource
srb.username.1 = mysrbuser
srb.password.1 = mysrbpassword
srb.homedirectory.1 = /mysrbzone/home/mysrbuser.mysrbdomain
srb.parentdir.1 = mysrbdspaceassetstore
```

Several of the terms, such as *mcatzone*, have meaning only in the SRB context and will be familiar to SRB users. The last, *srb.parentdir.n*, can be used for addition (SRB) upper directory structure within an SRB account. This property value could be blank as well.

(If asset store 0 would refer to SRB it would be *srb.host = ...*, *srb.port = ...*, and so on (.0 omitted) to be consistent with the traditional storage configuration above.)

The similar use of *assetstore.incoming* to reference asset store 0 (default) or 1..n (explicit property) means that new bitstreams will be written to traditional or SRB storage determined by whether a file system directory on the server is referenced or a set of SRB account parameters are referenced.

There are comments in *dspace.cfg* that further elaborate the configuration of traditional and SRB storage.

8. Directories

8.1. Overview

A complete DSpace installation consists of three separate directory trees:

- **The source directory::** This is where (surprise!) the source code lives. Note that the config files here are used only during the initial install process. After the install, config files should be changed in the install directory. It is referred to in this document as *[dspace-source]*.
- **The install directory::** This directory is populated during the install process and also by DSpace as it runs. It contains config files, command-line tools (and the libraries necessary to run them), and usually—~~although not necessarily~~—the contents of the DSpace archive (depending on how DSpace is configured). After the initial build and install, changes to config files should be made in this directory. It is referred to in this document as *[dspace]*.
- **The web deployment directory::** This directory is generated by the web server the first time it finds a *dspace.war* file in its *webapps* directory. It contains the unpacked contents of *dspace.war*, i.e. the JSPs and java classes and libraries necessary to run DSpace. Files in this directory should never be edited directly; if you wish to modify your DSpace installation, you should edit files in the source directory and then



rebuild. The contents of this directory aren't listed here since its creation is completely automatic. It is usually referred to in this document as *[tomcat]/webapps/dspace*.

8.2. Source Directory Layout

- *[dspace-source]*
 - *dspace/* - Directory which contains all build and configuration information for DSpace
 - *CHANGES* - Detailed list of code changes between versions.
 - *KNOWN_BUGS* - Known bugs in the current version.
 - *LICENSE* - DSpace source code license.
 - *README* - Obligatory basic information file.
 - *bin/* - Some shell and Perl scripts for running DSpace command-line tasks.
 - *config/* - Configuration files:
 - *controlled-vocabularies/* - Fixed, limited vocabularies used in metadata entry
 - *crosswalks/* - Metadata crosswalks - property files or XSL stylesheets
 - *dspace.cfg* - The Main DSpace configuration file (You will need to edit this).
 - *dc2mods.cfg* - Mappings from Dublin Core metadata to <http://www.loc.gov/standards/mods/> for the METS export.
 - *default.license* - The default license that users must grant when submitting items.
 - *dstat.cfg* , *dstat.map* - Configuration for statistical reports.
 - *input-forms.xml* - Submission UI metadata field configuration.
 - *news-side.html* - Text of the front-page news in the sidebar, only used in JSPUI.
 - *news-top.html* - Text of the front-page news in the top box, only used in teh JSPUI.
 - *emails/* - Text and layout templates for emails sent out by the system.
 - *registries/* - **Initial** contents of the bitstream format registry and Dublin Core element/qualifier registry. These are only used on initial system setup, after which they are maintained in the database.
 - *docs/* - DSpace system documentation. The technical documentation for functionality, installation, configuration, etc.
 - *etc/* -
 - This directory contains administrative files needed for the install process and by developers, mostly database initialization and upgrade scripts. Any *.xml* files in *etc/* are common to all supported database systems.
 - *postgres/* - Versions of the database schema and updater SQL scripts for PostgreSQL.
 - *oracle/* - Versions of the database schema and updater SQL scripts for Oracle.
 - *modules/* - The Web UI modules "overlay" directory. DSpace uses Maven to automatically look here for any customizations you wish to make to DSpace Web interfaces.
 - *jspui* - Contains all customizations for the JSP User Interface.



- *src/main/resources/* - The overlay for JSPUI Resources. This is the location to place any custom Messages.properties files. (Previously this file had been stored at: *_[dspace-source]/config/language-packs/Messages.properties_*)
- *src/main/webapp/* - The overlay for JSPUI Web Application. This is the location to place any custom JSPs to be used by DSpace.
- *ini* - Contains all customizations for the Lightweight Network Interface.
- *oai* - Contains all customizations for the OAI-PMH Interface.
- *sword* - Contains all customizations for the SWORD (Simple Web-service Offering Repository Deposit) Interface.
- *xmlui* - Contains all customizations for the XML User Interface (aka Manakin).
- *src/main/webapp/* - The overlay for XMLUI Web Application. This is the location to place custom Themes or Configurations.
 - *i18n/* - The location to place a custom version of the XMLUI's messages.xml (You have to manually create this folder)
 - *themes/* - The location to place custom Themes for the XMLUI (You have to manually create this folder).
- *src/* - Maven configurations for DSpace System. This directory contains the Maven and Ant build files for DSpace.
- *target/* - (Only exists after building DSpace) This is the location Maven uses to build your DSpace installation package.
- *dspace-[version].dir* - The location of the DSpace Installation Package (which can then be installed by running *ant update*)

8.3. Installed Directory Layout

Below is the basic layout of a DSpace installation using the default configuration. These paths can be configured if necessary.

- *[dspace]*
 - *assetstore/* - asset store files
 - *bin/* - shell and Perl scripts
 - *config/* - configuration, with sub-directories as above
 - *handle-server/* - Handles server files
 - *history/* - stored history files (generally RDF/XML)
 - *lib/* - JARs, including *dspace.jar*, containing the DSpace classes
 - *log/* - Log files
 - *reports/* - Reports generated by statistical report generator
 - *search/* - Lucene search index files
 - *upload/* - temporary directory used during file uploads etc.



- *webapps/* - location where DSpace installs all Web Applications

8.4. Contents of JSPUI Web Application

DSpace's Ant build file creates a *dspace-jspui-webapp/* directory with the following structure:

- (top level dir)
 - The JSPs
 - *WEB-INF/*
 - *web.xml* - DSpace JSPUI Web Application configuration and Servlet mappings
 - *dspace-tags.tld* - DSpace custom tag descriptor
 - *fnt.tld* - JSTL message format tag descriptor, for internationalization
 - *lib/* - All the third-party JARs and pre-compiled DSpace API JARs needed to run JSPUI
 - *classes/* - Any additional necessary class files

8.5. Contents of XMLUI Web Application (aka Manakin)

DSpace's Ant build file creates a *dspace-xmlui-webapp/* directory with the following structure:

- (top level dir)
 - *aspects/* - Contains overarching Aspect Generator config and Prototype DRI (Digital Repository Interface) document for Manakin.
 - *i18n/* - Internationalization / Multilingual support. Contains the *messages.xml* English language pack by default.
 - *themes/* - Contains all out-of-the-box Manakin themes
 - *Classic/* - The classic theme, which makes the XMLUI look like classic DSpace
 - *dri2xhtml/* - The base theme, which converts XMLUI DRI (Digital Repository Interface) format into XHTML for display
 - *Reference/* - The default reference theme for XMLUI
 - *template/* - A theme template...useful as a starting point for your own custom theme(s)
 - *dri2xhtml.xsl* - The DRI-to-XHTML XSL Stylesheet. Uses the above 'dri2xhtml' theme to generate XHTML
 - *themes.xmap* - The Theme configuration file. It determines which theme(s) are used by XMLUI
 - *WEB-INF/*
 - *lib/* - All the third-party JARs and pre-compiled DSpace JARs needed to run XMLUI
 - *classes/* - Any additional necessary class files
 - *cocoon.xconf* - XMLUI's Apache Cocoon configuration
 - *logkit.xconf* - XMLUI's Apache Cocoon Logging configuration



- *web.xml* - XMLUI Web Application configuration and Servlet mappings

8.6. Log Files

The first source of potential confusion is the log files. Since DSpace uses a number of third-party tools, problems can occur in a variety of places. Below is a table listing the main log files used in a typical DSpace setup. The locations given are defaults, and might be different for your system depending on where you installed DSpace and the third-party tools. The ordering of the list is roughly the recommended order for searching them for the details about a particular problem or error.

Log File	What's In It
<i>[dspace]/log/dspace.log</i>	Main DSpace log file. This is where the DSpace code writes a simple log of events and errors that occur within the DSpace code. You can control the verbosity of this by editing the <i>[dspace-source]/config/templates/log4j.properties</i> file and then running "ant init_configs". <i>[dspace]/bin/install-configs</i> in <i>[dspace-source]/dspace/target/dspace-1.5.2-build/</i> .
<i>[tomcat]/logs/catalina.out</i>	This is where Tomcat's standard output is written. Many errors that occur within the Tomcat code are logged here. For example, if Tomcat can't find the DSpace code (<i>dspace.jar</i>), it would be logged in <i>catalina.out</i> .
<i>[tomcat]/logs/hostname_log.yyyy-mm-dd.txt</i>	If you're running Tomcat stand-alone (without Apache), it logs some information and errors for specific Web applications to this log file. <i>hostname</i> will be your host name (e.g. <i>dspace.myu.edu</i>) and <i>yyyy-mm-dd</i> will be the date.
<i>[tomcat]/logs/apache_log.yyyy-mm-dd.txt</i>	If you're using Apache, Tomcat logs information about Web applications running through Apache (<i>mod_webapp</i>) in this log file (<i>yyyy-mm-dd</i> being the date.)
<i>[apache]/error_log</i>	Apache logs to this file. If there is a problem with getting <i>mod_webapp</i> working, this is a good place to look for clues. Apache also writes to several other log files, though <i>error_log</i> tends to contain the most useful information for tracking down problems.
<i>[dspace]/log/handle-plugin.log</i>	The Handle server runs as a separate process from the DSpace Web UI (which runs under Tomcat's JVM). Due to a limitation of log4j's 'rolling file appenders', the DSpace code running in the Handle server's JVM must use a separate log file. The DSpace code that is run as part of a Handle resolution request writes log information to this file. You can control the verbosity of this by editing <i>[dspace-source]/config/templates/log4j-handle-plugin.properties</i> .
<i>[dspace]/log/handle-server.log</i>	This is the log file for CNRI's Handle server code. If a problem occurs within the Handle server code, before DSpace's plug-in is invoked, this is where it may be logged.
<i>[dspace]/handle-server/error.log</i>	On the other hand, a problem with CNRI's Handle server code might be logged here.



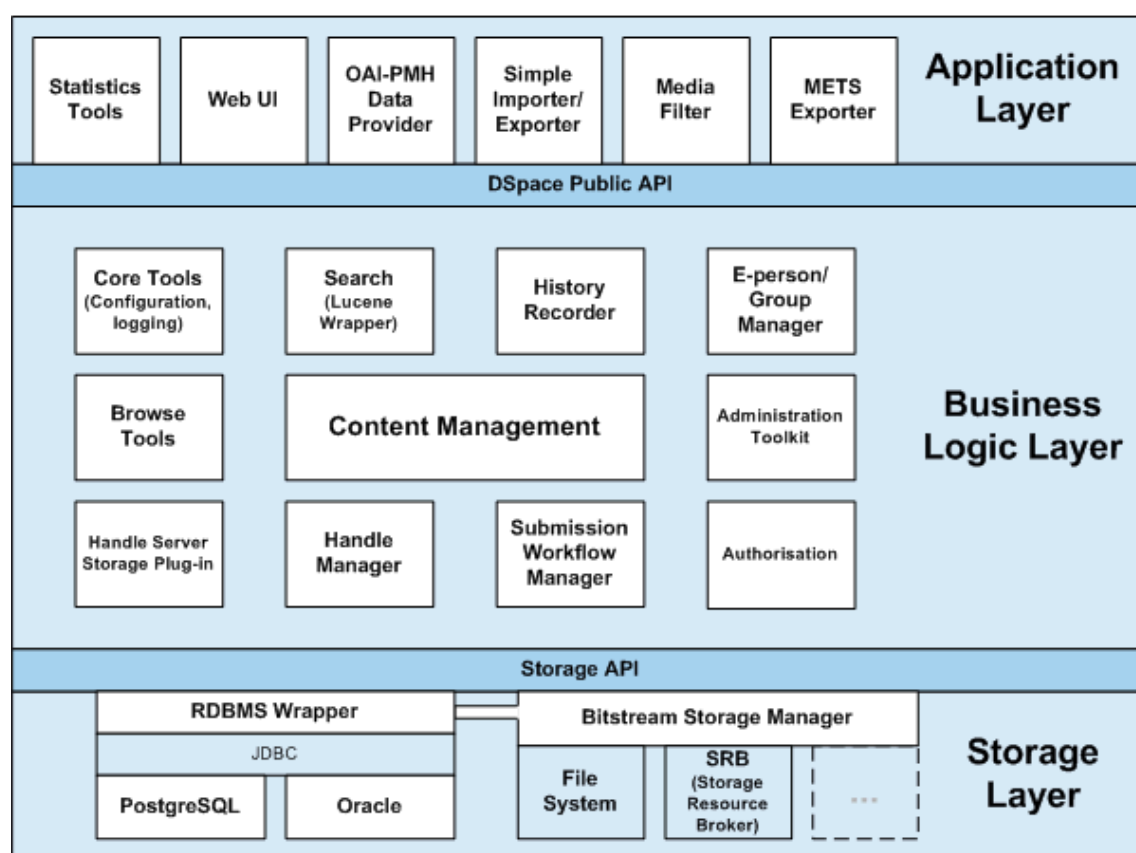
PostgreSQL log

PostgreSQL also writes a log file. This one doesn't seem to have a default location, you probably had to specify it yourself at some point during installation. In general, this log file rarely contains pertinent information--PostgreSQL is pretty stable, you're more likely to encounter problems with connecting via JDBC, and these problems will be logged in *dspace.log*.

9. Architecture

9.1. Overview

The DSpace system is organized into three layers, each of which consists of a number of components.



DSpace System Architecture

The storage layer is responsible for physical storage of metadata and content. The business logic layer deals with managing the content of the archive, users of the archive (e-people), authorization, and workflow. The application layer contains components that communicate with the world outside of the individual DSpace installation, for example the Web user interface and the <http://www.openarchives.org/> protocol for metadata harvesting service.

Each layer only invokes the layer below it; the application layer may not use the storage layer directly, for example. Each component in the storage and business logic layers has a defined public API. The union of the APIs of those components are referred to as the Storage API (in the case of the storage layer) and the DSpace Public API (in the case of the business logic layer). These APIs are in-process Java classes, objects and methods.



It is important to note that each layer is *trusted*. Although the logic for *authorising actions* is in the business logic layer, the system relies on individual applications in the application layer to correctly and securely *authenticate* e-people. If a 'hostile' or insecure application were allowed to invoke the Public API directly, it could very easily perform actions as any e-person in the system.

The reason for this design choice is that authentication methods will vary widely between different applications, so it makes sense to leave the logic and responsibility for that in these applications.

The source code is organized to cohere very strictly to this three-layer architecture. Also, only methods in a component's public API are given the *public* access level. This means that the Java compiler helps ensure that the source code conforms to the architecture.

Packages within	Correspond to components in
<i>org.dspace.app</i>	Application layer
<i>org.dspace</i>	Business logic layer (except <i>storage</i> and <i>app</i>)
<i>org.dspace.storage</i>	Storage layer

The storage and business logic layer APIs are extensively documented with Javadoc-style comments. Generate the HTML version of these by entering the [dspace-source]/dspace directory and running:

```
mvn javadoc:javadoc
```

The resulting documentation will be at *[dspace-source]dspace-api/target/site/apidocs/index.html*. The package-level documentation of each package usually contains an overview of the package and some example usage. This information is not repeated in this architecture document; this and the Javadoc APIs are intended to be used in parallel.

Each layer is described in a separate section:

- Storage Layer
 - RDBMS
 - Bitstream Store
- Business Logic Layer
 - Core Classes
 - Content Management API
 - Workflow System
 - Administration Toolkit
 - E-person/Group Manager
 - Authorisation
 - Handle Manager/Handle Plugin
 - Search
 - Browse API
 - History Recorder
 - Checksum Checker
- Application Layer



- Web User Interface
 - OAI-PMH Data Provider
 - Item Importer and Exporter
 - Transferring Items Between DSpace Instances
 - Registration
 - METS Tools
 - Media Filters
 - Sub-Community Management
- 2002-2008 The DSpace Foundation

10. Application

10.1. Web User Interface

The DSpace Web UI is the largest and most-used component in the application layer. Built on Java Servlet and JavaServer Page technology, it allows end-users to access DSpace over the Web via their Web browsers. As of Dspace 1.3.2 the UI meets both XHTML 1.0 standards and Web Accessibility Initiative (WAI) level-2 standard.

It also features an administration section, consisting of pages intended for use by central administrators. Presently, this part of the Web UI is not particularly sophisticated; users of the administration section need to know what they are doing! Selected parts of this may also be used by collection administrators.

10.1.1. Web UI Files

The Web UI-related files are located in a variety of directories in the DSpace source tree. Note that as of DSpace version 1.5, the deployment has changed. The build systems has moved to a maven-based system enabling the various projects (JSPUI, XMLUI, etc.) into separate projects. The system still uses the familiar 'Ant' to deploy the webapps in later stages.

Location	Description
<i>[dspace-source]/dspace-jspui/dspace-jspui-api/src/main/java/org/dspace/app/webui</i>	Web UI source files
<i>[dspace-source]/dspace-jspui/dspace-jspui-api/src/main/java/org/dspace/app/filters</i>	Servlet Filters (Servlet 2.3 spec)
<i>[dspace-source]/dspace-jspui/dspace-jspui-api/src/main/java/org/dspace/app/jsptag</i>	Custom JSP tag class files
<i>[dspace-source]/dspace-jspui/dspace-jspui-api/src/main/java/org/dspace/app/servlet</i>	Servlets for main Web UI (controllers)
<i>[dspace-source]/dspace-jspui/dspace-jspui-api/src/main/java/org/dspace/app/servlet/admin</i>	Servlets that comprise the administration part of the Web UI
<i>[dspace-source]/dspace-jspui/dspace-jspui-api/src/main/java/org/dspace/app/webui/util/</i>	Miscellaneous classes used by the servlets and filters
<i>[dspace-source]/dspace-jspui</i>	The JSP files
<i>[dspace-source]/dspace/modules/jspui/src/main/webapp</i>	This is where you place customized versions of JSPs —see 6. JSPUI Configuration and Customization



<code>[dspace-source]/dspace/modules/xmlui/src/main/webapp</code>	This is where you place customizations for the Manakin interface—see 7. Manakin [XMLUI] Configuration and Customization
<code>[dspace-source]/dspace/modules/jspui/src/main/resources</code>	This is where you can place you customize version of the <i>Messages.properties</i> file.
<code>[dspace-source]/dspace-jspui/dspace-jspui-webapp/src/main/webapp/WEB-INF/dspace-tags.tld</code>	Custom DSpace JSP tag descriptor

10.1.2. The Build Process

The DSpace build process constructs a Web application archive, which is placed in `[dspace-source]/build/dspace.war`. The `build_wars` Ant target does the work. The process works as follows:

- All the DSpace source code is compiled.
 - `[dspace-source]/etc/dspace-web.xml` is copied to `[dspace-source]/build` and the `@@dspace.dir@@` token inside it replaced with the DSpace installation directory (`dspace.dir` property from `dspace.cfg`)
 - The JSPs are all copied to `[dspace-source]/build/jsp`
 - Customized JSPs from `[dspace-source]/jsp/local` are copied on top of these, thus 'overriding' the default versions
 - `[dspace-source]/build/dspace.war` is built
The contents of `dspace.war` are:
 - (Top level) – the JSPs (customized versions from `[dspace-source]/jsp/local` will have overwritten the defaults from the DSpace source distribution)
 - `WEB-INF/classes` – the compiled DSpace classes
 - `WEB-INF/lib` – the third party library JAR files from `[dspace-source]/lib`, minus `servlet.jar` which will be available as part of Tomcat (or other servlet engine)
 - `WEB-INF/web.xml` – web deployment descriptor, copied from `[dspace-source]/build/dspace-web.xml`
 - `WEB-INF/dspace-tags.tld` – tag descriptor
- Note that this does mean there are multiple copies of the compiled DSpace code and third-party libraries in the system, so care must be taken to ensure that they are all in sync. (The storage overhead is a few megabytes, totally insignificant these days.) In general, when you change any DSpace code or JSP, it's best to do a complete update of both the installation (`[dspace]`), and to rebuild and redeploy the Web UI and OAI `.war` files, by running this in `[dspace-source]`:

```
ant -D [dspace]/config/dspace.cfg update
```

and then following the instructions that command writes to the console.

10.1.3. Servlets and JSPs

The Web UI is loosely based around the MVC (model, view, controller) model. The content management API corresponds to the model, the Java Servlets are the controllers, and the JSPs are the views. Interactions take the following basic form:

1. An HTTP request is received from a browser



2. The appropriate servlet is invoked, and processes the request by invoking the DSpace business logic layer public API
3. Depending on the outcome of the processing, the servlet invokes the appropriate JSP
4. The JSP is processed and sent to the browser
The reasons for this approach are:
 - All of the processing is done before the JSP is invoked, so any error or problem that occurs does not occur halfway through HTML rendering
 - The JSPs contain as little code as possible, so they can be customized without having to delve into Java code too much

The *org.dspace.app.webui.servlet.LoadDSpaceConfig* servlet is always loaded first. This is a very simple servlet that checks the *dspace-config* context parameter from the DSpace deployment descriptor, and uses it to locate *dspace.cfg*. It also loads up the Log4j configuration. It's important that this servlet is loaded first, since if another servlet is loaded up, it will cause the system to try and load DSpace and Log4j configurations, neither of which would be found.

All DSpace servlets are subclasses of the *DSpaceServlet* class. The *DSpaceServlet* class handles some basic operations such as creating a DSpace *Context* object (opening a database connection etc.), authentication and error handling. Instead of overriding the *doGet* and *doPost* methods as one normally would for a servlet, DSpace servlets implement *doDSGet* or *doDSPost* which have an extra context parameter, and allow the servlet to throw various exceptions that can be handled in a standard way.

The DSpace servlet processes the contents of the HTTP request. This might involve retrieving the results of a search with a query term, accessing the current user's eperson record, or updating a submission in progress. According to the results of this processing, the servlet must decide which JSP should be displayed. The servlet then fills out the appropriate attributes in the *HttpRequest* object that represents the HTTP request being processed. This is done by invoking the *setAttribute* method of the *javax.servlet.http.HttpServletRequest* object that is passed into the servlet from Tomcat. The servlet then forwards control of the request to the appropriate JSP using the *JSPManager.showJSP* method.

The *JSPManager.showJSP* method uses the standard Java servlet forwarding mechanism is then used to forward the HTTP request to the JSP. The JSP is processed by Tomcat and the results sent back to the user's browser.

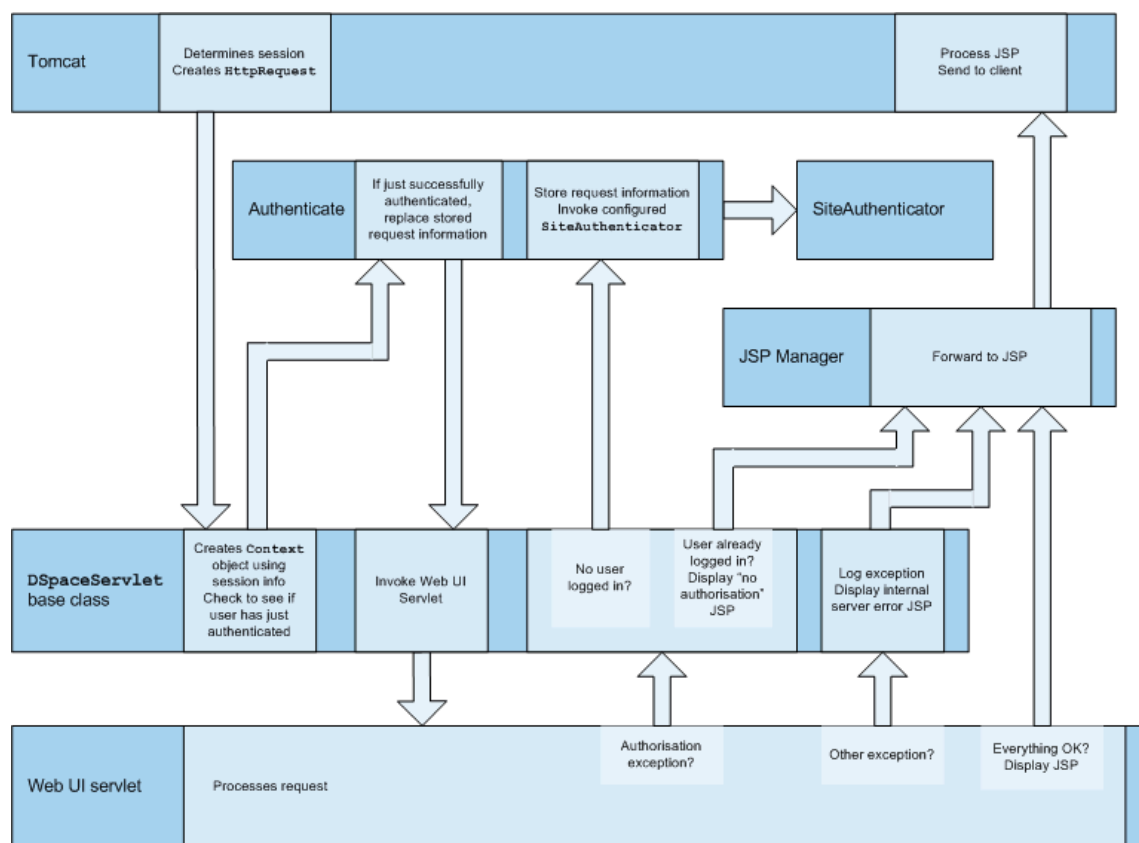
There is an exception to this servlet/JSP style: *index.jsp*, the 'home page', receives the HTTP request directly from Tomcat without a servlet being invoked first. This is because in the servlet 2.3 specification, there is no way to map a servlet to handle only requests made to '/'; such a mapping results in every request being directed to that servlet. By default, Tomcat forwards requests to '/' to *index.jsp*. To try and make things as clean as possible, *index.jsp* contains some simple code that would normally go in a servlet, and then forwards to *home.jsp* using the *JSPManager.showJSP* method. This means localized versions of the 'home page' can be created by placing a customized *home.jsp* in *[dspace-source]/jsp/local*, in the same manner as other JSPs.

[dspace-source]/jsp/dspace-admin/index.jsp, the administration UI index page, is invoked directly by Tomcat and not through a servlet for similar reasons.

At the top of each JSP file, right after the license and copyright header, is documented the appropriate attributes that a servlet must fill out prior to forwarding to that JSP. No validation is performed; if the servlet does not fill out the necessary attributes, it is likely that an internal server error will occur.

Many JSPs containing forms will include hidden parameters that tell the servlets which form has been filled out. The submission UI servlet (*SubmissionController* is a prime example of a servlet that deals with the input from many different JSPs. The *step* and *page* hidden parameters (written out by the *SubmissionController.getSubmissionParameters()* method) are used to inform the servlet which page of which step has just been filled out (i.e. which page of the submission the user has just completed).

Below is a detailed, scary diagram depicting the flow of control during the whole process of processing and responding to an HTTP request. More information about the authentication mechanism is mostly described in the configuration section.



Flow of Control During HTTP Request Processing

10.1.4. Custom JSP Tags

The DSpace JSPs all use some custom tags defined in `/dspace/jsp/WEB-INF/dspace-tags.tld`, and the corresponding Java classes reside in `org.dspace.app.webui.jsptag`. The tags are listed below. The `dspace-tags.tld` file contains detailed comments about how to use the tags, so that information is not repeated here.

- **layout**: Just about every JSP uses this tag. It produces the standard HTML header and `<BODY>_tag`. Thus the content of each JSP is nested inside a `_<dspace:layout>` tag. The (XML-style) attributes of this tag are slightly complicated--see `dspace-tags.tld`. The JSPs in the source code bundle also provide plenty of examples.
- **sidebar**: Can only be used inside a `layout` tag, and can only be used once per JSP. The content between the start and end `sidebar` tags is rendered in a column on the right-hand side of the HTML page. The contents can contain further JSP tags and Java 'scriptlets'.
- **date**: Displays the date represented by an `org.dspace.content.DCDate` object. Just the one representation of date is rendered currently, but this could use the user's browser preferences to display a localized date in the future.
- **include**: Obsolete, simple tag, similar to `jsp:include`. In versions prior to DSpace 1.2, this tag would use the locally modified version of a JSP if one was installed in `jsp/local`. As of 1.2, the build process now performs this function, however this tag is left in for backwards compatibility.
- **item**: Displays an item record, including Dublin Core metadata and links to the bitstreams within it. Note that the displaying of the bitstream links is simplistic, and does not take into account any of the bundling structure. This is because DSpace does not have a fully-fledged dissemination architectural piece yet. Displaying an item record is done by a tag rather than a JSP for two reasons: Firstly, it happens in several places (when verifying an item record during submission or workflow review, as well as during standard item accesses), and secondly, displaying the item turns out to be mostly code-work rather than HTML any-



way. Of course, the disadvantage of doing it this way is that it is slightly harder to customize exactly what is displayed from an item record; it is necessary to edit the tag code (*org.dspace.app.webui.jsptag.ItemTag*). Hopefully a better solution can be found in the future.

- **itemlist, collectionlist, communitylist:** These tags display ordered sequences of items, collections and communities, showing minimal information but including a link to the page containing full details. These need to be used in HTML tables.
- **popup:** This tag is used to render a link to a pop-up page (typically a help page.) If Javascript is available, the link will either open or pop to the front any existing DSpace pop-up window. If Javascript is not available, a standard HTML link is displayed that renders the link destination in a window named '*dspace.popup*'. In graphical browsers, this usually opens a new window or re-uses an existing window of that name, but if a window is re-used it is not 'raised' which might confuse the user. In text browsers, following this link will simply replace the current page with the destination of the link. This obviously means that Javascript offers the best functionality, but other browsers are still supported.
- **selectperson:** A tag which produces a widget analogous to HTML `<SELECT>`, that allows a user to select one or multiple e-people from a pop-up list.
- **sfxlink:** Using an item's Dublin Core metadata DSpace can display an SFX link, if an SFX server is available. This tag does so for a particular item if the *sfx.server.url* property is defined in *dspace.cfg*.

10.1.5. Internationalization

The <http://jakarta.apache.org/taglibs/doc/standard-1.0-doc/intro.html> is used to specify messages in the JSPs like this:

OLD:

```
<H1>Search Results</H1>
```

NEW:

```
<H1><fmt:message key="jsp.search.results.title"/></H1>
```

This message can now be changed using the *config/language-packs/Messages.properties* file. (This must be done at build-time: *Messages.properties* is placed in the *dspace.war* Web application file.)

```
jsp.search.results.title = Search Results
```

Phrases may have parameters to be passed in, to make the job of translating easier, reduce the number of 'keys' and to allow translators to make the translated text flow more appropriately for the target language.

OLD:

```
<P>Results <%= r.getFirst() %> to <%= r.getLast() %> of <%=r.getTotal() %></P>
```

NEW:

```
<fmt:message key="jsp.search.results.text">
  <fmt:param><%= r.getFirst() %></fmt:param>
  <fmt:param><%= r.getLast() %></fmt:param>
  <fmt:param><%= r.getTotal() %></fmt:param>
</fmt:message>
```

(Note: JSTL 1.0 does not seem to allow JSP `<%= %>` expressions to be passed in as values of attribute in `<fmt:param value=""/>`)

The above would appear in the *Messages_xx.properties* file as:

```
jsp.search.results.text = Results {0}-{1} of {2}
```



Introducing number parameters that should be formatted according to the locale used makes no difference in the message key compared to string parameters:

```
jsp.submit.show-uploaded-file.size-in-bytes = {0} bytes
```

In the JSP using this key can be used in the way below:

```
<fmt:message key="jsp.submit.show-uploaded-file.size-in-bytes">
  <fmt:param><fmt:formatNumber><%= bitstream.getSize() %></fmt:formatNumber></fmt:param>
</fmt:message>
```

(Note: JSTL offers a way to include numbers in the message keys as `_jsp.foo.key =`

Unknown macro: {0,number}

`bytes_`. Setting the parameter as `_<fmt:param value="$`

Unknown macro: {variable}

`</>` works when *variable* is a single variable name and doesn't work when trying to use a method's return value instead: `bitstream.getSize()`. Passing the number as string (or using the `<%= %>` expression) also does not work.)

Multiple *Messages.properties* can be created for different languages. See [http://java.sun.com/j2se/1.4.2/docs/api/java/util/ResourceBundle.html#getBundle\(java.lang.String,%20java.util.Locale,%20java.lang.ClassLoader\)](http://java.sun.com/j2se/1.4.2/docs/api/java/util/ResourceBundle.html#getBundle(java.lang.String,%20java.util.Locale,%20java.lang.ClassLoader)). e.g. you can add German and Canadian French translations:

```
Messages_de.properties
Messages_fr_CA.properties
```

The end user's browser settings determine which language is used. The English language file *Messages.properties* (or the default server locale) will be used as a default if there's no language bundle for the end user's preferred language. (Note that the English file is not called *Messages_en.properties* – this is so it is always available as a default, regardless of server configuration.)

The *dSPACE:layout* tag has been updated to allow dictionary keys to be passed in for the titles. It now has two new parameters: *titlekey* and *parenttitlekey*. So where before you'd do:

```
<dSPACE:layout title="Here"
  parentlink="/myspace"
  parenttitle="My DSpace">
```

You now do:

```
<dSPACE:layout titlekey="jsp.page.title"
  parentlink="/myspace"
  parenttitlekey="jsp.myspace">
```

And so the layout tag itself gets the relevant stuff out of the dictionary. *title* and *parenttitle* still work as before for backwards compatibility, and the odd spot where that's preferable.

Message Key Convention

When translating further pages, please follow the convention for naming message keys to avoid clashes.

For text in JSPs use the complete path + filename of the JSP, then a one-word name for the message. e.g. for the title of *jsp/myspace/main.jsp* use:

```
jsp.myspace.main.title
```

Some common words (e.g. "Help") can be brought out into keys starting *jsp*. for ease of translation, e.g.:

```
jsp.admin = Administer
```



Other common words/phrases are brought out into 'general' parameters if they relate to a set (directory) of JSPs, e.g.

```
jsp.tools.general.delete = Delete
```

Phrases that relate **strongly** to a topic (eg. MyDSpace) but used in many JSPs outside the particular directory are more convenient to be cross-referenced. For example one could use the key below in *jsp/submit/saved.jsp* to provide a link back to the user's *MyDSpace*:

*(Cross-referencing of keys **in general** is not a good idea as it may make maintenance more difficult. But in some cases it has more advantages as the meaning is obvious.)*

```
jsp.mydspace.general.goto-mydspace = Go to My DSpace
```

For text in servlet code, in custom JSP tags or wherever applicable use the fully qualified classname + a one-word name for the message. e.g.

```
org.dspace.app.webui.jsptag.ItemListTag.title = Title
```

Which Languages are currently supported?

To view translations currently being developed, please refer to the <http://wiki.dspace.org/I18nSupport> of the DSpace Wiki.

10.1.6. HTML Content in Items

For the most part, the DSpace item display just gives a link that allows an end-user to download a bitstream. However, if a bundle has a primary bitstream whose format is of MIME type *text/html*, instead a link to the HTML servlet is given.

So if we had an HTML document like this:

```
contents.html
chapter1.html
chapter2.html
chapter3.html
figure1.gif
figure2.jpg
figure3.gif
figure4.jpg
figure5.gif
figure6.gif
```

The Bundle's primary bitstream field would point to the *contents.html* Bitstream, which we know is HTML (check the format MIME type) and so we know which to serve up first.

The HTML servlet employs a trick to serve up HTML documents without actually modifying the HTML or other files themselves. Say someone is looking at *contents.html* from the above example, the URL in their browser will look like this:

```
https://dspace.mit.edu/html/1721.1/12345/contents.html
```

If there's an image called *figure1.gif* in that HTML page, the browser will do HTTP GET on this URL:

```
https://dspace.mit.edu/html/1721.1/12345/figure1.gif
```

The HTML document servlet can work out which item the user is looking at, and then which Bitstream in it is called *figure1.gif*, and serve up that bitstream. Similar for following links to other HTML pages. Of course all the links and image references have to be relative and not absolute.



HTML documents must be "self-contained", as explained here. Provided that full path information is known by DSpace, any depth or complexity of HTML document can be served subject to those constraints. This is usually possible with some kind of batch import. If, however, the document has been uploaded one file at a time using the Web UI, the path information has been stripped. The system can cope with relative links that refer to a deeper path, e.g.

```
<IMG SRC="images/figure1.gif">
```

If the item has been uploaded via the Web submit UI, in the Bitstream table in the database we have the 'name' field, which will contain the filename with no path (*figure1.gif*). We can still work out what *images/figure1.gif* is by making the HTML document servlet strip any path that comes in from the URL, e.g.

```
https://dspace.mit.edu/html/1721.1/12345/images/figure1.gif
          ^^^^^^^
          Strip this
```

BUT all the filenames (regardless of directory names) must be unique. For example, this wouldn't work:

```
contents.html
chapter1.html
chapter2.html
chapter1_images/figure.gif
chapter2_images/figure.gif
```

since the HTML document servlet wouldn't know which bitstream to serve up for:

```
https://dspace.mit.edu/html/1721.1/12345/chapter1_images/figure.gif
https://dspace.mit.edu/html/1721.1/12345/chapter2_images/figure.gif
```

since it would just have *figure.gif*

To prevent "infinite URL spaces" appearing (e.g. if a file *foo.html* linked to *bar/foo.html*, which would link to *bar/bar/foo.html...*) this behavior can be configured by setting the configuration property *webui.html.max-depth-guess*.

For example, if we receive a request for *foo/bar/index.html*, and we have a bitstream called just *index.html*, we will serve up that bitstream for the request if *webui.html.max-depth-guess* is 2 or greater. If *webui.html.max-depth-guess* is 1 or less, we would not serve that bitstream, as the depth of the file is greater. If *webui.html.max-depth-guess* is zero, the request filename and path must always exactly match the bitstream name. The default value (if that property is not present in *dspace.cfg*) is 3.

10.1.7. Thesis Blocking

The submission UI has an optional feature that came about as a result of MIT Libraries policy. If the *block.theses* parameter in *dspace.cfg* is *true*, an extra checkbox is included in the first page of the submission UI. This asks the user if the submission is a thesis. If the user checks this box, the submission is halted (deleted) and an error message displayed, explaining that DSpace should not be used to submit theses. This feature can be turned off and on, and the message displayed (*/dspace/jsp/submit/no-theses.jsp* can be localized as necessary).

10.2. OAI-PMH Data Provider

The DSpace platform supports the <http://www.openarchives.org/> (OAI-PMH) version 2.0 as a data provider. This is accomplished using the <http://www.oclc.org/research/software/oai/cat.shtm>.

The DSpace build process builds a Web application archive, [*dspace-source*]/*build/oai.war*), in much the same way as the Web UI build process described above. The only differences are that the JSPs are not included, and [*dspace-source*]/*etc/oai-web.xml* is used as the deployment descriptor. This 'webapp' is deployed to receive and respond to OAI-PMH requests via HTTP. Note that typically it should *not* be deployed on SSL (*https*: protocol). In a typical configuration, this is deployed at *oai*, for example:



```
http://dspace.myu.edu/oai/request?verb=Identify
```

The 'base URL' of this DSpace deployment would be:

```
http://dspace.myu.edu/oai/request
```

It is this URL that should be registered with <http://www.openarchives.org/>. Note that you can easily change the 'request' portion of the URL by editing `[dspace-source]/etc/oai-web.xml` and rebuilding and deploying `oai.war`.

DSpace provides implementations of the OAICat interfaces *AbstractCatalog*, *RecordFactory* and *Crosswalk* that interface with the DSpace content management API and harvesting API (in the search subsystem).

Only the basic *oai_dc* unqualified Dublin Core metadata set export is enabled by default; this is particularly easy since all items have qualified Dublin Core metadata. When this metadata is harvested, the qualifiers are simply stripped; for example, *description.abstract* is exposed as unqualified *description*. The *description.provenance* field is hidden, as this contains private information about the submitter and workflow reviewers of the item, including their e-mail addresses. Additionally, to keep in line with OAI community practices, values of *contributor.author* are exposed as *creator* values.

Other metadata formats are supported as well, using other *Crosswalk* implementations; consult the *oai.cat.properties* file described below. To enable a format, simply uncomment the lines beginning with *Crosswalks.**. Multiple formats are allowed, and the current list includes, in addition to unqualified DC: MPEG DIDL, METS, MODS. There is also an incomplete, experimental qualified DC.

Note that the current simple DC implementation (*org.dspace.app.oai.OAIDCCrosswalk*) does not currently strip out any invalid XML characters that may be lying around in the data. If your database contains a DC value with, for example, some ASCII control codes (form feed etc.) this may cause OAI harvesters problems. This should rarely occur, however. XML entities (such as `>`) are encoded (e.g. to `>`)

In addition to the implementations of the OAICat interfaces, there are two configuration files relevant to OAI support:

- **oai.cat.properties:** This resides as a template in `[dspace]/config/templates`, and the live version is written to `[dspace]/config`. You probably won't need to edit this; the *install-configs* script fills out the relevant deployment-specific parameters. You might want to change the *earliestDatestamp* field to accurately reflect the oldest datestamp in the system. (Note that this is the value of the *last_modified* column in the *Item* database table.)
- **oai-web.xml:** This standard Java Servlet 'deployment descriptor' is stored in the source as `[dspace-source]/etc/oai-web.xml`, and is written to `/dspace/oai/WEB-INF/web.xml`. Sets

OAI-PMH allows repositories to expose an hierarchy of sets in which records may be placed. A record can be in zero or more sets.

DSpace exposes collections as sets. The organization of communities is likely to change over time, and is therefore a less stable basis for selective harvesting.

Each collection has a corresponding OAI set, discoverable by harvesters via the ListSets verb. The setSpec is the Handle of the collection, with the ':' and '/' converted to underscores so that the Handle is a legal setSpec, for example:

```
hdl_1721.1_1234
```

Naturally enough, the collection name is also the name of the corresponding set.

10.2.1. Unique Identifier

Every item in OAI-PMH data repository must have an unique identifier, which must conform to the URI syntax. As of DSpace 1.2, Handles are not used; this is because in OAI-PMH, the OAI identifier identifies the



metadata record associated with the *resource*. The *resource* is the DSpace item, whose *resource identifier* is the Handle. In practical terms, using the Handle for the OAI identifier may cause problems in the future if DSpace instances share items with the same Handles; the OAI metadata record identifiers should be different as the different DSpace instances would need to be harvested separately and may have different metadata for the item.

The OAI identifiers that DSpace uses are of the form:

oai:host name:handle

For example:

oai:dspace.myu.edu:123456789/345

If you wish to use a different scheme, this can easily be changed by editing the value of *OAI_ID_PREFIX* at the top of the *org.dspace.app.oai.DSpaceOAI Catalog* class. (You do not need to change the code if the above scheme works for you; the code picks up the host name and Handles automatically from the DSpace configuration.)

10.2.2. Access control

OAI provides no authentication/authorisation details, although these could be implemented using standard HTTP methods. It is assumed that all access will be anonymous for the time being.

A question is, "is all metadata public?" Presently the answer to this is yes; all metadata is exposed via OAI-PMH, even if the item has restricted access policies. The reasoning behind this is that people who do actually have permission to read a restricted item should still be able to use OAI-based services to discover the content.

If in the future, this 'expose all metadata' approach proves unsatisfactory for any reason, it should be possible to expose only publicly readable metadata. The authorisation system has separate permissions for READING and item and READING the content (bitstreams) within it. This means the system can differentiate between an item with public metadata and hidden content, and an item with hidden metadata as well as hidden content. In this case the OAI data repository should only expose items those with anonymous READ access, so it can hide the existence of records to the outside world completely. In this scenario, one should be wary of protected items that are made public after a time. When this happens, the items are "new" from the OAI-PMH perspective.

10.2.3. Modification Date (OAI Date Stamp)

OAI-PMH harvesters need to know when a record has been created, changed or deleted. DSpace keeps track of a 'last modified' date for each item in the system, and this date is used for the OAI-PMH date stamp. This means that any changes to the metadata (e.g. admins correcting a field, or a withdrawal) will be exposed to harvesters.

10.2.4. 'About' Information

As part of each record given out to a harvester, there is an optional, repeatable "about" section which can be filled out in any (XML-schema conformant) way. Common uses are for provenance and rights information, and there are schemas in use by OAI communities for this. Presently DSpace does not provide any of this information.

10.2.5. Deletions

DSpace keeps track of deletions (withdrawals). These are exposed via OAI, which has a specific mechanism for dealing with this. Since DSpace keeps a permanent record of withdrawn items, in the OAI-PMH sense DSpace supports deletions 'persistently'. This is as opposed to 'transient' deletion support, which would mean that deleted records are forgotten after a time.



Once an item has been withdrawn, OAI-PMH harvests of the date range in which the withdrawal occurred will find the 'deleted' record header. Harvests of a date range prior to the withdrawal will *not* find the record, despite the fact that the record did exist at that time.

As an example of this, consider an item that was created on 2002-05-02 and withdrawn on 2002-10-06. A request to harvest the month 2002-10 will yield the 'record deleted' header. However, a harvest of the month 2002-05 will not yield the original record.

Note that presently, the deletion of 'expunged' items is not exposed through OAI.

10.2.6. Flow Control (Resumption Tokens)

An OAI data provider can prevent any performance impact caused by harvesting by forcing a harvester to receive data in time-separated chunks. If the data provider receives a request for a lot of data, it can send part of the data with a resumption token. The harvester can then return later with the resumption token and continue.

DSpace supports resumption tokens for 'ListRecords' OAI-PMH requests. ListIdentifiers and ListSets requests do not produce a particularly high load on the system, so resumption tokens are not used for those requests.

Each OAI-PMH ListRecords request will return at most 100 records. This limit is set at the top of *org.dspace.app.oai.DSpaceOAICatalog.java* (*MAX_RECORDS*). A potential issue here is that if a harvest yields an exact multiple of *MAX_RECORDS*, the last operation will result in a harvest with no records in it. It is unclear from the OAI-PMH specification if this is acceptable.

When a resumption token is issued, the optional *completeListSize* and *cursor* attributes are not included. OAICat sets the *expirationDate* of the resumption token to one hour after it was issued, though in fact since DSpace resumption tokens contain all the information required to continue a request they do not actually expire.

Resumption tokens contain all the state information required to continue a request. The format is:

```
from/until/setSpec/offset
```

from and *until* are the ISO 8601 dates passed in as part of the original request, and *setSpec* is also taken from the original request. *offset* is the number of records that have already been sent to the harvester. For example:

```
2003-01-01//hdl_1721_1_1234/300
```

This means the harvest is 'from'

2003-01-01, has no 'until' date, is for collection hdl:1721.1/1234, and 300 records have already been sent to the harvester. (Actually, if the original OAI-PMH request doesn't specify a 'from' or 'until', OAICat fills them out automatically to '0000-00-00T00:00:00Z' and '9999-12-31T23:59:59Z' respectively. This means DSpace resumption tokens will always have from and until dates in them.)

10.3. DSpace Command Launcher

Introduced in Release 1.6, the DSpace Command Launcher brings together the various command and scripts into a standard-practice for running CLI runtime programs.

10.3.1. Older Versions

Prior to Release 1.6, there were various scripts written that masked a more manual approach to running CLI programs. The user had to issue *[dspace]/bin/dsrun* and then java class that ran that program. With release 1.5, scripts were written to mask the *[dspace]/bin/dsrun* command. We have left the java class in the System Administration section since it does have value for debugging purposes and for those who wish to learn about DSpace programming or wish to customize the code at any time.



10.3.2. Command Launcher Structure

There are two components to the command launcher: the `dspace` script and the `launcher.xml`. The DSpace command calls a java class which in turn refers to `launcher.xml` that is stored in the `[dspace]/config` directory

`launcher.xml` is made of several components:

- `<command>` begins the stanza for a comand
- `<name>_name of command_</name>` the name of the command that you would use.
- `<description>_the description of the command_</description>`
- `<step> </step>` User arguments are parsed and tested.
- `<class>_<the java class that is being used to run the CLI program>_</class>`

Prior to release 1.5 if one wanted to regenerate the browse index, one would have to issue the following commands manually:

```
[dspace]/bin/dsrun org.dspace.browse.IndexBrowse -f -r
[dspace]/bin/dsrun org.dspace.browse.ItemCounter
[dspace]/bin/dsrun org.dspace.search.DSIndexer
```

In release 1.5 a script was written and in release 1.6 the command `[dspace]/bin/dspace index-init` replaces the script. The stanza from `launcher.xml` show us how one can build more commands if needed:

```
<command>
  <name>index-update</name>
  <description>Update the search and browse indexes</description>
  <step passuserargs="false">
    <class>org.dspace.browse.IndexBrowse</class>
    <argument>-i</argument>
  </step>
  <step passuserargs="false">
    <class>org.dspace.browse.ItemCounter</class>
  </step>
  <step passuserargs="false">
    <class>org.dspace.search.DSIndexer</class>
  </step>
</command>
```

11. Business

11.1. Core Classes

The `org.dspace.core` package provides some basic classes that are used throughout the DSpace code.

11.1.1. The Configuration Manager (ConfigurationManager)

The configuration manager is responsible for reading the main `dspace.cfg` properties file, managing the 'template' configuration files for other applications such as Apache, and for obtaining the text for e-mail messages.

The system is configured by editing the relevant files in `/dspace/config`, as described in the configuration section.

When editing configuration files for applications that DSpace uses, such as Apache, remember to edit the file in `/dspace/config/templates` and then run `/dspace/bin/install-configs` rather than editing the 'live' version directly!



The *ConfigurationManager* class can also be invoked as a command line tool, with two possible uses:

- `_/dspace/bin/install-configs_` This processes and installs configuration files for other applications, as described in the configuration section.
- `/dspace/bin/dsrun org.dspace.core.ConfigurationManager -property property.name_` This writes the value of `_property.name` from `dspace.cfg` to the standard output, so that shell scripts can access the DSpace configuration. For an example, see `/dspace/bin/start-handle-server`. If the property has no value, nothing is written.

11.1.2. Constants

This class contains constants that are used to represent types of object and actions in the database. For example, authorization policies can relate to objects of different types, so the *resourcepolicy* table has columns *resource_id*, which is the internal ID of the object, and *resource_type_id*, which indicates whether the object is an item, collection, bitstream etc. The value of *resource_type_id* is taken from the *Constants* class, for example *Constants.ITEM*.

11.1.3. Context

The *Context* class is central to the DSpace operation. Any code that wishes to use the any API in the business logic layer must first create itself a *Context* object. This is akin to opening a connection to a database (which is in fact one of the things that happens.)

A context object is involved in most method calls and object constructors, so that the method or object has access to information about the current operation. When the context object is constructed, the following information is automatically initialized:

- A connection to the database. This is a transaction-safe connection. i.e. the 'auto-commit' flag is set to false.
- A cache of content management API objects. Each time a content object is created (for example *Item* or *Bitstream*) it is stored in the *Context* object. If the object is then requested again, the cached copy is used. Apart from reducing database use, this addresses the problem of having two copies of the same object in memory in different states.

The following information is also held in a context object, though it is the responsibility of the application creating the context object to fill it out correctly:

- The current authenticated user, if any
- Any 'special groups' the user is a member of. For example, a user might automatically be part of a particular group based on the IP address they are accessing DSpace from, even though they don't have an e-person record. Such a group is called a 'special group'.
- Any extra information from the application layer that should be added to log messages that are written within this context. For example, the Web UI adds a session ID, so that when the logs are analysed the actions of a particular user in a particular session can be tracked.
- A flag indicating whether authorization should be circumvented. This should only be used in rare, specific circumstances. For example, when first installing the system, there are no authorized administrators who would be able to create an administrator account! As noted above, the public API is *trusted*, so it is up to applications in the application layer to use this flag responsibly.

Typical use of the context object will involve constructing one, and setting the current user if one is authenticated. Several operations may be performed using the context object. If all goes well, *complete* is called to commit the changes and free up any resources used by the context. If anything has gone wrong, *abort* is called to roll back any changes and free up the resources.

You should always *abort* a context if *any* error happens during its lifespan; otherwise the data in the system may be left in an inconsistent state. You can also *commit* a context, which means that any changes are written to the database, and the context is kept active for further use.



11.1.4. Email

Sending e-mails is pretty easy. Just use the configuration manager's *getEmail* method, set the arguments and recipients, and send.

The e-mail texts are stored in */dspace/config/emails*. They are processed by the standard *java.text.MessageFormat*. At the top of each e-mail are listed the appropriate arguments that should be filled out by the sender. Example usage is shown in the *org.dspace.core.Email* Javadoc API documentation.

11.1.5. LogManager

The log manager consists of a method that creates a standard log header, and returns it as a string suitable for logging. Note that this class does not actually write anything to the logs; the log header returned should be logged directly by the sender using an appropriate Log4J call, so that information about where the logging is taking place is also stored.

The level of logging can be configured on a per-package or per-class basis by editing */dspace/config/templates/log4j.properties* and then executing */dspace/bin/install-configs*. You will need to stop and restart Tomcat for the changes to take effect.

A typical log entry looks like this:

```
2002-11-11      08:11:32,903      INFO      org.dspace.app.webui.servlet.DSpaceServlet      @
anonymous:session_id=BD84E7C194C2CF4BD0EC3A6CAD0142BB:view_item:handle=1721.1/1686
```

This is breaks down like this:

Date and time, milliseconds	2002-11-11 08:11:32,903
Level (<i>FATAL</i> , <i>WARN</i> , <i>INFO</i> or <i>DEBUG</i>)	<i>INFO</i>
Java class	<i>org.dspace.app.webui.servlet.DSpaceServlet</i>
	@
User email or <i>anonymous</i>	<i>anonymous</i>
	:
Extra log info from context	<i>session_id=BD84E7C194C2CF4BD0EC3A6CAD0142BB</i>
	:
Action	<i>view_item</i>
	:
Extra info	<i>handle=1721.1/1686</i>

The above format allows the logs to be easily parsed and analysed. The */dspace/bin/log-reporter* script is a simple tool for analysing logs. Try:

```
/dspace/bin/log-reporter --help
```

It's a good idea to 'nice' this log reporter to avoid an impact on server performance.

11.1.6. Utils

Utils contains miscellaneous utility method that are required in a variety of places throughout the code, and thus have no particular 'home' in a subsystem.

11.2. Content Management API

The content management API package *org.dspace.content* contains Java classes for reading and manipulating content stored in the DSpace system. This is the API that components in the application layer will probably use most.



Classes corresponding to the main elements in the DSpace data model (*Community*, *Collection*, *Item*, *Bundle* and *Bitstream*) are sub-classes of the abstract class *DSpaceObject*. The *Item* object handles the Dublin Core metadata record.

Each class generally has one or more static *find* methods, which are used to instantiate content objects. Constructors do not have public access and are just used internally. The reasons for this are:

- "Constructing" an object may be misconstrued as the action of creating an object in the DSpace system, for example one might expect something like:

```
Context dsContent = new Context();
Item myItem = new Item(context, id)
```

to construct a brand new item in the system, rather than simply instantiating an in-memory instance of an object in the system.

- *find* methods may often be called with invalid IDs, and return *null* in such a case. A constructor would have to throw an exception in this case. A *null* return value from a static method can in general be dealt with more simply in code.
- If an instantiation representing the same underlying archival entity already exists, the *find* method can simply return that same instantiation to avoid multiple copies and any inconsistencies which might result. *Collection*, *Bundle* and *Bitstream* do not have *create* methods; rather, one has to create an object using the relevant method on the container. For example, to create a collection, one must invoke *createCollection* on the community that the collection is to appear in:

```
Context context = new Context();
Community existingCommunity = Community.find(context, 123);
Collection myNewCollection = existingCommunity.createCollection();
```

The primary reason for this is for determining authorization. In order to know whether an e-person may create an object, the system must know which container the object is to be added to. It makes no sense to create a collection outside of a community, and the authorization system does not have a policy for that.

Item_s are first created in the form of an implementation of *_InProgressSubmission*. An *InProgressSubmission* represents an item under construction; once it is complete, it is installed into the main archive and added to the relevant collection by the *InstallItem* class. The *org.dspace.content* package provides an implementation of *InProgressSubmission* called *WorkspaceItem*; this is a simple implementation that contains some fields used by the Web submission UI. The *org.dspace.workflow* also contains an implementation called *WorkflowItem* which represents a submission undergoing a workflow process.

In the previous chapter there is an overview of the item ingest process which should clarify the previous paragraph. Also see the section on the workflow system.

Community and *BitstreamFormat* do have static *create* methods; one must be a site administrator to have authorization to invoke these.

11.2.1. Other Classes

Classes whose name begins *DC* are for manipulating Dublin Core metadata, as explained below.

The *FormatIdentifier* class attempts to guess the bitstream format of a particular bitstream. Presently, it does this simply by looking at any file extension in the bitstream name and matching it up with the file extensions associated with bitstream formats. Hopefully this can be greatly improved in the future!

The *ItemIterator* class allows items to be retrieved from storage one at a time, and is returned by methods that may return a large number of items, more than would be desirable to have in memory at once.

The *ItemComparator* class is an implementation of the standard *java.util.Comparator* that can be used to compare and order items based on a particular Dublin Core metadata field.



11.2.2. Modifications

When creating, modifying or for whatever reason removing data with the content management API, it is important to know when changes happen in-memory, and when they occur in the physical DSpace storage.

Primarily, one should note that no change made using a particular *org.dspace.core.Context* object will actually be made in the underlying storage unless *complete* or *commit* is invoked on that *Context*. If anything should go wrong during an operation, the context should always be aborted by invoking *abort*, to ensure that no inconsistent state is written to the storage.

Additionally, some changes made to objects only happen in-memory. In these cases, invoking the *update* method lines up the in-memory changes to occur in storage when the *Context* is committed or completed. In general, methods that change any [meta]data field only make the change in-memory; methods that involve relationships with other objects in the system line up the changes to be committed with the context. See individual methods in the API Javadoc.

Some examples to illustrate this are shown below:

<pre>Context context = new Context(); Bitstream b = Bitstream.find(context, 1234); b.setName("newfile.txt"); b.update(); context.complete();</pre>	<p>Will change storage</p>
<pre>Context context = new Context(); Bitstream b = Bitstream.find(context, 1234); b.setName("newfile.txt"); b.update(); context.abort();</pre>	<p>Will not change storage (context aborted)</p>
<pre>Context context = new Context(); Bitstream b = Bitstream.find(context, 1234); b.setName("newfile.txt"); context.complete();</pre>	<p>The new name will not be stored since <i>update</i> was not invoked</p>
<pre>Context context = new Context(); Bitstream bs = Bitstream.find(context, 1234); Bundle bnd = Bundle.find(context, 5678); bnd.add(bs); context.complete();</pre>	<p>The bitstream will be included in the bundle, since <i>update</i> doesn't need to be called</p>

11.2.3. What's In Memory?

Instantiating some content objects also causes other content objects to be loaded into memory.

Instantiating a *Bitstream* object causes the appropriate *BitstreamFormat* object to be instantiated. Of course the *Bitstream* object does not load the underlying bits from the bitstream store into memory!

Instantiating a *Bundle* object causes the appropriate *Bitstream* objects (and hence *_BitstreamFormat_s*) to be instantiated.

Instantiating an *Item* object causes the appropriate *Bundle* objects (etc.) and hence *_BitstreamFormat_s* to be instantiated. All the Dublin Core metadata associated with that item are also loaded into memory.

The reasoning behind this is that for the vast majority of cases, anyone instantiating an item object is going to need information about the bundles and bitstreams within it, and this methodology allows that to be done in the most efficient way and is simple for the caller. For example, in the Web UI, the servlet (controller) needs to pass information about an item to the viewer (JSP), which needs to have all the information in-memory to display the item without further accesses to the database which may cause errors mid-display.



You do not need to worry about multiple in-memory instantiations of the same object, or any inconsistencies that may result; the *Context* object keeps a cache of the instantiated objects. The *find* methods of classes in *org.dspace.content* will use a cached object if one exists.

It may be that in enough cases this automatic instantiation of contained objects reduces performance in situations where it is important; if this proves to be true the API may be changed in the future to include a *loadContents* method or *somesuch*, or perhaps a Boolean parameter indicating what to do will be added to the *find* methods.

When a *Context* object is completed, aborted or garbage-collected, any objects instantiated using that context are invalidated and should not be used (in much the same way an AWT button is invalid if the window containing it is destroyed).

11.2.4. Dublin Core Metadata

The *DCValue* class is a simple container that represents a single Dublin Core element, optional qualifier, value and language. Note that since DSpace 1.4 the *MetadataValue* and associated classes are preferred (see Support for Other Metadata Schemas). The other classes starting with *DC* are utility classes for handling types of data in Dublin Core, such as people's names and dates. As supplied, the DSpace registry of elements and qualifiers corresponds to the <http://www.dublincore.org/documents/2002/09/24/library-application-profile/> for Dublin Core. It should be noted that these utility classes assume that the values will be in a certain syntax, which will be true for all data generated within the DSpace system, but since Dublin Core does not always define strict syntax, this may not be true for Dublin Core originating outside DSpace.

Below is the specific syntax that DSpace expects various fields to adhere to:

Element	Qualifier	Syntax	Helper Class
<i>date</i>	Any or unqualified	ISO 8601 in the UTC time zone, with either year, month, day, or second precision. Examples: <i>_2000</i> <i>2002-10</i> <i>2002-08-14</i> <i>1999-01-01T14:35:23Z</i> _	<i>DCDate</i>
<i>contributor</i>	Any or unqualified	In general last name, then a comma, then first names, then any additional information like "Jr.". If the contributor is an organization, then simply the name. Examples: <i>_Doe, John Smith,</i> <i>John Jr. van Dyke, Dick</i> <i>Massachusetts Institute of</i> <i>Technology</i> _	<i>DCPersonName</i>
<i>language</i>	<i>iso</i>	A two letter code taken ISO 639, followed optionally by a two letter country code taken from ISO 3166. Examples: <i>_en</i> <i>fr en_US</i> _	<i>DCLanguage</i>
<i>relation</i>	<i>ispartofseries</i>	The series name, following by a semicolon followed by the number in that series. Alternatively, just free text. <i>_MIT-TR;</i>	<i>DCSeriesNumber</i>



		1234 My Report Series; ABC-1234 NS1234 _	
--	--	---	--

11.2.5. Support for Other Metadata Schemas

To support additional metadata schemas a new set of metadata classes have been added. These are backwards compatible with the DC classes and should be used rather than the DC specific classes wherever possible. Note that hierarchical metadata schemas are not currently supported, only flat schemas (such as DC) are able to be defined.

The *MetadataField* class describes a metadata field by schema, element and optional qualifier. The value of a *MetadataField* is described by a *MetadataValue* which is roughly equivalent to the older *DCValue* class. Finally the *MetadataSchema* class is used to describe supported schemas. The DC schema is supported by default. Refer to the javadoc for method details.

11.2.6. Packager Plugins

The Packager plugins let you *ingest* a package to create a new DSpace Object, and *disseminate* a content Object as a package. A package is simply a data stream; its contents are defined by the packager plugin's implementation.

To ingest an object, which is currently only implemented for Items, the sequence of operations is:

1. Get an instance of the chosen *PackageIngester* plugin.
2. Locate a Collection in which to create the new Item.
3. Call its *ingest* method, and get back a *WorkspaceItem*.

The packager also takes a *PackageParameters* object, which is a property list of parameters specific to that packager which might be passed in from the user interface.

Here is an example package ingestion code fragment:

```
Collection collection = find target collection
InputStream source = ...;
PackageParameters params = ...;
String license = null;

PackageIngester sip = (PackageIngester) PluginManager
    .getNamedPlugin(PackageIngester.class, packageType);

WorkspaceItem wi = sip.ingest(context, collection, source, params, license);
```

Here is an example of a package dissemination:

```
OutputStream destination = ...;
PackageParameters params = ...;
DSpaceObject dso = ...;

PackageIngester dip = (PackageDisseminator) PluginManager
    .getNamedPlugin(PackageDisseminator.class, packageType);

dip.disseminate(context, dso, params, destination);
```

11.3. Plugin Manager

The *PluginManager* is a very simple component container. It creates and organizes components (plugins), and helps select a plugin in the cases where there are many possible choices. It also gives some limited control over the lifecycle of a plugin.



11.3.1. Concepts

The following terms are important in understanding the rest of this section:

- **Plugin Interface** A Java interface, the defining characteristic of a plugin. The consumer of a plugin asks for its plugin by interface.
- **Plugin** a.k.a. Component, this is an instance of a class that implements a certain interface. It is interchangeable with other implementations, so that any of them may be "plugged in", hence the name. A Plugin is an instance of any class that implements the plugin interface.
- **Implementation class** The actual class of a plugin. It may implement several plugin interfaces, but must implement at least one.
- **Name** Plugin implementations can be distinguished from each other by name, a short String meant to symbolically represent the implementation class. They are called "named plugins". Plugins only need to be named when the caller has to make an active choice between them.
- **SelfNamedPlugin class** Plugins that extend the *SelfNamedPlugin* class can take advantage of additional features of the Plugin Manager. Any class can be managed as a plugin, so it is not necessary, just possible.
- **Reusable** Reusable plugins are only instantiated once, and the Plugin Manager returns the same (cached) instance whenever that same plugin is requested again. This behavior can be turned off if desired.

11.3.2. Using the Plugin Manager

Types of Plugin

The Plugin Manager supports three different patterns of usage:

1. **Singleton Plugins** There is only one implementation class for the plugin. It is indicated in the configuration. This type of plugin chooses an implementation of a service, for the entire system, at configuration time. Your application just fetches the plugin for that interface and gets the configured-in choice. See the `getSinglePlugin()` method.
2. **Sequence Plugins** You need a sequence or series of plugins, to implement a mechanism like Stackable Authentication or a pipeline, where each plugin is called in order to contribute its implementation of a process to the whole. The Plugin Manager supports this by letting you configure a sequence of plugins for a given interface. See the `getPluginSequence()` method.
3. **Named Plugins** Use a named plugin when the application has to choose one plugin implementation out of many available ones. Each implementation is bound to one or more names (symbolic identifiers) in the configuration. The name is just a string to be associated with the combination of implementation class and interface. It may contain any characters except for comma (,) and equals (=). It may contain embedded spaces. Comma is a special character used to separate names in the configuration entry. Names must be unique within an interface: No plugin classes implementing the same interface may have the same name. Think of plugin names as a controlled vocabulary – for a given plugin interface, there is a set of names for which plugins can be found. The designer of a Named Plugin interface is responsible for deciding what the name means and how to derive it; for example, names of metadata crosswalk plugins may describe the target metadata format. See the `getNamedPlugin()` method and the `getPluginNames()` methods.

Self-Named Plugins

Named plugins can get their names either from the configuration or, for a variant called self-named plugins, from within the plugin itself.

Self-named plugins are necessary because one plugin implementation can be configured itself to take on many "personalities", each of which deserves its own plugin name. It is already managing its own config-



uration for each of these personalities, so it makes sense to allow it to export them to the Plugin Manager rather than expecting the plugin configuration to be kept in sync with its own configuration.

An example helps clarify the point: There is a named plugin that does crosswalks, call it *CrosswalkPlugin*. It has several implementations that crosswalk some kind of metadata. Now we add a new plugin which uses XSL stylesheet transformation (XSLT) to crosswalk many types of metadata – so the single plugin can act like many different plugins, depending on which stylesheet it employs.

This XSLT-crosswalk plugin has its own configuration that maps a Plugin Name to a stylesheet – it has to, since of course the Plugin Manager doesn't know anything about stylesheets. It becomes a self-named plugin, so that it reads its configuration data, gets the list of names to which it can respond, and passes those on to the Plugin Manager.

When the Plugin Manager creates an instance of the XSLT-crosswalk, it records the Plugin Name that was responsible for that instance. The plugin can look at that Name later in order to configure itself correctly for the Name that created it. This mechanism is all part of the *SelfNamedPlugin* class which is part of any self-named plugin.

Obtaining a Plugin Instance

The most common thing you will do with the Plugin Manager is obtain an instance of a plugin. To request a plugin, you must always specify the plugin interface you want. You will also supply a name when asking for a named plugin.

A sequence plugin is returned as an array of *_Object_s* since it is actually an ordered list of plugins.

See the *getSinglePlugin()*, *getPluginSequence()*, *getNamedPlugin()* methods.

Lifecycle Management

When *PluginManager* fulfills a request for a plugin, it checks whether the implementation class is reusable; if so, it creates one instance of that class and returns it for every subsequent request for that interface and name. If it is not reusable, a new instance is always created.

For reasons that will become clear later, the manager actually caches a separate instance of an implementation class for each name under which it can be requested.

You can ask the *PluginManager* to forget about (decache) a plugin instance, by releasing it. See the *PluginManager.releasePlugin()* method. The manager will drop its reference to the plugin so the garbage collector can reclaim it. The next time that plugin/name combination is requested, it will create a new instance.

Getting Meta-Information

The *PluginManager* can list all the names of the Named Plugins which implement an interface. You may need this, for example, to implement a menu in a user interface that presents a choice among all possible plugins. See the *getPluginNames()* method.

Note that it only returns the plugin name, so if you need a more sophisticated or meaningful "label" (i.e. a key into the I18N message catalog) then you should add a method to the plugin itself to return that.

11.3.3. Implementation

Note: The *PluginManager* refers to interfaces and classes internally only by their names whenever possible, to avoid loading classes until absolutely necessary (i.e. to create an instance). As you'll see below, self-named classes still have to be loaded to query them for names, but for the most part it can avoid loading classes. This saves a lot of time at start-up and keeps the JVM memory footprint down, too. As the Plugin Manager gets used for more classes, this will become a greater concern.



The only downside of "on-demand" loading is that errors in the configuration don't get discovered right away. The solution is to call the `checkConfiguration()` method after making any changes to the configuration.

PluginManager Class

The `PluginManager` class is your main interface to the Plugin Manager. It behaves like a factory class that never gets instantiated, so its public methods are static.

Here are the public methods, followed by explanations:

- `static Object getSinglePlugin(Class intface)`
throws `PluginConfigurationException`;

Returns an instance of the singleton (single) plugin implementing the given interface. There must be exactly one single plugin configured for this interface, otherwise the `PluginConfigurationException` is thrown. Note that this is the only "get plugin" method which throws an exception. It is typically used at initialization time to set up a permanent part of the system so any failure is fatal. See the `plugin.single` configuration key for configuration details.

- `static Object[] getPluginSequence(Class intface)`; Returns instances of all plugins that implement the interface `intface`, in an `Array`. Returns an empty array if there are no matching plugins. The order of the plugins in the array is the same as their class names in the configuration's value field. See the `plugin.sequence` configuration key for configuration details.
- `static Object getNamedPlugin(Class intface, String name)`; Returns an instance of a plugin that implements the interface `intface` and is bound to a name matching name. If there is no matching plugin, it returns null. The names are matched by `String.equals()`. See the `plugin.named` and `plugin.selfnamed` configuration keys for configuration details.
- `static void releasePlugin(Object plugin)`; Tells the Plugin Manager to let go of any references to a reusable plugin, to prevent it from being given out again and to allow the object to be garbage-collected. Call this when a plugin instance must be taken out of circulation.
- `static String[] getAllPluginNames(Class intface)`; Returns all of the names under which a named plugin implementing the interface `intface` can be requested (with `getNamedPlugin()`). The array is empty if there are no matches. Use this to populate a menu of plugins for interactive selection, or to document what the possible choices are. The names are NOT returned in any predictable order, so you may wish to sort them first. Note: Since a plugin may be bound to more than one name, the list of names this returns does not represent the list of plugins. To get the list of unique implementation classes corresponding to the names, you might have to eliminate duplicates (i.e. create a `Set` of classes).
- `static void checkConfiguration()`; Validates the keys in the DSpace `ConfigurationManager` pertaining to the Plugin Manager and reports any errors by logging them. This is intended to be used interactively by a DSpace administrator, to check the configuration file after modifying it. See the section about validating configuration for details.

SelfNamedPlugin Class

A named plugin implementation must extend this class if it wants to supply its own Plugin Name(s). See Self-Named Plugins for why this is sometimes necessary.

```
abstract class SelfNamedPlugin
{
    // Your class must override this:
    // Return all names by which this plugin should be known.
    public static String[] getPluginNames();

    // Returns the name under which this instance was created.
    // This is implemented by SelfNamedPlugin and should NOT be
    overridden.
}
```



```
public String getPluginInstanceName();
}
```

Errors and Exceptions

```
public class PluginConfigurationError extends Error
{
    public PluginConfigurationError(String message);
}
```

An error of this type means the caller asked for a single plugin, but either there was no single plugin configured matching that interface, or there was more than one. Either case causes a fatal configuration error.

```
public class PluginInstantiationException extends RuntimeException
{
    public PluginInstantiationException(String msg, Throwable cause)
}
```

This exception indicates a fatal error when instantiating a plugin class. It should only be thrown when something unexpected happens in the course of instantiating a plugin, e.g. an access error, class not found, etc. Simply not finding a class in the configuration is not an exception.

This is a *RuntimeException* so it doesn't have to be declared, and can be passed all the way up to a generalized fatal exception handler.

11.3.4. Configuring Plugins

All of the Plugin Manager's configuration comes from the DSpace Configuration Manager, which is a Java Properties map. You can configure these characteristics of each plugin:

1. **Interface:** Classname of the Java interface which defines the plugin, including package name. e.g. *org.dspace.app.mediafilter.FormatFilter*
2. **Implementation Class:** Classname of the implementation class, including package. e.g. *org.dspace.app.mediafilter.PDFFilter*
3. **Names:** (Named plugins only) There are two ways to bind names to plugins: listing them in the value of a *plugin.named.interface* key, or configuring a class in *plugin.selfnamed.interface* which extends the *SelfNamedPlugin* class.
4. **Reusable option:** (Optional) This is declared in a *plugin.reusable* configuration line. Plugins are reusable by default, so you only need to configure the non-reusable ones. Configuring Singleton (Single) Plugins

This entry configures a Single Plugin for use with `getSinglePlugin()`:

```
plugin.single.interface = classname
```

For example, this configures the class *org.dspace.checker.SimpleDispatcher* as the plugin for interface *org.dspace.checker.BitstreamDispatcher*:

```
plugin.single.org.dspace.checker.BitstreamDispatcher=org.dspace.checker.SimpleDispatcher
```

Configuring Sequence of Plugins

This kind of configuration entry defines a Sequence Plugin, which is bound to a sequence of implementation classes. The key identifies the interface, and the value is a comma-separated list of classnames:

```
plugin.sequence.interface = classname, ...
```

The plugins are returned by `getPluginSequence()` in the same order as their classes are listed in the configuration value.



For example, this entry configures Stackable Authentication with three implementation classes:

```
plugin.sequence.org.dspace.eperson.AuthenticationMethod = \
    org.dspace.eperson.X509Authentication, \
    org.dspace.eperson.PasswordAuthentication, \
    edu.mit.dspace.MITSpecialGroup
```

Configuring Named Plugins

There are two ways of configuring named plugins:

1. **Plugins Named in the Configuration** A named plugin which gets its name(s) from the configuration is listed in this kind of entry: `_plugin.named.interface = classname = name [, name..] [classname = name..]` The syntax of the configuration value is: classname, followed by an equal-sign and then at least one plugin name. Bind more names to the same implementation class by adding them here, separated by commas. Names may include any character other than comma (,) and equal-sign (=). For example, this entry creates one plugin with the names GIF, JPEG, and image/png, and another with the name TeX:

```
plugin.named.org.dspace.app.mediafilter.MediaFilter = \
    org.dspace.app.mediafilter.JPEGFilter = GIF, JPEG, image/png \
    org.dspace.app.mediafilter.TeXFilter = TeX
```

This example shows a plugin name with an embedded whitespace character. Since comma (,) is the separator character between plugin names, spaces are legal (between words of a name; leading and trailing spaces are ignored). This plugin is bound to the names "Adobe PDF", "PDF", and "Portable Document Format".

```
plugin.named.org.dspace.app.mediafilter.MediaFilter = \
    org.dspace.app.mediafilter.TeXFilter = TeX \
    org.dspace.app.mediafilter.PDFFilter = Adobe PDF, PDF, Portable Document Format
```

NOTE: Since there can only be one key with `plugin.named.` followed by the interface name in the configuration, all of the plugin implementations must be configured in that entry.

2. **Self-Named Plugins** Since a self-named plugin supplies its own names through a static method call, the configuration only has to include its interface and classname: `plugin.selfnamed.interface = classname [, classname..]` The following example first demonstrates how the plugin class, `_XsltDisseminationCrosswalk` is configured to implement its own names "MODS" and "DublinCore". These come from the keys starting with `crosswalk.dissemination.stylesheet.`. The value is a stylesheet file. The class is then configured as a self-named plugin:

```
crosswalk.dissemination.stylesheet.DublinCore = xwalk/TESTDIM-2-DC_copy.xml
crosswalk.dissemination.stylesheet.MODS = xwalk/mods.xml

plugin.selfnamed.crosswalk.org.dspace.content.metadata.DisseminationCrosswalk = \
    org.dspace.content.metadata.MODSDisseminationCrosswalk, \
    org.dspace.content.metadata.XsltDisseminationCrosswalk
```

NOTE: Since there can only be one key with `plugin.selfnamed.` followed by the interface name in the configuration, all of the plugin implementations must be configured in that entry. The `MODSDisseminationCrosswalk` class is only shown to illustrate this point.

Configuring the Reusable Status of a Plugin

Plugins are assumed to be reusable by default, so you only need to configure the ones which you would prefer not to be reusable. The format is as follows:

```
plugin.reusable.classname = ( true | false )
```

For example, this marks the PDF plugin from the example above as non-reusable:



plugin.reusable.org.dspace.app.mediafilter.PDFFilter = false

11.3.5. Validating the Configuration

The Plugin Manager is very sensitive to mistakes in the DSpace configuration. Subtle errors can have unexpected consequences that are hard to detect: for example, if there are two "plugin.single" entries for the same interface, one of them will be silently ignored.

To validate the Plugin Manager configuration, call the *PluginManager.checkConfiguration()* method. It looks for the following mistakes:

- Any duplicate keys starting with "plugin."
- Keys starting *plugin.single*, *plugin.sequence*, *plugin.named*, and *plugin.selfnamed* that don't include a valid interface.
- Classnames in the configuration values that don't exist, or don't implement the plugin interface in the key.
- Classes declared in *plugin.selfnamed* lines that don't extend the *SelfNamedPlugin* class.
- Any name collisions among named plugins for a given interface.
- Named plugin configuration entries without any names.
- Classnames mentioned in *plugin.reusable* keys must exist and have been configured as a plugin implementation class.

The *PluginManager* class also has a *main()* method which simply runs *checkConfiguration()*, so you can invoke it from the command line to test the validity of plugin configuration changes.

Eventually, someone should develop a general configuration-file sanity checker for DSpace, which would just call *PluginManager.checkConfiguration()*.

11.3.6. Use Cases

Here are some usage examples to illustrate how the Plugin Manager works.

Managing the MediaFilter plugins transparently

The existing DSpace 1.3 MediaFilterManager implementation has been largely replaced by the Plugin Manager. The MediaFilter classes become plugins named in the configuration. Refer to the configuration guide for further details.

A Singleton Plugin

This shows how to configure and access a single anonymous plugin, such as the BitstreamDispatcher plugin:

Configuration:

plugin.single.org.dspace.checker.BitstreamDispatcher=org.dspace.checker.SimpleDispatcher

The following code fragment shows how dispatcher, the service object, is initialized and used:

```
BitstreamDispatcher dispatcher =
    (BitstreamDispatcher)PluginManager.getSinglePlugin(BitstreamDispatcher
        .class);

int id = dispatcher.next();

while (id != BitstreamDispatcher.SENTINEL)
{
    /*
```



```

        do some processing here
    */
    id = dispatcher.next();
}

```

Plugin that Names Itself

This crosswalk plugin acts like many different plugins since it is configured with different XSL translation stylesheets. Since it already gets each of its stylesheets out of the DSpace configuration, it makes sense to have the plugin give PluginManager the names to which it answers instead of forcing someone to configure those names in two places (and try to keep them synchronized).

NOTE: Remember how *getPlugin()* caches a separate instance of an implementation class for every name bound to it? This is why: the instance can look at the name under which it was invoked and configure itself specifically for that name. Since the instance for each name might be different, the Plugin Manager has to cache a separate instance for each name.

Here is the configuration file listing both the plugin's own configuration and the *PluginManager* config line:

```

crosswalk.dissemination.stylesheet.DublinCore = xwalk/TESTDIM-2-DC_copy.xml
crosswalk.dissemination.stylesheet.MODS = xwalk/mods.xml

plugin.selfnamed.org.dspace.content.metadata.DisseminationCrosswalk = \
  org.dspace.content.metadata.XsltDisseminationCrosswalk

```

This look into the implementation shows how it finds configuration entries to populate the array of plugin names returned by the *getPluginNames()* method. Also note, in the *getStylesheet()* method, how it uses the plugin name that created the current instance (returned by *getPluginInstanceName()*) to find the correct stylesheet.

```

public class XsltDisseminationCrosswalk extends SelfNamedPlugin
{
    ....
    private final String prefix =
    "crosswalk.dissemination.stylesheet.";
    ....
    public static String[] getPluginNames()
    {
        List aliasList = new ArrayList();
        Enumeration pe = ConfigurationManager.propertyNames();

        while (pe.hasMoreElements())
        {
            String key = (String)pe.nextElement();
            if (key.startsWith(prefix))
                aliasList.add(key.substring(prefix.length()));
        }
        return (String[])aliasList.toArray(new
        String[aliasList.size()]);
    }

    // get the crosswalk stylesheet for an instance of the plugin:
    private String getStylesheet()
    {
        return ConfigurationManager.getProperty(prefix +
        getPluginInstanceName());
    }
}

```

Stackable Authentication

The Stackable Authentication mechanism needs to know all of the plugins configured for the interface, in the order of configuration, since order is significant. It gets a Sequence Plugin from the Plugin Manager. Refer to the Configuration Section on Stackable Authentication for further details.



11.4. Workflow System

The primary classes are:

<i>org.dspace.content.WorkspaceItem</i>	contains an Item before it enters a workflow
<i>org.dspace.workflow.WorkflowItem</i>	contains an Item while in a workflow
<i>org.dspace.workflow.WorkflowManager</i>	responds to events, manages the WorkflowItem states
<i>org.dspace.content.Collection</i>	contains List of defined workflow steps
<i>org.dspace.eperson.Group</i>	people who can perform workflow tasks are defined in EPerson Groups
<i>org.dspace.core.Email</i>	used to email messages to Group members and submitters

The workflow system models the states of an Item in a state machine with 5 states (SUBMIT, STEP_1, STEP_2, STEP_3, ARCHIVE.) These are the three optional steps where the item can be viewed and corrected by different groups of people. Actually, it's more like 8 states, with STEP_1_POOL, STEP_2_POOL, and STEP_3_POOL. These pooled states are when items are waiting to enter the primary states.

The WorkflowManager is invoked by events. While an Item is being submitted, it is held by a WorkspaceItem. Calling the start() method in the WorkflowManager converts a WorkspaceItem to a WorkflowItem, and begins processing the WorkflowItem's state. Since all three steps of the workflow are optional, if no steps are defined, then the Item is simply archived.

Workflows are set per Collection, and steps are defined by creating corresponding entries in the List named workflowGroup. If you wish the workflow to have a step 1, use the administration tools for Collections to create a workflow Group with members who you want to be able to view and approve the Item, and the workflowGroup[0] becomes set with the ID of that Group.

If a step is defined in a Collection's workflow, then the WorkflowItem's state is set to that step_POOL. This pooled state is the WorkflowItem waiting for an EPerson in that group to claim the step's task for that WorkflowItem. The WorkflowManager emails the members of that Group notifying them that there is a task to be performed (the text is defined in config/emails,) and when an EPerson goes to their 'My DSpace' page to claim the task, the WorkflowManager is invoked with a claim event, and the WorkflowItem's state advances from STEP_x_POOL to STEP_x (where x is the corresponding step.) The EPerson can also generate an 'unclaim' event, returning the WorkflowItem to the STEP_x_POOL.

Other events the WorkflowManager handles are advance(), which advances the WorkflowItem to the next state. If there are no further states, then the WorkflowItem is removed, and the Item is then archived. An EPerson performing one of the tasks can reject the Item, which stops the workflow, rebuilds the WorkspaceItem for it and sends a rejection note to the submitter. More drastically, an abort() event is generated by the admin tools to cancel a workflow outright.

11.5. Administration Toolkit

The *org.dspace.administer* package contains some classes for administering a DSpace system that are not generally needed by most applications.

The *CreateAdministrator* class is a simple command-line tool, executed via */dspace/bin/create-administrator*, that creates an administrator e-person with information entered from standard input. This is generally used only once when a DSpace system is initially installed, to create an initial administrator who can then use the Web administration UI to further set up the system. This script does not check for authorization, since it is typically run before there are any e-people to authorize! Since it must be run as a command-line tool on the server machine, generally this shouldn't cause a problem. A possibility is to have the script only operate when there are no e-people in the system already, though in general, someone with access to command-line scripts on your server is probably in a position to do what they want anyway!



The *DCType* class is similar to the *org.dspace.content.BitstreamFormat* class. It represents an entry in the Dublin Core type registry, that is, a particular element and qualifier, or unqualified element. It is in the *administer* package because it is only generally required when manipulating the registry itself. Elements and qualifiers are specified as literals in *org.dspace.content.Item* methods and the *org.dspace.content.DCValue* class. Only administrators may modify the Dublin Core type registry.

The *org.dspace.administer.RegistryLoader* class contains methods for initialising the Dublin Core type registry and bitstream format registry with entries in an XML file. Typically this is executed via the command line during the build process (see *build.xml* in the source.) To see examples of the XML formats, see the files in *config/registries* in the source directory. There is no XML schema, they aren't validated strictly when loaded in.

11.6. E-person/Group Manager

DSpace keeps track of registered users with the *org.dspace.eperson.EPerson* class. The class has methods to create and manipulate an *EPerson* such as get and set methods for first and last names, email, and password. (Actually, there is no *getPassword()* method--an MD5 hash of the password is stored, and can only be verified with the *checkPassword()* method.) There are find methods to find an *EPerson* by email (which is assumed to be unique,) or to find all *EPeople* in the system.

The *EPerson* object should probably be reworked to allow for easy expansion; the current *EPerson* object tracks pretty much only what MIT was interested in tracking - first and last names, email, phone. The access methods are hardcoded and should probably be replaced with methods to access arbitrary name/value pairs for institutions that wish to customize what *EPerson* information is stored.

Groups are simply lists of *EPerson* objects. Other than membership, *Group* objects have only one other attribute: a name. Group names must be unique, so we have adopted naming conventions where the role of the group is its name, such as *COLLECTION_100_ADD*. Groups add and remove *EPerson* objects with *addMember()* and *removeMember()* methods. One important thing to know about groups is that they store their membership in memory until the *update()* method is called - so when modifying a group's membership don't forget to invoke *update()* or your changes will be lost! Since group membership is used heavily by the authorization system a fast *isMember()* method is also provided.

Another kind of Group is also implemented in DSpace--special Groups. The Context object for each session carries around a List of Group IDs that the user is also a member of--currently the MITUser Group ID is added to the list of a user's special groups if certain IP address or certificate criteria are met.

11.7. Authorization

The primary classes are:

<i>org.dspace.authorize.AuthorizeManager</i>	does all authorization, checking policies against Groups
<i>org.dspace.authorize.ResourcePolicy</i>	defines all allowable actions for an object
<i>org.dspace.eperson.Group</i>	all policies are defined in terms of <i>EPerson</i> Groups

The authorization system is based on the classic 'police state' model of security; no action is allowed unless it is expressed in a policy. The policies are attached to resources (hence the name *ResourcePolicy*.) and detail who can perform that action. The resource can be any of the DSpace object types, listed in *org.dspace.core.Constants* (*BITSTREAM*, *ITEM*, *COLLECTION*, etc.) The 'who' is made up of *EPerson* groups. The actions are also in *Constants.java* (*READ*, *WRITE*, *ADD*, etc.) The only non-obvious actions are *ADD* and *REMOVE*, which are authorizations for container objects. To be able to create an *Item*, you must have *ADD* permission in a *Collection*, which contains *Items*. (*Communities*, *Collections*, *Items*, and *Bundles* are all container objects.)

Currently most of the read policy checking is done with items--communities and collections are assumed to be openly readable, but items and their bitstreams are checked. Separate policy checks for items and their



bitstreams enables policies that allow publicly readable items, but parts of their content may be restricted to certain groups.

The *AuthorizeManager* class'

authorizeAction(Context, object, action) is the primary source of all authorization in the system. It gets a list of all of the *ResourcePolicies* in the system that match the object and action. It then iterates through the policies, extracting the *EPerson Group* from each policy, and checks to see if the *EPersonID* from the *Context* is a member of any of those groups. If all of the policies are queried and no permission is found, then an *AuthorizeException* is thrown. An *authorizeAction()* method is also supplied that returns a boolean for applications that require higher performance.

ResourcePolicies are very simple, and there are quite a lot of them. Each can only list a single group, a single action, and a single object. So each object will likely have several policies, and if multiple groups share permissions for actions on an object, each group will get its own policy. (It's a good thing they're small.)

11.7.1. Special Groups

All users are assumed to be part of the public group (ID=0.) *DSpace* admins (ID=1) are automatically part of all groups, much like super-users in the Unix OS. The *Context* object also carries around a List of special groups, which are also first checked for membership. These special groups are used at MIT to indicate membership in the MIT community, something that is very difficult to enumerate in the database! When a user logs in with an MIT certificate or with an MIT IP address, the login code adds this MIT user group to the user's *Context*.

11.7.2. Miscellaneous Authorization Notes

Where do items get their read policies? From the their collection's read policy. There once was a separate item read default policy in each collection, and perhaps there will be again since it appears that administrators are notoriously bad at defining collection's read policies. There is also code in place to enable policies that are timed--have a start and end date. However, the admin tools to enable these sorts of policies have not been written.

11.8. Handle Manager/Handle Plugin

The *org.dspace.handle* package contains two classes; *HandleManager* is used to create and look up Handles, and *HandlePlugin* is used to expose and resolve *DSpace* Handles for the outside world via the CNRI Handle Server code.

Handles are stored internally in the *handle* database table in the form:

```
1721.123/4567
```

Typically when they are used outside of the system they are displayed in either URI or "URL proxy" forms:

```
hdl:1721.123/4567
http://hdl.handle.net/1721.123/4567
```

It is the responsibility of the caller to extract the basic form from whichever displayed form is used.

The *handle* table maps these Handles to resource type/resource ID pairs, where resource type is a value from *org.dspace.core.Constants* and resource ID is the internal identifier (database primary key) of the object. This allows Handles to be assigned to any type of object in the system, though as explained in the functional overview, only communities, collections and items are presently assigned Handles.

HandleManager contains static methods for:

- Creating a Handle
- Finding the Handle for a *DSpaceObject*, though this is usually only invoked by the object itself, since *DSpaceObject* has a *getHandle* method



- Retrieving the *DSpaceObject* identified by a particular Handle
- Obtaining displayable forms of the Handle (URI or "proxy URL").
HandlePlugin is a simple implementation of the Handle Server's *net.handle.hdllib.HandleStorage* interface. It only implements the basic Handle retrieval methods, which get information from the *handle* database table. The CNRI Handle Server is configured to use this plug-in via its *config.dct* file.

Note that since the Handle server runs as a separate JVM to the DSpace Web applications, it uses a separate 'Log4J' configuration, since Log4J does not support multiple JVMs using the same daily rolling logs. This alternative configuration is held as a template in */dspace/config/templates/log4j-handle-plugin.properties*, written to */dspace/config/log4j-handle-plugin.properties* by the *install-configs* script. The */dspace/bin/start-handle-server* script passes in the appropriate command line parameters so that the Handle server uses this configuration.

11.9. Search

DSpace's search code is a simple API which currently wraps the Lucene search engine. The first half of the search task is indexing, and *org.dspace.search.DSIndexer* is the indexing class, which contains *indexContent()* which if passed an *Item*, *Community*, or *Collection*, will add that content's fields to the index. The methods *unIndexContent()* and *reIndexContent()* remove and update content's index information. The *DSIndexer* class also has a *main()* method which will rebuild the index completely. This can be invoked by the *dspace/bin/index-init* (complete rebuild) or *dspace/bin/index-update* (update) script. The intent was for the *main()* method to be invoked on a regular basis to avoid index corruption, but we have had no problem with that so far.

Which fields are indexed by *DSIndexer*? These fields are defined in *dspace.cfg* in the section "Fields to index for search" as name-value-pairs. The name must be unique in the form *search.index.i* (*i* is an arbitrary positive number). The value on the right side has a unique value again, which can be referenced in search-form (e.g. title, author). Then comes the metadata element which is indexed. '*' is a wildcard which includes all subelements. For example:

```
search.index.4 = keyword:dc.subject.*
```

tells the indexer to create a keyword index containing all *dc.subject* element values. Since the wildcard ('*') character was used in place of a qualifier, all subject metadata fields will be indexed (e.g. *dc.subject.other*, *dc.subject.lcsh*, etc)

By default, the fields shown in the *Indexed Fields* section below are indexed. These are hardcoded in the *DSIndexer* class. If any *search.index.i* items are specified in *dspace.cfg* these are used rather than these hardcoded fields.

The query class *DSQuery* contains the three flavors of *doQuery()* methods--one searches the DSpace site, and the other two restrict searches to Collections and Communities. The results from a query are returned as three lists of handles; each list represents a type of result. One list is a list of Items with matches, and the other two are Collections and Communities that match. This separation allows the UI to handle the types of results gracefully without resolving all of the handles first to see what kind of content the handle points to. The *DSQuery* class also has a *main()* method for debugging via command-line searches.

11.9.1. Current Lucene Implementation

Currently we have our own Analyzer and Tokenizer classes (*DSAnalyzer* and *DSTokenizer*) to customize our indexing. They invoke the stemming and stop word features within Lucene. We create an *IndexReader* for each query, which we now realize isn't the most efficient use of resources - we seem to run out of filehandles on really heavy loads. (A wildcard query can open many filehandles!) Since Lucene is thread-safe, a better future implementation would be to have a single Lucene *IndexReader* shared by all queries, and then is invalidated and re-opened when the index changes. Future API growth could include relevance scores (Lucene generates them, but we ignore them,) and abstractions for more advanced search concepts such as booleans.



11.9.2. Indexed Fields

The *DSIndexer* class shipped with DSpace indexes the Dublin Core metadata in the following way:

Search Field	Taken from Dublin Core Fields
Authors	<i>contributor.creator.description.statementsofresponsibility</i>
Titles	<i>title.*</i>
Keywords	<i>subject.*</i>
Abstracts	<i>description.abstractdescription.tableofcontents</i>
Series	<i>relation.ispartofseries</i>
MIME types	<i>format.mimetype</i>
Sponsors	<i>description.sponsorship</i>
Identifiers	<i>identifier.*</i>

11.9.3. Harvesting API

The *org.dspace.search* package also provides a 'harvesting' API. This allows callers to extract information about items modified within a particular timeframe, and within a particular scope (all of DSpace, or a community or collection.) Currently this is used by the Open Archives Initiative metadata harvesting protocol application, and the e-mail subscription code.

The *Harvest.harvest* is invoked with the required scope and start and end dates. Either date can be omitted. The dates should be in the ISO8601, UTC time zone format used elsewhere in the DSpace system.

HarvestedItemInfo objects are returned. These objects are simple containers with basic information about the items falling within the given scope and date range. Depending on parameters passed to the *harvest* method, the *containers* and *item* fields may have been filled out with the IDs of communities and collections containing an item, and the corresponding *Item* object respectively. Electing not to have these fields filled out means the harvest operation executes considerable faster.

In case it is required, *Harvest* also offers a method for creating a single *HarvestedItemInfo* object, which might make things easier for the caller.

11.10. Browse API

The browse API maintains indices of dates, authors, titles and subjects, and allows callers to extract parts of these:

- ***Title: Values of the Dublin Core element *title** (unqualified) are indexed. These are sorted in a case-insensitive fashion, with any leading article removed. For example: *The DSpace System* Appears under 'D' rather than 'T'.
- ***Author: Values of the *contributor** (any qualifier or unqualified) element are indexed. Since *contributor* values typically are in the form 'last name, first name', a simple case-insensitive alphanumeric sort is used which orders authors in last name order. Note that this is an index of *authors*, and not *items by author*. If four items have the same author, that author will appear in the index only once. Hence, the index of authors may be greater or smaller than the index of titles; items often have more than one author, though the same author may have authored several items. The author indexing in the browse API does have limitations:
 - Ideally, a name that appears as an author for more than one item would appear in the author index only once. For example, 'Doe, John' may be the author of tens of items. However, in practice, author's names often appear in slightly differently forms, for example:

Doe, John



```
Doe, John Stewart
Doe, John S.
```

Currently, the above three names would all appear as separate entries in the author index even though they may refer to the same author. In order for an author of several papers to be correctly appear once in the index, each item must specify *exactly* the same form of their name, which doesn't always happen in practice.

- Another issue is that two authors may have the same name, even within a single institution. If this is the case they may appear as one author in the index. These issues are typically resolved in libraries with *authority control records*, in which are kept a 'preferred' form of the author's name, with extra information (such as date of birth/death) in order to distinguish between authors of the same name. Maintaining such records is a huge task with many issues, particularly when metadata is received from faculty directly rather than trained library cataloguers. For these reasons, DSpace does not yet feature 'authority control' functionality.
- ***Date of Issue: Items are indexed by date of issue. This may be different from the date that an item appeared in DSpace; many items may have been originally published elsewhere beforehand. The Dublin Core field used is *date.issued.** The ordering of this index may be reversed so 'earliest first' and 'most recent first' orderings are possible. Note that the index is of *items by date*, as opposed to an index of *dates*. If 30 items have the same issue date (say 2002), then those 30 items all appear in the index adjacent to each other, as opposed to a single 2002 entry. Since dates in DSpace Dublin Core are in ISO8601, all in the UTC time zone, a simple alphanumeric sort is sufficient to sort by date, including dealing with varying granularities of date reasonably. For example:

```
2001-12-10
2002
2002-04
2002-04-05
2002-04-09T15:34:12Z
2002-04-09T19:21:12Z
2002-04-10
```

- ***Date Accessioned: In order to determine which items most recently appeared, rather than using the date of issue, an item's accession date is used. This is the Dublin Core field *date.accessioned.** In other aspects this index is identical to the date of issue index.
- ***Items by a Particular Author*:** The browse API can perform is to extract items by a particular author. They do not have to be primary author of an item for that item to be extracted. You can specify a scope, too; that is, you can ask for items by author X in collection Y, for example. This particular flavour of browse is slightly simpler than the others. You cannot presently specify a particular subset of results to be returned. The API call will simply return all of the items by a particular author within a certain scope. Note that the author of the item must *exactly* match the author passed in to the API; see the explanation about the caveats of the author index browsing to see why this is the case.
- ***Subject: Values of the Dublin Core element *subject (both unqualified and with any qualifier) are indexed. These are sorted in a case-insensitive fashion. Using the API**

The API is generally invoked by creating a *BrowseScope* object, and setting the parameters for which particular part of an index you want to extract. This is then passed to the relevant *Browse* method call, which returns a *BrowseInfo* object which contains the results of the operation. The parameters set in the *BrowseScope* object are:

- How many entries from the index you want
- Whether you only want entries from a particular community or collection, or from the whole of DSpace
- Which part of the index to start from (called the *focus* of the browse). If you don't specify this, the start of the index is used
- How many entries to include before the *focus* entry



To illustrate, here is an example:

- We want 7 entries in total
- We want entries from collection *x*
- We want the focus to be 'Really'
- We want 2 entries included before the focus.

The results of invoking *Browse.getItemsByTitle* with the above parameters might look like this:

```
Rabble-Rousing Rabbis From Sardinia
Reality TV: Love It or Hate It?
FOCUS> The Really Exciting Research Video
Recreational Housework Addicts: Please Visit My House
Regional Television Variation Studies
Revenue Streams
Ridiculous Example Titles: I'm Out of Ideas
```

Note that in the case of title and date browses, *Item* objects are returned as opposed to actual titles. In these cases, you can specify the 'focus' to be a specific item, or a partial or full literal value. In the case of a literal value, if no entry in the index matches exactly, the closest match is used as the focus. It's quite reasonable to specify a focus of a single letter, for example.

Being able to specify a specific item to start at is particularly important with dates, since many items may have the same issue date. Say 30 items in a collection have the issue date 2002. To be able to page through the index 20 items at a time, you need to be able to specify exactly which item's 2002 is the focus of the browse, otherwise each time you invoked the browse code, the results would start at the first item with the issue date 2002.

Author browses return *String* objects with the actual author names. You can only specify the focus as a full or partial literal *String*.

Another important point to note is that presently, the browse indices contain metadata for all items in the main archive, regardless of authorization policies. This means that all items in the archive will appear to all users when browsing. Of course, should the user attempt to access a non-public item, the usual authorization mechanism will apply. Whether this approach is ideal is under review; implementing the browse API such that the results retrieved reflect a user's level of authorization may be possible, but rather tricky.

11.10.1. Index Maintenance

The browse API contains calls to add and remove items from the index, and to regenerate the indices from scratch. In general the content management API invokes the necessary browse API calls to keep the browse indices in sync with what is in the archive, so most applications will not need to invoke those methods.

If the browse index becomes inconsistent for some reason, the *InitializeBrowse* class is a command line tool (generally invoked using the */dspace/bin/dspace index-init* command) that causes the indexes to be regenerated from scratch.

11.10.2. Caveats

Presently, the browse API is not tremendously efficient. 'Indexing' takes the form of simply extracting the relevant Dublin Core value, normalising it (lower-casing and removing any leading article in the case of titles), and inserting that normalized value with the corresponding item ID in the appropriate browse database table. Database views of this table include collection and community IDs for browse operations with a limited scope. When a browse operation is performed, a simple *SELECT* query is performed, along the lines of:

```
SELECT item_id FROM ItemsByTitle ORDER BY sort_title OFFSET 40 LIMIT 20
```

There are two main drawbacks to this: Firstly, *LIMIT* and *OFFSET* are PostgreSQL-specific keywords. Secondly, the database is still actually performing dynamic sorting of the titles, so the browse code as it



stands will not scale particularly well. The code does cache *BrowseInfo* objects, so that common browse operations are performed quickly, but this is not an ideal solution.

11.11. Checksum checker

Checksum checker is used to verify every item within DSpace. While DSpace calculates and records the checksum of every file submitted to it, the checker can determine whether the file has been changed. The idea being that the earlier you can identify a file has changed, the more likely you would be able to record it (assuming it was not a wanted change).

org.dspace.checker.CheckerCommand class, is the class for the checksum checker tool, which calculates checksums for each bitstream whose ID is in the *most_recent_checksum* table, and compares it against the last calculated checksum for that bitstream.

11.12. OpenSearch Support

DSpace is able to support OpenSearch. For those not acquainted with the standard, a very brief introduction, with emphasis on what possibilities it holds for current use and future development.

OpenSearch is a small set of conventions and documents for describing and using 'search engines', meaning any service that returns a set of results for a query. It is nearly ubiquitous—but also nearly invisible—in modern web sites with search capability. If you look at the page source of Wikipedia, Facebook, CNN, etc you will find buried a link element declaring OpenSearch support. It is very much a lowest-common-denominator abstraction (think Google box), but does provide a means to extend its expressive power. This first implementation for DSpace supports *none* of these extensions—many of which are of potential value—so it should be regarded as a foundation, not a finished solution. So the short answer is that DSpace appears as a 'search-engine' to OpenSearch-aware software.

Another way to look at OpenSearch is as a RESTful web service for search, very much like SRW/U, but considerably simpler. This comparative loss of power is offset by the fact that it is widely supported by web tools and players: browsers understand it, as do large metasearch tools.

How Can It Be Used

- **Browser Integration** Many recent browsers (IE7+, FF2+) can detect, or 'autodiscover', links to the document describing the search engine. Thus you can easily add your or other DSpace instances to the drop-down list of search engines in your browser. This list typically appears in the upper right corner of the browser, with a search box. In Firefox, for example, when you visit a site supporting OpenSearch, the color of the drop-down list widget changes color, and if you open it to show the list of search engines, you are offered an opportunity to add the site to the list. IE works nearly the same way but instead labels the web sites 'search providers'. When you select a DSpace instance as the search engine and enter a search, you are simply sent to the regular search results page of the instance.
- **Flexible, interesting RSS Feeds** Because one of the formats that OpenSearch specifies for its results is RSS (or Atom), you can turn any search query into an RSS feed. So if there are keywords highly discriminative of content in a collection or repository, these can be turned into a URL that a feed reader can subscribe to. Taken to the extreme, one could take any search a user makes, and dynamically compose an RSS feed URL for it in the page of returned results. To see an example, if you have a DSpace with OpenSearch enabled, try: <http://dspace.mysite.edu/open-search/?query-<your query>> The default format returned is Atom 1.0, so you should see an Atom document containing your search results.
- You can extend the syntax with a few other parameters, as follows: |Parameter |Values |

format	atom, rss, html
scope	<handle>—search is restricted to a collection or community with the indicated handle.



rpp	number indicating the number of results per page (i.e. per request)
start	number of page to start with (if paginating results)
sort_by	number indicating sorting criteria (same as DSpace advanced search values)

- Cheap metasearch Search aggregators like A9 (Amazon) recognize OpenSearch-compliant providers, and so can be added to metasearch sets using their UIs. Then you site can be used to aggregate search results with others.
Configuration is through the `_dspace.cfg` file. See OpenSearch Support

11.13. Embargo

The architecture of Embargo is documented in the package javadocs. Run `cd [dspace-source]/dspace; mvn javadoc:javadoc` and look in `[dspace-source]/dspace-api/target/site/apidocs/index.html`.

12. Submission

12.1. Understanding the Submission Configuration File

The `[dspace]/config/item-submission.xml` contains the submission configurations for *both* the DSpace JSP user interface (JSPUI) or the DSpace XML user interface (XMLUI or Manakin). This configuration file contains detailed documentation within the file itself, which should help you better understand how to best utilize it.

12.1.1. The Structure of item-submission.xml

```
<item-submission>
  <!-- Where submission processes are mapped to specific Collections -->
  <submission-map>
    <name-map collection-handle="default" submission-name="traditional" /> ...
  </submission-map>
  <!-- Where "steps" which are used across many submission processes can be defined in a
       single place. They can then be referred to by ID later. -->
  <step-definitions>
    <step id="collection">
      <processing-class>org.dspace.submit.step.SelectCollectionStep</processing-
class>
      <workflow-editable>>false</workflow-editable>
    </step>
    ...
  </step-definitions>
  <!-- Where actual submission processes are defined and given names. Each <submission-
process> has
       many <step> nodes which are in the order that the steps should be in.-->
  <submission-definitions> <submission-process name="traditional">
    ...
  <!-- Step definitions appear here! -->
  </submission-process>
  ...
</submission-definitions>
</item-submission>
```

Because this file is in XML format, you should be familiar with XML before editing this file. By default, this file contains the "traditional" Item Submission Process for DSpace, which consists of the following Steps (in this order):

Select Collection -> Initial Questions -> Describe -> Upload -> Verify -> License -> Complete



If you would like to customize the steps used or the ordering of the steps, you can do so within the `<submission-definition>` section of the `item-submission.xml`.

In addition, you may also specify different Submission Processes for different DSpace Collections. This can be done in the `<submission-map>` section. The `item-submission.xml` file itself documents the syntax required to perform these configuration changes.

12.1.2. Defining Steps (`<step>`) within the `item-submission.xml`

This section describes how Steps of the Submission Process are defined within the `item-submission.xml`.

Where to place your `<step>` definitions

`<step>` definitions can appear in one of two places within the `item-submission.xml` configuration file.

1. Within the `<step-definitions>` section

- This is for globally defined `<step>` definitions (i.e. steps which are used in multiple `<submission-process>` definitions). Steps defined in this section **must** define a unique `id` which can be used to reference this step.
- For example:

```
<step-definitions>
  <step id="custom-step">
    ...
  </step>
  ...
</step-definitions>
```

- The above step definition could then be referenced from within a `<submission-process>` as simply `<step id="custom-step"/>`

2. Within a specific `<submission-process>` definition

- This is for steps which are specific to a single `<submission-process>` definition.
- For example:

```
<submission-process>
  <step>
    ...
  </step>
</submission-process>
```

The ordering of `<step>` definitions matters !

The ordering of the `<step>` tags within a `<submission-process>` definition directly corresponds to the order in which those steps will appear!

For example, the following defines a Submission Process where the *License* step directly precedes the *Initial Questions* step (more information about the structure of the information under each `<step>` tag can be found in the section on Structure of the `<step>` Definition below):

```
<submission-process>
  <!--Step 1 will be to Sign off on the License-->
  <step>
    <heading>submit.progressbar.license</heading>
    <processing-class>org.dspace.submit.step.LicenseStep</processing-class>
    <jspui-binding>org.dspace.app.webui.submit.step.JSPLicenseStep</jspui-binding>
    <xmlui-binding>org.dspace.app.xmlui.aspect.submission.submit.LicenseStenseStep</xmlui-binding>
```




```

        <workflow-editable>false</workflow-editable>
    </step>
    <!--Step 2 will be to Ask Initial Questions-->
    <step>
        <heading>submit.progressbar.initial-questions</heading>
        <processing-class>org.dspace.submit.step.InitialQuestionsStep</process;/
processing-class>
        <jspui-binding>org.dspace.app.webui.submit.step.JSPInitialQuestionsSteonsStep</
jspui-binding>
        <xmlui-
binding>org.dspace.app.xmlui.aspect.submission.submit.InitialQutialQuestionsStep</xmlui-
binding>
        <workflow-editable>true</workflow-editable>
    </step>
    ...[other steps]...
</submission-process>

```

Structure of the <step> Definition

The same <step> definition is used by both the DSpace JSP user interface (JSPUI) an the DSpace XML user interface (XMLUI or Manakin). Therefore, you will notice each <step> definition contains information specific to each of these two interfaces.

The structure of the <step> Definition is as follows:

```

<step>
<heading>submit.progressbar.describe</heading>
<processing-class>org.dspace.submit.step.DescribeStep</processing-classing-class>
<jspui-binding>org.dspace.app.webui.submit.step.JSPDescribeStep</jspuilt;/jspui-binding>
<xmlui-binding>org.dspace.app.xmlui.aspect.submission.submit.DescribeScribeStep</xmlui-
binding>
<workflow-editable>true</workflow-editable>
</step>

```

Each *step* contains the following elements. The required elements are so marked:

- ***heading***: Partial I18N key (defined in *Messages.properties* for JSPUI or *messages.xml* for XMLUI) which corresponds to the text that should be displayed in the submission Progress Bar for this step. This partial I18N key is prefixed within either the *Messages.properties* or *messages.xml* file, depending on the interface you are using. Therefore, to find the actual key, you will need to search for the partial key with the following prefix:
 - XMLUI: prefix is *xmlui.Submission*. (e.g. "xmlui.Submission.submit.progressbar.describe" for 'Describe' step)
 - JSPUI: prefix is *jsp*. (e.g. "jsp.submit.progressbar.describe" for 'Describe' step)*The 'heading' need not be defined if the step should not appear in the progress bar (e.g. steps which perform automated processing, i.e. non-interactive, should not appear in the progress bar).*
- ***processing-class(Required)**: Full Java path to the Processing Class for this Step. This Processing Class must perform the primary processing of any information gathered in this step, for both the XMLUI and JSPUI. All valid step processing classes must extend the abstract `org.dspace.submit.AbstractProcessingStep`` class (or alternatively, extend one of the pre-existing step processing classes in `org.dspace.submit.step`.)
- ***jspui-binding***: Full Java path of the JSPUI "binding" class for this Step. This "binding" class should initialize and call the appropriate JSPs to display the step's user interface. A valid JSPUI "binding" class must extend the abstract `org.dspace.app.webui.submit.JSPStep`` class. *This property need not be defined if you are using the XMLUI interface, or for steps which only perform automated processing, i.e. non-interactive steps.*
- ***xmlui-binding***: Full Java path of the XMLUI "binding" class for this Step. This "binding" class should generate the Manakin XML (DRI document) necessary to generate the step's user interface. A valid XMLUI "binding" class must extend the abstract `org.dspace.app.xmlui.submission.AbstractSubmissionStep``



class. *This property need not be defined if you are using the JSPUI interface, or for steps which only perform automated processing, i.e. non-interactive steps.*

- ***workflow-editable***: Defines whether or not this step can be edited during the *Edit Metadata* process with the DSpace approval/rejection workflow process. Possible values include *true* and *false*. If undefined, defaults to *true* (which means that workflow reviewers would be allowed to edit information gathered during that step).

12.2. Reordering/Removing Submission Steps

The removal of existing steps and reordering of existing steps is a relatively easy process!

Reordering steps

1. Locate the `<submission-process>` tag which defines the Submission Process that you are using. If you are unsure which Submission Process you are using, it's likely the one with `name="traditional"`, since this is the traditional DSpace submission process.
2. Reorder the `<step>` tags within that `<submission-process>` tag. Be sure to move the *entire* `<step>` tag (i.e. everything between and including the opening `<step>` and closing `</step>` tags).
 - *Hint #1*: The `<step>` defining the *Review/Verify* step only allows the user to review information from steps which appear **before** it. So, it's likely you'd want this to appear as one of your last few steps
 - *Hint #2*: If you are using it, the `<step>` defining the *Initial Questions* step should always appear **before** the *Upload* or *Describe* steps since it asks questions which help to set up those later steps.

Removing one or more steps

1. Locate the `<submission-process>` tag which defines the Submission Process that you are using. If you are unsure which Submission Process you are using, it's likely the one with `name="traditional"`, since this is the traditional DSpace submission process.
2. Comment out (i.e. surround with `<!--` and `-->`) the `<step>` tags which you want to remove from that `<submission-process>` tag. Be sure to comment out the *entire* `<step>` tag (i.e. everything between and including the opening `<step>` and closing `</step>` tags).
 - *Hint #1*: You cannot remove the *Select a Collection* step, as an DSpace Item cannot exist without belonging to a Collection.
 - *Hint #2*: If you decide to remove the `<step>` defining the *Initial Questions* step, you should be aware that this may affect your *Describe* and *Upload* steps! The *Initial Questions* step asks questions which help to initialize these later steps. If you decide to remove the *Initial Questions* step you may wish to create a custom, automated step which will provide default answers for the questions asked!

12.3. Assigning a custom Submission Process to a Collection

Assigning a custom submission process to a Collection in DSpace involves working with the *submission-map* section of the *item-submission.xml*. For a review of the structure of the *item-submission.xml* see the section above on Understanding the Submission Configuration File.

Each *name-map* element within *submission-map* associates a collection with the name of a submission definition. Its *collection-handle* attribute is the Handle of the collection. Its *submission-name* attribute is the submission definition name, which must match the *name* attribute of a *submission-process* element (in the *submission-definitions* section of *item-submission.xml*).

For example, the following fragment shows how the collection with handle "12345.6789/42" is assigned the "custom" submission process:



```
<submission-map>
  <name-map collection-handle=" 12345.6789/42" submission-name="
  custom" />
  ...
</submission-map>

<submission-definitions>
  <submission-process name="
  custom">
  ...
</submission-definitions>
```

It's a good idea to keep the definition of the *default* name-map from the example *input-forms.xml* so there is always a default for collections which do not have a custom form set.

12.3.1. Getting A Collection's Handle

You will need the *handle* of a collection in order to assign it a custom form set. To discover the handle, go to the "Communities & Collections" page under "**Browse**" in the left-hand menu on your DSpace home page. Then, find the link to your collection. It should look something like:

```
http://myhost.my.edu/dspace/handle/
12345.6789/42
```

The underlined part of the URL is the handle. It should look familiar to any DSpace administrator. That is what goes in the *collection-handle* attribute of your *name-map* element.

12.4. Custom Metadata-entry Pages for Submission

12.4.1. Introduction

This section explains how to customize the Web forms used by submitters and editors to enter and modify the metadata for a new item. These metadata web forms are controlled by the *Describe* step within the Submission Process. However, they are also configurable via their own XML configuration file (*input-forms.xml*).

You can customize the "default" metadata forms used by all collections, and also create alternate sets of metadata forms and assign them to specific collections. In creating custom metadata forms, you can choose:

- The number of metadata-entry pages.
 - Which fields appear on each page, and their sequence.
 - Labels, prompts, and other text associated with each field.
 - List of available choices for each menu-driven field.
- *N.B.*The cosmetic and ergonomic details of metadata entry fields remain the same as the fixed metadata pages in previous DSpace releases, and can only be altered by modifying the appropriate stylesheet and JSP pages.

All of the custom metadata-entry forms for a DSpace instance are controlled by a single XML file, *input-forms.xml*, in the *config* subdirectory under the DSpace home. DSpace comes with a sample configuration that implements the traditional metadata-entry forms, which also serves as a well-documented example. The rest of this section explains how to create your own sets of custom forms.

12.4.2. Describing Custom Metadata Forms

The description of a set of pages through which submitters enter their metadata is called a *form* (although it is actually a set of forms, in the HTML sense of the term). A form is identified by a unique symbolic *name*.



In the XML structure, the *form* is broken down into a series of *pages*: each of these represents a separate Web page for collecting metadata elements.

To set up one of your DSpace collections with customized submission forms, first you make an entry in the *form-map*. This is effectively a table that relates a collection to a form set, by connecting the collection's *Handle* to the form name. Collections are identified by handle because their names are mutable and not necessarily unique, while handles are unique and persistent.

A special map entry, for the collection handle "default", defines the *default* form set. It applies to all collections which are not explicitly mentioned in the map. In the example XML this form set is named *traditional* (for the "traditional" DSpace user interface) but it could be named anything.

12.4.3. The Structure of input-forms.xml

The XML configuration file has a single top-level element, *input-forms*, which contains three elements in a specific order. The outline is as follows:

```
<input-forms>
  <!-- Map of Collections to Form Sets -->
  <form-map>
    <name-map collection-handle="default" form-name="traditional"
  />
    ...
  </form-map>

  <!-- Form Set Definitions -->
  <form-definitions>
    <form name="traditional">
    ...
  </form-definitions>

  <!-- Name/Value Pairs used within Multiple Choice Widgets
  -->
  <form-value-pairs>
    <value-pairs value-pairs-name="common_iso_languages"
  dc-term="language_iso">
    ...
  </form-value-pairs>
</input-forms>
```

Adding a Collection Map

Each *name-map* element within *form-map* associates a collection with the name of a form set. Its *collection-handle* attribute is the Handle of the collection, and its *form-name* attribute is the form set name, which must match the *name* attribute of a *form* element.

For example, the following fragment shows how the collection with handle "12345.6789/42" is attached to the "TechRpt" form set:

```
<form-map>
  <name-map collection-handle=" 12345.6789/42" form-name=" TechRpt "
  />
  ...
</form-map>

<form-definitions>
  <form name="
  TechRept ">
  ...
</form-definitions>
```

It's a good idea to keep the definition of the *default* name-map from the example *input-forms.xml* so there is always a default for collections which do not have a custom form set.



Getting A Collection's Handle

You will need the *handle* of a collection in order to assign it a custom form set. To discover the handle, go to the "Communities & Collections" page under "**Browse**" in the left-hand menu on your DSpace home page. Then, find the link to your collection. It should look something like:

```
http://myhost.my.edu/dspace/handle/  
12345.6789/42
```

The underlined part of the URL is the handle. It should look familiar to any DSpace administrator. That is what goes in the *collection-handle* attribute of your *name-map* element.

Adding a Form Set

You can add a new form set by creating a new *form* element within the *form-definitions* element. It has one attribute, *name*, which as seen above must match the value of the *name-map* for the collections it is to be used for.

Forms and Pages

The content of the *form* is a sequence of *page* elements. Each of these corresponds to a Web page of forms for entering metadata elements, presented in sequence between the initial "Describe" page and the final "Verify" page (which presents a summary of all the metadata collected).

A *form* must contain at least one and at most six pages. They are presented in the order they appear in the XML. Each *page* element must include a *number* attribute, that should be its sequence number, e.g.

```
<page number="1">
```

The *page* element, in turn, contains a sequence of *field* elements. Each field defines an interactive dialog where the submitter enters one of the Dublin Core metadata items.

Composition of a Field

Each *field* contains the following elements, in the order indicated. The required sub-elements are so marked:

- ***dc-schema(Required)***: Name of metadata schema employed, e.g. *dc* for Dublin Core. This value must match the value of the *schema* element defined in *dublin-core-types.xml*
- ***dc-element(Required)***: Name of the Dublin Core element entered in this field, e.g. *contributor*.
- ***dc-qualifier***: Qualifier of the Dublin Core element entered in this field, e.g. when the field is *contributor.advisor* the value of this element would be *advisor*. Leaving this out means the input is for an unqualified DC element.
- ***repeatable***: Value is *true* when multiple values of this field are allowed, *false* otherwise. When you mark a field repeatable, the UI servlet will add a control to let the user ask for more fields to enter additional values. Intended to be used for arbitrarily-repeating fields such as subject keywords, when it is impossible to know in advance how many input boxes to provide.
- ***label(Required)***: Text to display as the label of this field, describing what to enter, e.g. "*Your Advisor's Name*".
- ***input-type(Required)***: Defines the kind of interactive widget to put in the form to collect the Dublin Core value. Content must be one of the following keywords:
 - **onebox** – A single text-entry box.



- **twobox** – A pair of simple text-entry boxes, used for *repeatable* values such as the DC *subject* item. *Note:* The 'twobox' input type is rendered the same as a 'onebox' in the XML-UI, but both allow for ease of adding multiple values.
- **textarea** – Large block of text that can be entered on multiple lines, e.g. for an abstract.
- **name** – Personal name, with separate fields for family name and first name. When saved they are appended in the format 'LastName, FirstName'
- **date** – Calendar date. When required, demands that at least the year be entered.
- **series** – Series/Report name and number. Separate fields are provided for series name and series number, but they are appended (with a semicolon between) when saved.
- **dropdown** – Choose value(s) from a "drop-down" menu list. **Note:** You must also include a value for the *value-pairs-name* attribute to specify a list of menu entries from which to choose. Use this to make a choice from a restricted set of options, such as for the *language* item.
- **qualdrop_value** – Enter a "qualified value", which includes *both* a qualifier from a drop-down menu and a free-text value. Used to enter items like alternate identifiers and codes for a submitted item, e.g. the DC *identifier* field. **Note:** As for the *dropdown* type, you must include the *value-pairs-name* attribute to specify a menu choice list.
- **list** – Choose value(s) from a checkbox or radio button list. If the *repeatable* attribute is set to *true*, a list of checkboxes is displayed. If the *repeatable* attribute is set to *false*, a list of radio buttons is displayed. **Note:** You must also include a value for the *value-pairs-name* attribute to specify a list of values from which to choose.
- ***hint(Required)*:** Content is the text that will appear as a "hint", or instructions, next to the input fields. Can be left empty, but it must be present.
- ***required*:** When this element is included with any content, it marks the field as a required input. If the user tries to leave the page without entering a value for this field, that text is displayed as a warning message. For example, `<required>You must enter a title.</required>` *Note that leaving the required element empty will not mark a field as required, e.g.:<required></required>*
- ***visibility*:** When this optional element is included with a value, it restricts the visibility of the field to the scope defined by that value. If the element is missing or empty, the field is visible in all scopes. Currently supported scopes are:
 - **workflow** : the field will only be visible in the workflow stages of submission. This is good for hiding difficult fields for users, such as subject classifications, thereby easing the use of the submission system.
 - **submit** : the field will only be visible in the initial submission, and not in the workflow stages. In addition, you can decide which type of restriction apply: read-only or full hidden the field (default behaviour) using the *otherwise* attribute of the *visibility* XML element. For example:`<visibility otherwise="readonly">workflow</visibility>` *Note that it is considered a configuration error to limit a field's scope while also requiring it - an exception will be generated when this combination is detected. Look at the example *input-forms.xml* and experiment with a a trial custom form to learn this specification language thoroughly. It is a very simple way to express the layout of data-entry forms, but the only way to learn all its subtleties is to use it.*

For the use of controlled vocabularies see the Configuring Controlled Vocabularies section.

Automatically Elided Fields

You may notice that some fields are automatically skipped when a custom form page is displayed, depending on the kind of item being submitted. This is because the DSpace user-interface engine skips Dublin Core fields which are not needed, according to the initial description of the item. For example, if the user indicates



there are no alternate titles on the first "Describe" page (the one with a few checkboxes), the input for the *title.alternative* DC element is automatically elided, *even on custom submission pages*.

When a user initiates a submission, DSpace first displays what we'll call the "initial-questions page". By default, it contains three questions with check-boxes:

1. **The item has more than one title, e.g. a translated title** Controls *title.alternative* field.
2. **The item has been published or publicly distributed before** Controls DC fields:
 - *date.issued*
 - *publisher*
 - *identifier.citation*
3. **The item consists of more than one file** *Does not affect any metadata input fields*. The answers to the first two questions control whether inputs for certain of the DC metadata fields will displayed, even if they are defined as fields in a custom page. Conversely, if the metadata fields controlled by a checkbox are not mentioned in the custom form, the checkbox is elided from the initial page to avoid confusing or misleading the user.

The two relevant checkbox entries are "The item has more than one title, e.g. a translated title", and "The item has been published or publicly distributed before". The checkbox for multiple titles trigger the display of the field with dc-element equal to 'title' and dc-qualifier equal to 'alternative'. If the controlling collection's form set does not contain this field, then the multiple titles question will not appear on the initial questions page.

Adding Value-Pairs

Finally, your custom form description needs to define the "value pairs" for any fields with input types that refer to them. Do this by adding a *value-pairs* element to the contents of *form-value-pairs*. It has the following required attributes:

- **value-pairs-name** – Name by which an *input-type* refers to this list.
- **dc-term** – Qualified Dublin Core field for which this choice list is selecting a value. Each *value-pairs* element contains a sequence of *pair* sub-elements, each of which in turn contains two elements:
- **displayed-value** – Name shown (on the web page) for the menu entry.
- **stored-value** – Value stored in the DC element when this entry is chosen. Unlike the HTML *select* tag, there is no way to indicate one of the entries should be the default, so the first entry is always the default choice.

Example

Here is a menu of types of common identifiers:

```
<value-pairs value-pairs-name="common_identifiers"
dc-term="identifier">
  <pair>
    <displayed-value>Gov't Doc
  #</displayed-value>
    <stored-value>govdoc</stored-value>
  </pair>
  <pair>
    <displayed-value>URI</displayed-value>
    <stored-value>uri</stored-value>
  </pair>
  <pair>
    <displayed-value>ISBN</displayed-value>
    <stored-value>isbn</stored-value>
  </pair>
</value-pairs>
```



It generates the following HTML, which results in the menu widget below. (Note that there is no way to indicate a default choice in the custom input XML, so it cannot generate the HTML *SELECTED* attribute to mark one of the options as a pre-selected default.)

```
<select name="identifier_qualifier_0">
<option VALUE="govdoc">Gov't Doc
 #</option>
<option VALUE="uri">URI</option>
<option VALUE="isbn">ISBN</option>
</select>
```

*Identifiers:*Gov't Doc #URIISBN

12.4.4. Deploying Your Custom Forms

The DSpace web application only reads your custom form definitions when it starts up, so it is important to remember:

You must always restart Tomcat (or whatever servlet container you are using) for changes made to the *_input-forms.xml* file take effect.

Any mistake in the syntax or semantics of the form definitions, such as poorly formed XML or a reference to a nonexistent field name, will cause a fatal error in the DSpace UI. The exception message (at the top of the stack trace in the *dspace.log* file) usually has a concise and helpful explanation of what went wrong. Don't forget to stop and restart the servlet container before testing your fix to a bug.

12.5. Configuring the File Upload step

The *Upload* step in the DSpace submission process has two configuration options which can be set with your *[dspace]/config/dspace.cfg* configuration file. They are as follows:

- *upload.max* - The maximum size of a file (in bytes) that can be uploaded from the JSPUI (not applicable for the XMLUI). It defaults to 536870912 bytes (512MB). You may set this to -1 to disable any file size limitation.
- *Note*: Increasing this value or setting to -1 does **not** guarantee that DSpace will be able to successfully upload larger files via the web, as large uploads depend on many other factors including bandwidth, web server settings, internet connection speed, etc.
- *webui.submit.upload.required* - Whether or not all users are *required* to upload a file when they submit an item to DSpace. It defaults to 'true'. When set to 'false' users will see an option to skip the upload step when they submit a new item.

12.6. Creating new Submission Steps

First, a brief warning: *Creating a new Submission Step requires some Java knowledge, and is therefore recommended to be undertaken by a Java programmer whenever possible*

That being said, at a higher level, creating a new Submission Step requires the following (in this relative order):

1. **(Required)** Create a new Step Processing class
 - This class **must** extend the abstract *org.dspace.submit.AbstractProcessingStep* class and implement all methods defined by that abstract class.
 - This class should be built in such a way that it can process the input gathered from *either* the XMLUI or JSPUI interface.
2. *(For steps using JSPUI)* Create the JSPs to display the user interface. Create a new JSPUI "binding" class to initialize and call these JSPs.



3. • Your JSPUI "binding" class must extend the abstract class *org.dspace.app.webui.submit.JSPStep* and implement all methods defined there. It's recommended to use one of the classes in *org.dspace.app.webui.submit.step.** as a reference.
 - Any JSPs created should be loaded by calling the `showJSP()` method of the *org.dspace.app.webui.submit.JSPStepManager* class
 - If this step gathers information to be reviewed, you must also create a Review JSP which will display a read-only view of all data gathered during this step. The path to this JSP must be returned by your `getReviewJSP()` method. You will find examples of Review JSPs (named similar to *review-[step].jsp*) in the JSP *submit/* directory.
4. (For steps using XMLUI) Create an XMLUI "binding" Step Transformer which will generate the DRI XML which Manakin requires.
 - The Step Transformer must extend and implement all necessary methods within the abstract class *org.dspace.app.xmlui.submission.AbstractSubmissionStep*
 - It is useful to use the existing classes in *org.dspace.app.xmlui.submission.submit.** as references
5. (Required) Add a valid Step Definition to the *item-submission.xml* configuration file.
 - This may also require that you add an I18N (Internationalization) key for this step's *heading*. See the sections on Configuring Multilingual Support for JSPUI or Configuring Multilingual Support for XMLUI for more details.
 - For more information on `<step>` definitions within the *item-submission.xml*, see the section above on Defining Steps (`<step>`) within the *item-submission.xml*. Creating a Non-Interactive Step

Non-interactive steps are ones that have no user interface and only perform backend processing. You may find a need to create non-interactive steps which perform further processing of previously entered information.

To create a non-interactive step, do the following:

1. Create the required Step Processing class, which extends the abstract *org.dspace.submit.AbstractProcessingStep* class. In this class add any processing which this step will perform.
2. Add your non-interactive step to your *item-submission.xml* at the place where you wish this step to be called during the submission process. For example, if you want it to be called *immediately after* the existing 'Upload File' step, then place its configuration immediately after the configuration for that 'Upload File' step. The configuration should look similar to the following:

```
<step>
  <processing-class>org.dspace.submit.step.MyNonInteractiveStep</processi
  /processing-class> <workflow-editable>>false</workflow-editable>
</step>
```

Note: Non-interactive steps will not appear in the Progress Bar! Therefore, your submitters will not even know they are there. However, because they are not visible to your users, you should make sure that your non-interactive step does not take a large amount of time to finish its processing and return control to the next step (otherwise there will be a visible time delay in the user interface).

13. Appendices

13.1. Appendix

DSpace System Documentation: Appendix A



13.1.1. Default Dublin Core Metadata Registry

contributor		A person, organization, or service responsible for the content of the resource. Catch-all for unspecified contributors.
contributor	advisor	Use primarily for thesis advisor.
contributor ¹	author	
contributor	editor	
contributor	illustrator	
contributor	other	
coverage	spatial	Spatial characteristics of content.
coverage	temporal	Temporal characteristics of content.
creator		Do not use; only for harvested metadata.
date		Use qualified form if possible.
date ¹	accessioned	Date DSpace takes possession of item.
date ¹	available	Date or date range item became available to the public.
date	copyright	Date of copyright.
date	created	Date of creation or manufacture of intellectual content if different from date.issued.
date ¹	issued	Date of publication or distribution.
date	submitted	Recommend for theses/dissertations.
identifier		Catch-all for unambiguous identifiers not defined by qualified form; use identifier.other for a known identifier common to a local collection instead of unqualified form.
identifier ¹	citation	Human-readable, standard bibliographic citation of non-DSpace format of this item
identifier ¹	govdoc	A government document number
identifier ¹	isbn	International Standard Book Number
identifier ¹	issn	International Standard Serial Number
identifier	sici	Serial Item and Contribution Identifier
identifier ¹	ismn	International Standard Music Number
identifier ¹	other	A known identifier type common to a local collection.



identifier ¹	uri	Uniform Resource Identifier
description ¹		Catch-all for any description not defined by qualifiers.
description ¹	abstract	Abstract or summary.
description ¹	provenance	The history of custody of the item since its creation, including any changes successive custodians made to it.
description ¹	sponsorship	Information about sponsoring agencies, individuals, or contractual arrangements for the item.
description	statementofresponsibility	To preserve statement of responsibility from MARC records.
description	tableofcontents	A table of contents for a given item.
description	uri	Uniform Resource Identifier pointing to description of this item.
format ¹		Catch-all for any format information not defined by qualifiers.
format ¹	extent	Size or duration.
format	medium	Physical medium.
format ¹	mimetype	Registered MIME type identifiers.
language		Catch-all for non-ISO forms of the language of the item, accommodating harvested values.
language ¹	iso	Current ISO standard for language of intellectual content, including country codes (e.g. "en_US").
publisher ¹		Entity responsible for publication, distribution, or imprint.
relation		Catch-all for references to other related items.
relation	isformatof	References additional physical form.
relation	ispartof	References physically or logically containing item.
relation ¹	ispartofseries	Series name and number within that series, if available.
relation	haspart	References physically or logically contained item.
relation	isversionof	References earlier version.
relation	hasversion	References later version.
relation	isbasedon	References source.
relation	isreferencedby	Pointed to by referenced resource.
relation	requires	Referenced resource is required to support function, delivery, or coherence of item.
relation	replaces	References preceding item.



relation	isreplacedby	References succeeding item.
relation	uri	References Uniform Resource Identifier for related item
rights		Terms governing use and reproduction.
rights	uri	References terms governing use and reproduction.
source		Do not use; only for harvested metadata.
source	uri	Do not use; only for harvested metadata.
subject ¹		Uncontrolled index term.
subject	classification	Catch-all for value from local classification system. Global classification systems will receive specific qualifier
subject	ddc	Dewey Decimal Classification Number
subject	lcc	Library of Congress Classification Number
subject	lcsb	Library of Congress Subject Headings
subject	mesh	MEDical Subject Headings
subject	other	Local controlled vocabulary; global vocabularies will receive specific qualifier.
title ¹		Title statement/title proper.
title ¹	alternative	Varying (or substitute) form of title proper appearing in item, e.g. abbreviation or translation
type ¹		Nature or genre of content.

¹Used by system. **DO NOT REMOVE**

13.1.2. Default Bitstream Format Registry

Mimetype	Short Description	Description	Support Level	Internal	Extensions
application/octet-stream ¹	Unknown	Unknown data format	Unknown	false	
text/plain ¹	License	Item-specific license agreed upon to submission	Known	true	
application/marc	MARC	Machine-Readable Cataloging records	Known	false	



application/mathematica	Mathematica	Mathematica Notebook	Known	false	ma
application/msword	Microsoft Word	Microsoft Word	Known	false	doc
application/pdf	Adobe PDF	Adobe Portable Document Format	Known	false	pdf
application/postscript	Postscript	Postscript Files	Known	false	ai, eps, ps
application/sgml	SGML	SGML application (RFC 1874)	Known	false	sgm, sgml
application/vnd.ms-excel	Microsoft Excel	Microsoft Excel	Known	false	xls
application/vnd.ms-powerpoint	Microsoft PowerPoint	Microsoft PowerPoint	Known	false	ppt
application/vnd.ms-project	Microsoft Project	Microsoft Project	Known	false	mpd, mpp, mpx
application/vnd.visio	Microsoft Visio	Microsoft Visio	Known	false	vsd
application/wordperfect5.1	WordPerfect	WordPerfect 5.1 document	Known	false	wpd
application/x-dvi	TeX dvi	TeX dvi format	Known	false	dvi
application/x-filemaker	FMP3	Filemaker Pro	Known	false	fm
application/x-latex	LateX	LaTeX document	Known	false	latex
application/x-photoshop	Photoshop	Photoshop	Known	false	pdd, psd
application/x-tex	TeX	TeX/LaTeX document	Known	false	tex
audio/basic	audio/basic	Basic Audio	Known	false	au, snd
audio/x-aiff	AIFF	Audio Interchange File Format	Known	false	aif, aifc, aiff
audio/x-mpeg	MPEG Audio	MPEG Audio	Known	false	abs, mpa, mpeg-a
audio/x-pn-realaudio	RealAudio	RealAudio file	Known	false	ra, ram
audio/x-wav	WAV	Broadcast Wave Format	Known	false	wav
image/gif	GIF	Graphics Interchange Format	Known	false	gif



image/jpeg	JPEG	Joint Photo-graphic Experts Group/JPEG File Inter-change Format (JFIF)	Known	false	jpeg, jpg
image/png	image/png	Portable Network Graphics	Known	false	png
image/tiff	TIFF	Tag Image File Format	Known	false	tif, tiff
image/x-ms-bmp	BMP	Microsoft Windows bitmap	Known	false	bmp
image/x-photo-cd	Photo CD	Kodak Photo CD image	Known	false	pcd
text/css	CSS	Cascading Style Sheets	Known	false	css
text/html	HTML	Hypertext Markup Language	Known	false	htm, html
text/plain	Text	Plain Text	Known	false	asc, txt
text/richtext	RTF	Rich Text Format	Known	false	rtf
text/xml	XML	Extensible Markup Language	Known	false	xml
video/mpeg	MPEG	Moving Picture Experts Group	Known	false	mpe, mpeg, mpg
video/quicktime	Video Quick-time	Video Quick-time	Known	false	mov, qt

¹ Used by system: do not remove

13.2. DRI Schema Reference

13.2.1. Introduction

This manual describes the Digital Repository Interface (DRI) as it applies to the DSpace digital repository and XMLUI Manakin based interface. DSpace XML UI is a comprehensive user interface system. It is centralized and generic, allowing it to be applied to all DSpace pages, effectively replacing the JSP-based interface system. Its ability to apply specific styles to arbitrarily large sets of DSpace pages significantly eases the task of adapting the DSpace look and feel to that of the adopting institution. This also allows for several levels of branding, lending institutional credibility to the repository and collections.

Manakin, the second version of DSpace XML UI, consists of several components, written using Java, XML, and XSL, and is implemented in <http://cocoon.apache.org/>. Central to the interface is the XML Document, which is a semantic representation of a DSpace page. In Manakin, the XML Document adheres to a schema called the Digital Repository Interface (DRI) Schema, which was developed in conjunction with Manakin and is the subject of this guide. For the remainder of this guide, the terms XML Document, DRI Document, and Document will be used interchangeably.

This reference document explains the purpose of DRI, provides a broad architectural overview, and explains common design patterns. The appendix includes a complete reference for elements used in the DRI Schema, a graphical representation of the element hierarchy, and a quick reference table of elements and attributes.



The Purpose of DRI

DRI is a schema that governs the structure of the XML Document. It determines the elements that can be present in the Document and the relationship of those elements to each other. Since all Manakin components produce XML Documents that adhere to the DRI schema, The XML Document serves as the abstraction layer. Two such components, Themes and Aspects, are essential to the workings of Manakin and are described briefly in this manual.

The Development of DRI

The DRI schema was developed for use in Manakin. The choice to develop our own schema rather than adapt an existing one came after a careful analysis of the schema's purpose as well as the lessons learned from earlier attempts at customizing the DSpace interface. Since every DSpace page in Manakin exists as an XML Document at some point in the process, the schema describing that Document had to be able to structurally represent all content, metadata and relationships between different parts of a DSpace page. It had to be precise enough to avoid losing any structural information, and yet generic enough to allow Themes a certain degree of freedom in expressing that information in a readable format.

Popular schemas such as XHTML suffer from the problem of not relating elements together explicitly. For example, if a heading precedes a paragraph, the heading is related to the paragraph not because it is encoded as such but because it happens to precede it. When these structures are attempted to be translated into formats where these types of relationships are explicit, the translation becomes tedious, and potentially problematic. More structured schemas, like TEI or Docbook, are domain specific (much like DRI itself) and therefore not suitable for our purposes.

We also decided that the schema should natively support a metadata standard for encoding artifacts. Rather than encoding artifact metadata in structural elements, like tables or lists, the schema would include artifacts as objects encoded in a particular standard. The inclusion of metadata in native format would enable the Theme to choose the best method to render the artifact for display without being tied to a particular structure.

Ultimately, we chose to develop our own schema. We have constructed the DRI schema by incorporating other standards when appropriate, such as <http://cocoon.apache.org/2.1/userdocs/i18nTransformer.html> for internationalization, <http://dublincore.org/>, and the <http://www.loc.gov/standards/mets/>. The design of structural elements was derived primarily from <http://www.tei-c.org/index.xml>, with some of the design patterns borrowed from other existing standards such as DocBook and XHTML. While the structural elements were designed to be easily translated into XHTML, they preserve the semantic relationships for use in more expressive languages.

13.2.2. DRI in Manakin

The general process for handling a request in DSpace XML UI consists of two parts. The first part builds the XML Document, and the second part stylizes that Document for output. In Manakin, the two parts are not discrete and instead wrapped within two processes: Content Generation, which builds an XML representation of the page, and Style Application, which stylizes the resulting Document. Content Generation is performed by Aspect chaining, while Style Application is performed by a Theme.

Themes

A Theme is a collection of XSL stylesheets and supporting files like images, CSS styles, translations, and help documents. The XSL stylesheets are applied to the DRI Document to convert it into a readable format and give it structure and basic visual formatting in that format. The supporting files are used to provide the page with a specific look and feel, insert images and other media, translate the content, and perform other tasks. The currently used output format is XHTML and the supporting files are generally limited to CSS, images, and JavaScript. More output formats, like PDF or SVG, may be added in the future.

A DSpace installation running Manakin may have several Themes associated with it. When applied to a page, a Theme determines most of the page's look and feel. Different themes can be applied to different sets of DSpace pages allowing for both variety of styles between sets of pages and consistency within those sets. The `xmlui.xconf` configuration file determines which Themes are applied to which DSpace pages (see



the Configuration and Customization chapter for more information on installing and configuring themes). Themes may be configured to apply to all pages of specific type, like browse-by-title, to all pages of a one particular community or collection or sets of communities and collections, and to any mix of the two. They can also be configured to apply to a single arbitrary page or handle.

Aspect Chains

Manakin Aspects are arrangements of Cocoon components (transformers, actions, matchers, etc) that implement a new set of coupled features for the system. These Aspects are chained together to form all the features of Manakin. Five Aspects exist in the default installation of Manakin, each handling a particular set of features of DSpace, and more can be added to implement extra features. All Aspects take a DRI Document as input and generate one as output. This allows Aspects to be linked together to form an Aspect chain. Each Aspect in the chain takes a DRI Document as input, adds its own functionality, and passes the modified Document to the next Aspect in the chain.

13.2.3. Common Design Patterns

There are several design patterns used consistently within the DRI schema. This section identifies the need for and describes the implementation of these patterns. Three patterns are discussed: language and internationalization issues, standard attribute triplet (*id*, *n*, and *rend*), and the use of structure-oriented markup.

Localization and Internationalization

Internationalization is a very important component of the DRI system. It allows content to be offered in other languages based on user's locale and conditioned upon availability of translations, as well as present dates and currency in a localized manner. There are two types of translated content: content stored and displayed by DSpace itself, and content introduced by the DRI styling process in the XSL transformations. Both types are handled by Cocoon's *i18n* transformer without regard to their origin.

When the Content Generation process produces a DRI Document, some of the textual content may be marked up with *i18n* elements to signify that translations are available for that content. During the Style Application process, the Theme can also introduce new textual content, marking it up with *i18n* tags. As a result, after the Theme's XSL templates are applied to the DRI Document, the final output consists of a DSpace page marked up in the chosen display format (like XHTML) with *i18n* elements from both DSpace and XSL content. This final document is sent through Cocoon's *i18n* transformer that translates the marked up text.

Standard attribute triplet

Many elements in the DRI system (all top-level containers, character classes, and many others) contain one or several of the three standard attributes: *id*, *n*, and *rend*. The *id* and *n* attributes can be required or optional based on the element's purpose, while the *rend* attribute is always optional. The first two are used for identification purposes, while the third is used as a display hint issued to the styling step.

Identification is important because it allows elements to be separated from their peers for sorting, special case rendering, and other tasks. The first attribute, *id*, is the global identifier and it is unique to the entire document. Any element that contains an *id* attribute can thus be uniquely referenced by it. The *id* attribute of an element can be either assigned explicitly, or generated from the Java Class Path of the originating object if no name is given. While all elements that can be uniquely identified can carry the *id* attribute, only those that are independent on their context are required to do so. For example, tables are required to have an *id* since they retain meaning regardless of their location in the document, while table rows and cells can omit the attribute since their meaning depends on the parent element.

The name attribute *n* is simply the name assigned to the element, and it is used to distinguish an element from its immediate peers. In the example of a particular list, all items in that list will have different names to distinguish them from each other. Other lists in the document, however, can also contain items whose names will be different from each other, but identical to those in the first list. The *n* attribute of an element is therefore unique only in the scope of that element's parent and is used mostly for sorting purposes and special rendering of a certain class of elements, like, for example, all first items in lists, or all items named



"browse". The *n* attribute follows the same rules as *id* when determining whether or not it is required for a given element.

The last attribute in the standard triplet is *rend*. Unlike *id* and *n*, the *rend* attribute can consist of several space delimited values and is optional for all elements that can contain it. Its purpose is to provide a rendering hint from the middle layer component to the styling theme. How that hint is interpreted and whether it is used at all when provided, is completely up to the theme. There are several cases, however, where the content of the *rend* attribute is outlined in detail and its use is encouraged. Those cases are the emphasis element *hi*, the division element *div*, and the *list* element. Please refer to the Element Reference for more detail on these elements.

Structure-oriented markup

The final design pattern is the use of structure-oriented markup for content carried by the XML Document. Once generated by Cocoon, the Document contains two major types of information: metadata about the repository and its contents, and the actual content of the page to be displayed. A complete overview of metadata and content markup and their relationship to each other is given in the next section. An important thing to note here, however, is that the markup of the content is oriented towards explicitly stating structural relationships between the elements rather than focusing on the presentational aspects. This makes the markup used by the Document more similar to TEI or Docbook rather than HTML. For this reason, XSL templates are used by the themes to convert structural DRI markup to XHTML. Even then, an attempt is made to create XHTML as structural as possible, leaving presentation entirely to CSS. This allows the XML Document to be generic enough to represent any DSpace page without dictating how it should be rendered.

13.2.4. Schema Overview

The DRI XML Document consists of the root element *document* and three top-level elements that contain two major types of elements. The three top-level containers are *meta*, *body*, and *options*. The two types of elements they contain are metadata and content, carrying metadata about the page and the contents of the page, respectively. Figure 1 depicts the relationship between these six components.

Figure 1: The two content types across three major divisions of a DRI page. The *document* element is the root for all DRI pages and contains all other elements. It bears only one attribute, *version*, that contains the version number of the DRI system and the schema used to validate the produced document. At the time of writing the working version number is "1.1".

The *meta* element is a the top-level element under *document* and contains all metadata information about the page, the user that requested it, and the repository it is used with. It contains no structural elements, instead being the only container of metadata elements in a DRI Document. The metadata stored by the *meta* element is broken up into three major groups: *userMeta*, *pageMeta*, and *objectMeta*, each storing metadata information about their respective component. Please refer to the reference entries for more information about these elements.

The *options* element is another top-level element that contains all navigation and action options available to the user. The options are stored as items in list elements, broken up by the type of action they perform. The five types of actions are: browsing, search, language selection, actions that are always available, and actions that are context dependent. The two action types also contain sub-lists that contain actions available to users of varying degrees of access to the system. The *options* element contains no metadata elements and can only make use of a small set of structural elements, namely the *list* element and its children.

The last major top-level element is the *body* element. It contains all structural elements in a DRI Document, including the lists used by the *options* element. Structural elements are used to build a generic representation of a DSpace page. Any DSpace page can be represented with a combination of the structural elements, which will in turn be transformed by the XSL templates into another format. This is the core mechanism that allows DSpace XML UI to apply uniform templates and styling rules to all DSpace pages and is the fundamental difference from the JSP approach currently used by DSpace.

The *body* element directly contains only one type of element: *div*. The *div* element serves as a major division of content and any number of them can be contained by the *body*. Additionally, divisions are recursive, allowing *divs* to contain other *divs*. It is within these elements that all other structural elements are contained.



Those elements include tables, paragraph elements *p*, and lists, as well as their various children elements. At the lower levels of this hierarchy lie the character container elements. These elements, namely paragraphs *p*, table *cells*, lists *items*, and the emphasis element *hi*, contain the textual content of a DSpace page, optionally modified with links, figures, and emphasis. If the division within which the character class is contained is tagged as interactive (via the *interactive* attribute), those elements can also contain interactive form fields. Divisions tagged as interactive must also provide *method* and *action* attributes for its fields to use.

Figure 2: All the elements in the DRI schema. Note: This image is out-of-date, it does not reflect the changes between 1.0 and 1.1 such as *reference* and *referenceSet*.

13.2.5. Merging of DRI Documents

Having described the structure of the DRI Document, as well as its function in Manakin's Aspect chains, we now turn our attention to the one last detail of their use: merging two Documents into one. There are several situations where the need to merge two documents arises. In Manakin, for example, every Aspect is responsible for adding different functionality to a DSpace page. Since every instance of a page has to be a complete DRI Document, each Aspect is faced with the task of merging the Document it generated with the ones generated (and merged into one Document) by previously executed Aspects. For this reason rules exist that describe which elements can be merged together and what happens to their data and child elements in the process.

When merging two DRI Documents, one is considered to be the main document, and the other a feeder document that is added in. The three top level containers (*meta*, *body* and *options*) of both documents are then individually analyzed and merged. In the case of the *options* and *meta* elements, the children tags are taken individually as well and treated differently from their siblings.

The *body* elements are the easiest to merge: their respective *div* children are preserved along with their ordering and are grouped together under one element. Thus, the new *body* tag will contain all the *divs* of the main document followed by all the *divs* of the feeder. However, if two *divs* have the same *n* and *rend* attributes (and in case of an interactive *div* the same *action* and *method* attributes as well), those *divs* will be merged into one. The resulting *div* will bear the *id*, *n*, and *rend* attributes of the main document's *div* and contain all the *divs* of the main document followed by all the *divs* of the feeder. This process continues recursively until all the *divs* have been merged. It should be noted that two divisions with separate pagination rules cannot be merged together.

Merging the *options* elements is somewhat different. First, *list* elements under *options* of both documents are compared with each other. Those unique to either document are simply added under the new options element, just like *divs* under *body*. In case of duplicates, that is *list* elements that belong to both documents and have the same *n* attribute, the two *lists* will be merged into one. The new *list* element will consist of the main document's head element, followed *label-item* pairs from the main document, and then finally the *label-item* pairs of the feeder, provided they are different from those of the main.

Finally, the *meta* elements are merged much like the elements under *body*. The three children of *meta* - *userMeta*, *pageMeta*, and *objectMeta* - are individually merged, adding the contents of the feeder after the contents of the main.

13.2.6. Version Changes

The DRI schema will continue to evolve overtime as the needs of interface design require. The version attribute on the document will indicate which version of the schema the document conforms to. At the time Manakin was incorporated into the standard distribution of DSpace the current version was "1.1", however earlier versions of the Manakin interface may use "1.0".

Changes from 1.0 to 1.1

There were major structural changes between these two version numbers. Several elements were removed from the schema: *includeSet*, *include*, *objectMeta*, and *object*. Originally all metadata for objects were included in-line with the DRI document, this proved to have several problems and has been removed in version 1.1 of the DRI schema. Instead of including metadata in-line, external references to the metadata is included.



Thus, a *reference* element has been added along with *referenceSet*. These new elements operate like their counterparts in the previous version except referencing metadata contained on the *objectMeta* element they reference metadata in external files. The *repository* and *repositoryMeta* elements were also modified in a similar manner removing in-line metadata and referencing external metadata documents.

13.2.7. Element Reference

Element	Attributes (if required, noted)	Required	
BODY			
cell	cols		
id			
n			
rend			
role			
rows			
div	action	required for interactive behavior	
behaviorSensitivFields			
currentPage			
firstItemIndex			
id	required		
interactive			
itemsTotal			
lastItemIndex			
method	required for interactive		
n	required		
nextPage			
pagesTotal			
pageURLMask			
pagination			
previousPage			
rend			
DOCUMENT	version	required	
field	disabled		
id	required		
n	required		
rend			
required			
type	required		
figure	rend		
source			
target			
head	id		



n			
rend			
help			
hi	rend	required	
instance			
item	id		
n			
rend			
label	id		
n			
rend			
list	id	required	
n	required		
rend			
type			
META			
metadata	element	required	
language			
qualifier			
OPTIONS			
p	id		
n			
rend			
pageMeta			
params	cols		
maxlength			
multiple			
operations			
rows			
size			
reference	url	required	
repositoryID	required		
type			
referenceSet	id	required	
n	required		
orderBy			
rend			
type	required		
repository	repositoryID	required	
url	required		
repositoryMeta			
row	id		



n			
rend			
role	required		
table	cols	required	
id	required		
n	required		
rend			
rows	required		
trail	rend		
target			
userMeta	authenticated	required	
value	optionSelected		
optionValue			
type	required		
xref	target	required	

BODY

Top-Level Container

The *body* element is the main container for all content displayed to the user. It contains any number of *div* elements that group content into interactive and display blocks.

Parent

document

Children

div

(any)

Attributes

None

```

<document version=1.0>
  <meta> ... </meta>
  <body>
    <div n="division-example1"
id="XMLExample.div.division-example1">
      ...
    </div>
    <div n="division-example2" id="XMLExample.div.division-example2"
interactive="yes" action="www.DRItest.com"
method="post">
      ...
    </div>
    ...
  </body>
  <options> ... </options>
</document>

```

cell

Rich Text Container



Structural Element

The *cell* element contained in a *row* of a *table* carries content for that table. It is a character container, just like *p*, *item*, and *hi*, and its primary purpose is to display textual data, possibly enhanced with hyperlinks, emphasized blocks of text, images and form fields. Every *cell* can be annotated with a *role* (the most common being `header` and `data`) and can stretch across any number of rows and columns. Since cells cannot exist outside their container, *row*, their *id* attribute is optional.

Parent

row

Children

hi

(any)

xref

(any)

figure

(any)

field

(any)

Attributes

- **cols**: optional The number of columns the cell spans.
- **id**: optional A unique identifier of the element.
- **n**: optional A local identifier used to differentiate the element from its siblings.
- **rend**: optional A rendering hint used to override the default display of the element.
- **role**: optional An optional attribute to override the containing row's role settings.
- **rows**: optional The number of rows the cell spans.

```
<table n="table-example" id="XMLExample.table.table-example" rows="2"
cols="3">
  <row role="head">
    <cell cols="2">Data Label One and Two</cell> <cell>Data Label
Three</cell>
    ...
  </row>
  <row>
    <cell> Value One </cell> <cell> Value Two </cell> <cell> Value
Three </cell>
    ...
  </row>
  ...
</table>
```

div

Structural Element

The *div* element represents a major section of content and can contain a wide variety of structural elements to present that content to the user. It can contain paragraphs, tables, and lists, as well as references to artifact information stored in *artifactMeta*, *repositoryMeta*, *collections*, and *communities*. The *div* element is also



recursive, allowing it to be further divided into other divs. Divs can be of two types: interactive and static. The two types are set by the use of the *interactive* attribute and differ in their ability to contain interactive content. Children elements of divs tagged as interactive can contain form fields, with the *action* and *method* attributes of the *div* serving to resolve those fields.

Parent

body

div

Children

head

(zero or one)

pagination

(zero or one)

table

(any)

p

(any)

referenceSet

(any)

list

(any)

div

(any)

Attributes

- **action:** required for interactive The form action attribute determines where the form information should be sent for processing.
- **behavior:** optional for interactive The acceptable behavior options that may be used on this form. The only possible value defined at this time is *ajax* which means that the form may be submitted multiple times for each individual field in this form. Note that if the form is submitted multiple times it is best for the *behaviorSensitiveFields* to be updated as well.
- **behaviorSensitiveFields:** optional for interactive A space separated list of field names that are sensitive to behavior. These fields must be updated each time a form is submitted with out a complete refresh of the page (i.e. ajax).
- **currentPage:** optional For paginated divs, the *currentPage* attribute indicates the index of the page currently displayed for this div.
- **firstItemIndex:** optional For paginated divs, the *firstItemIndex* attribute indicates the index of the first item included in this div.
- **id:** required A unique identifier of the element.
- **interactive:** optional Accepted values are *yes*, *no*. This attribute determines whether the div is interactive or static. Interactive divs must provide *action* and *method* and can contain field elements.



- **itemsTotal**: optional For paginated divs, the itemsTotal attribute indicates how many items exist across all paginated divs.
 - **lastItemIndex**: optional For paginated divs, the lastItemIndex attribute indicates the index of the last item included in this div.
 - **method**: required for interactive Accepted values are `get`, `post`, and `multipart`. Determines the method used to pass gathered field values to the handler specified by the action attribute. The multipart method should be used for uploading files.
 - **n**: required A local identifier used to differentiate the element from its siblings.
 - **nextPage**: optional For paginated divs the nextPage attribute points to the URL of the next page of the div, if it exists.
 - **pagesTotal**: optional For paginated divs, the pagesTotal attribute indicates how many pages the paginated divs spans.
 - **pageURLMask**: optional For paginated divs, the pageURLMask attribute contains the mask of a url to a particular page within the paginated set. The destination page's number should replace the Unknown macro: {pagenum}
- string in the URL mask to generate a full URL to that page.
- **pagination**: optional Accepted values are `simple` and `masked`. This attribute determines whether the div is spread over several pages. Simple paginated divs must provide previousPage, nextPage, itemsTotal, firstItemIndex, lastItemIndex attributes. Masked paginated divs must provide currentPage, pagesTotal, pageURLMask, itemsTotal, firstItemIndex, lastItemIndex attributes.
 - **previousPage**: optional For paginated divs the previousPage attribute points to the URL of the previous page of the div, if it exists.
 - **rend**: optional A rendering hint used to override the default display of the element. In the case of the div tag, it is also encouraged to label it as either `primary` or `secondary`. Divs marked as primary contain content, while secondary divs contain auxiliary information or supporting fields.

```

<body>
  <div n="division-example"
  id="XMLExample.div.division-example">
    <head> Example Division </head>
    <p> This example shows the use of divisions. </p>
    <table ...>
      ...
    </table>
    <referenceSet ...>
      ...
    </referenceSet>
    <list ...>
      ...
    </list>
    <div n="sub-division-example"
  id="XMLExample.div.sub-division-example">
      <p> Divisions may be nested </p>
      ...
    </div>
    ...
  </div>
  ...
</body>

```

DOCUMENT

Document Root



The document element is the root container of an XML UI document. All other elements are contained within it either directly or indirectly. The only attribute it carries is the version of the Schema to which it conforms.

Parent

none

Children

meta

(one)

body

(one)

options

(one)

Attributes

- **version:** required Version number of the schema this document adheres to. At the time of writing the only valid version number is 1.0. Future iterations of this schema may increment the version number.

```
<document
  version="1.0">
  <meta>
    ...
  </meta>
  <body>
    ...
  </body>
  <options>
    ...
  </options>
</document>
```

field

Text Container

Structural Element

The *field* element is a container for all information necessary to create a form field. The required *type* attribute determines the type of the field, while the children tags carry the information on how to build it. Fields can only occur in divisions tagged as "interactive".

Parent

cell

p

hi

item

Children

params

(one)

help



(zero or one)

error

(any)

option

(any - only with the select type)

value

(any - only available on fields of type: select, checkbox, or radio)

field

(one or more - only with the composite type)

valueSet

(any)

Attributes

- **disabled:** optional Accepted values are `yes`, `no`. Determines whether the field allows user input. Rendering of disabled fields may vary with implementation and display media.
- **id:** required A unique identifier for a field element.
- **n:** required A non-unique local identifier used to differentiate the element from its siblings within an interactive division. This is the name of the field use when data is submitted back to the server.
- **rend:** optional A rendering hint used to override the default display of the element.
- **required:** optional Accepted values are `yes`, `no`. Determines whether the field is a required component of the form and thus cannot be left blank.
- **type:** required A required attribute to specify the type of value. Accepted types are:
 - **button:** A button input control that when activated by the user will submit the form, including all the fields, back to the server for processing.
 - **checkbox:** A boolean input control which may be toggled by the user. A checkbox may have several fields which share the same name and each of those fields may be toggled independently. This is distinct from a radio button where only one field may be toggled.
 - **file:** An input control that allows the user to select files to be submitted with the form. Note that a form which uses a file field must use the multipart method.
 - **hidden:** An input control that is not rendered on the screen and hidden from the user.
 - **password:** A single-line text input control where the input text is rendered in such a way as to hide the characters from the user.
 - **radio:** A boolean input control which may be toggled by the user. Multiple radio button fields may share the same name. When this occurs only one field may be selected to be true. This is distinct from a checkbox where multiple fields may be toggled.
 - **select:** A menu input control which allows the user to select from a list of available options.
 - **text:** A single-line text input control.
 - **textarea:** A multi-line text input control.



- **composite:** A composite input control combines several input controls into a single field. The only fields that may be combined together are: checkbox, password, select, text, and textarea. When fields are combined together they can possess multiple combined values.

```

<p>
  <hi> ... </hi>
  <xref> ... </xref>
  <figure> ... </figure>
  ...
  <field id="XMLExample.field.name" n="name" type="text "
required="yes">
  <params size="16" maxlength="32"/>
  <help>Some help text with <i18n>localized
content</i18n>.</help>
  <value type="raw">Default value goes
here</value>
  </field>
</p>

```

figure

Text Container

Structural Element

The *figure* element is used to embed a reference to an image or a graphic element. It can be mixed freely with text, and any text within the tag itself will be used as an alternative descriptor or a caption.

Parent

cell

p

hi

item

Children

none

Attributes

- **rend:** optional A rendering hint used to override the default display of the element.
- **source:** optional The source for the image, using either a URL or a pre-defined XML entity.
- **target:** optional A target for an image used as a link, using either a URL or an id of an existing element as a destination.

```

<p>
  <hi> ... </hi>
  ...
  <xref> ... </xref>
  ...
  <field> ... </field>
  ...
  <figure source="www.example.com/fig1"> This is a static image.
</figure> <figure source="www.example.com/fig1"
target="www.example.net">
  This image is also a link.
  </figure>
  ...
</p>

```



head

Text Container

Structural Element

The *head* element is primarily used as a label associated with its parent element. The rendering is determined by its parent tag, but can be overridden by the *rend* attribute. Since there can only be one *head* element associated with a particular tag, the *n* attribute is not needed, and the *id* attribute is optional.

Parent

div

table

list

referenceSet

Children

none

Attributes

- **id**: optional A unique identifier of the element
- **n**: optional A local identifier used to differentiate the element from its siblings
- **rend**: optional A rendering hint used to override the default display of the element.

```

<div ...>
  <head> This is a simple header associated with its div element.
</head>
  <div ...>
    <head rend="green"> This header will be green.
  </head>
  <p>
    <head> A header with <i18n>localized content</i18n>.
  </head>
  ...
  </p>
</div>
<table ...>
  <head> ...
</head>
  ...
</table>
<list ...>
  <head> ...
</head>
  ...
</list>
  ...
</body>

```

help

Text Container

Structural Element

The optional *help* element is used to supply help instructions in plain text and is normally contained by the *field* element. The method used to render the help text in the target markup is up to the theme.

Parent



field

Children

none

Attributes

None

```

<p>
  <hi> ... </hi>
  ...
  <xref> ... </xref>
  ...
  <figure> ... </figure>
  ...
  <field id="XMLExample.field.name" n="name" type="text"
  required="yes">
    <params size="16" maxlength="32" />
    <help>Some help text with <i18n>localized
  content</i18n>.</help>
  </field>
  ...
</p>

```

hi

Rich Text Container

Structural Element

The *hi* element is used for emphasis of text and occurs inside character containers like *p* and *list* item. It can be mixed freely with text, and any text within the tag itself will be emphasized in a manner specified by the required *rend* attribute. Additionally, *hi* element is the only text container component that is a rich text container itself, meaning it can contain other tags in addition to plain text. This allows it to contain other text containers, including other *hi* tags.

Parent

cell

p

item

hi

Children

hi

(any)

xref

(any)

figure

(any)

field

(any)

Attributes



- **rend**: required A required attribute used to specify the exact type of emphasis to apply to the contained text. Common values include but are not limited to "bold", "italic", "underline", and "emph".

```
<p>
  This text is normal, while <hi rend="bold">this text is bold and
  this text is <hi rend="italic">bold and
  italic.</hi></hi>
</p>
```

instance

Structural Element

The *instance* element contains the value associated with a form field's multiple instances. Fields encoded as an instance should also include the values of each instance as a hidden field. The hidden field should be appended with the index number for the instance. Thus if the field is "firstName" each instance would be named "firstName_1", "firstName_2", "firstName_3", etc...

Parent

field

Children

value

Attributes

None listed yet.

Example needed.

item

Rich Text Container

Structural Element

The *item* element is a rich text container used to display textual data in a list. As a rich text container it can contain hyperlinks, emphasized blocks of text, images and form fields in addition to plain text.

The *item* element can be associated with a label that directly precedes it. The Schema requires that if one *item* in a *list* has an associated *label*, then all other items must have one as well. This mitigates the problem of loose connections between elements that is commonly encountered in XHTML, since every item in particular list has the same structure.

Parent

list

Children

hi

(any)

xref

(any)

figure

(any)

field



(any)

list

(any)

Attributes

- **id**: optional A unique identifier of the element
- **n**: optional A non-unique local identifier used to differentiate the element from its siblings
- **rend**: optional A rendering hint used to override the default display of the element.

```

<list n="list-example"
  id="XMLExample.list.list-example">
  <head> Example List </head>
  <item> This is the first item
</item> <item> This is the second item with <hi ...>highlighted text</hi>,
<xref ...> a link</xref> and an <figure
...>image</figure>.</item>
  ...
<list n="list-example2"
  id="XMLExample.list.list-example2">
  <head> Example List </head>
  <label>ITEM ONE:</label>
  <item> This is the first item
</item>
  <label>ITEM TWO:</label>
  <item> This is the second item with <hi ...>highlighted
text</hi>, <xref ...> a link</xref> and an <figure
...>image</figure>.</item>
  <label>ITEM THREE:</label>
  <item> This is the third item with a <field ...> ... </field>
</item>
  ...
</list>
  <item> This is the third item in the list
</item>
  ...
</list>

```

label

Text Container

Structural Element

The *label* element is associated with an item and annotates that item with a number, a textual description of some sort, or a simple bullet.

Parent

item

Children

none

Attributes

- **id**: optional A unique identifier of the element
- **n**: optional A local identifier used to differentiate the element from its siblings
- **rend**: optional An optional rend attribute provides a hint on how the label should be rendered, independent of its type.



```

<list n="list-example"
  id="XMLExample.list.list-example">
  <head>Example List</head>
  <label>1</label>
  <item> This is the first item  </item>
  <label>2</label>
  <item> This is the second item with <hi ...>highlighted text</hi>,
<xref ...> a link</xref> and an <figure
...>image</figure>.</item>
  ...
  <list n="list-example2"
  id="XMLExample.list.list-example2">
    <head>Example Sublist</head>
    <label>ITEM
ONE:</label>
    <item> This is the first item  </item>
    <label>ITEM
TWO:</label>
    <item> This is the second item with <hi ...>highlighted
text</hi>, <xref ...> a link</xref> and an <figure
...>image</figure>.</item>
    <label>ITEM
THREE:</label>
    <item> This is the third item with a <field ...> ... </field>
</item>
    ...
  </list>
  <item> This is the third item in the list </item>
  ...
</list>

```

list

Structural Element

The *list* element is used to display sets of sequential data. It contains an optional *head* element, as well as any number of *item* and *list* elements. *Items* contain textual information, while sublists contain other *item* or *list* elements. An *item* can also be associated with a *label* element that annotates an item with a number, a textual description of some sort, or a simple bullet. The list type (ordered, bulleted, gloss, etc.) is then determined either by the content of *labels* on *items* or by an explicit value of the *type* attribute. Note that if *labels* are used in conjunction with any *items* in a list, all of the *items* in that list must have a *label*. It is also recommended to avoid mixing *label* styles unless an explicit type is specified.

Parent

div

list

Children

head

(zero or one)

label

(any)

item

(any)

list

(any)



Attributes

- **id**: required A unique identifier of the element
- **n**: required A local identifier used to differentiate the element from its siblings
- **rend**: optional An optional rend attribute provides a hint on how the list should be rendered, independent of its type. Common values are but not limited to:
 - **alphabet**: The list should be rendered as an alphabetical index
 - **columns**: The list should be rendered in equal length columns as determined by the theme.
 - **columns2**: The list should be rendered in two equal columns.
 - **columns3**: The list should be rendered in three equal columns.
 - **horizontal**: The list should be rendered horizontally.
 - **numeric**: The list should be rendered as a numeric index.
 - **vertical**: The list should be rendered vertically.
- **type**: optional An optional attribute to explicitly specify the type of list. In the absence of this attribute, the type of a list will be inferred from the presence and content of labels on its items. Accepted values are:
 - **form**: Used for form lists that consist of a series of fields.
 - **bulleted**: Used for lists with bullet-marked items.
 - **gloss**: Used for lists consisting of a set of technical terms, each marked with a *label* element and accompanied by the definition marked as an *item* element.
 - **ordered**: Used for lists with numbered or lettered items.
 - **progress**: Used for lists consisting of a set of steps currently being performed to accomplish a task. For this type to apply, each *item* in the list should represent a step and be accompanied by a *label* that contains the displayable name for the step. The *item* contains an *xref* that references the step. Also the *rend* attribute on the *item* element should be: `ïavailableï` (meaning the user may jump to the step using the provided *xref*), `ïunavailableï` (the user has not meet the requirements to jump to the step), or `ïcurrentï` (the user is currently on the step)
 - **simple**: Used for lists with items not marked with numbers or bullets.

```

<div ...>
  ...
  <list n="list-example"
id="XMLExample.list.list-example">
  <head>Example List</head>
  <item> ... </item>
  <item> ... </item>
  ...
  <list n="list-example2"
id="XMLExample.list.list-example2">
  <head>Example Sublist</head>
  <label> ... </label>
  <item> ... </item>
  <label> ... </label>
  <item> ... </item>
  <label> ... </label>
  <item> ... </item>
  ...
</list>
<label> ... </label>
<item> ... </item>

```



```
...
</list>
</div>
```

META

Top-Level Container

The *meta* element is a top level element and exists directly inside the *document* element. It serves as a container element for all metadata associated with a document broken up into categories according to the type of metadata they carry.

Parent

document

Children

userMeta

(one)

pageMeta

(one)

repositoryMeta

(one)

Attributes

None

```
<document version=1.0>
  <meta>
    <userMeta> ... </userMeta>
    <pageMeta> ... </pageMeta>
    <repositoryMeta> ... </repositoryMeta>
  </meta>
  <body> ... </body>
  <options> ... </options>
</document>
```

metadata

Text Container

Structural Element

The *metadata* element carries generic metadata information in the form of an attribute-value pair. The type of information it contains is determined by two attributes: *element*, which specifies the general type of metadata stored, and an optional *qualifier* attribute that narrows the type down. The standard representation for this pairing is *element.qualifier*. The actual metadata is contained in the text of the tag itself. Additionally, a *language* attribute can be used to specify the language used for the metadata entry.

Parent

userMeta

pageMeta

Children

none



Attributes

- **element**: required The name of a metadata field.
- **language**: optional An optional attribute to specify the language used in the metadata tag.
- **qualifier**: optional An optional postfix to the field name used to further differentiate the names.

```

<meta>
  <userMeta>
    <metadata element="identifier" qualifier="firstName"> Bob
  </metadata> <metadata element="identifier" qualifier="lastName"> Jones
  </metadata> <metadata ...> ...
  </metadata>
  ...
</userMeta>
<pageMeta>
  <metadata element="rights"
qualifier="accessRights">user</metadata> <metadata ...> ...
</metadata>
  ...
</pageMeta>
</meta>

```

OPTIONS

Top-Level Container

The *options* element is the main container for all actions and navigation options available to the user. It consists of any number of *list* elements whose items contain navigation information and actions. While any list of navigational options may be contained in this element, it is suggested that at least the following 5 lists be included.

Parent

document

Children

list

(any)

Attributes

None

```

<document version=1.0>

  <meta> Ö </meta>

  <body> Ö </body>

  <options>

    <list n="navigation-example1"
id="XMLExample.list.navigation-example1">

      <head>Example Navigation List 1</head>

      <item><xref target="/link/to/option">Option
One</xref></item>

      <item><xref target="/link/to/option">Option
two</xref></item>

      ...
    </list>
  </options>

```



```

    </list>

    <list n="navigation-example2"
    id="XMLExample.list.navigation-example2">

        <head>Example Navigation List 2</head>

        <item><xref target="/link/to/option">Option
    One</xref></item>

        <item><xref target="/link/to/option">Option
    two</xref></item>

        ...

    </list>

    ...

    </options>

</document>

```

p

Rich Text Container

Structural Element

The *p* element is a rich text container used by *divs* to display textual data in a paragraph format. As a rich text container it can contain hyperlinks, emphasized blocks of text, images and form fields in addition to plain text.

Parent

div

Children

hi

(any)

xref

(any)

figure

(any)

field

(any)

Attributes

- **id**: optional A unique identifier of the element.
- **n**: optional A local identifier used to differentiate the element from its siblings.
- **rend**: optional A rendering hint used to override the default display of the element.

```

<div n="division-example"
    id="XMLExample.div.division-example">

    <p> This is a regular paragraph.
    </p> <p> This text is normal, while <hi rend="bold">this text is bold

```



```

and this text is <hi rend="italic">bold and italic.</hi></hi>
</p> <p> This paragraph contains a <xref
target="/link/target">link</xref>, a static <figure
source="/image.jpg">image</figure>, and a <figure target=
"/link/target" source="/image.jpg">image link.</figure>
</p>
</div>

```

pageMeta

Metadata Element

The *pageMeta* element contains metadata associated with the document itself. It contains generic *metadata* elements to carry the content, and any number of *trail* elements to provide information on the user's current location in the system. Required and suggested values for *metadata* elements contained in *pageMeta* include but are not limited to:

- browser (suggested): The user's browsing agent as reported to server in the HTTP request.
- browser.type (suggested): The general browser family as derived from the browser metadata field. Possible values may include "MSIE" (for Microsoft Internet Explorer), "Opera" (for the Opera browser), "Apple" (for Apple web kit based browsers), "Gecko" (for Netscape, Mozilla, and Firefox based browsers), or "Lynx" (for text based browsers).
- browser.version (suggested): The browser version as reported by HTTP Request.
- contextPath (required): The base URL of the Digital Repository system.
- redirect.time (suggested): The time that must elapse before the page is redirected to an address specified by the redirect.url *metadata* element.
- redirect.url (suggested): The URL destination of a redirect page
- title (required): The title of the document/page that the user currently browsing. See the *metadata* and *trail* tag entries for more information on their structure.

ParentmetaChildrenmetadata (any)trail
(any)AttributesNone

```

<meta>
  <userMeta> ... </userMeta>
  <pageMeta>
    <metadata element="title">Example DRI
page</metadata>
    <metadata
element="contextPath"/>xmlui/</metadata>
    <metadata ...> ... </metadata>
    ...
    <trail source="123456789/6"> A bread crumb item
</trail>
    <trail ...> ... </trail>
    ...
  </pageMeta>
</meta>

```



params

Structural Component

The *params* element identifies extra parameters used to build a form field. There are several attributes that may be available for this element depending on the field type.

Parent

field

Children

none

Attributes

- **cols**: optional The default number of columns that the text area should span. This applies only to textarea field types.
- **maxlength**: optional The maximum length that the theme should accept for form input. This applies to text and password field types.
- **multiple**: optional yes/no value. Determine if the field can accept multiple values for the field. This applies only to select lists.
- **operations**: optional The possible operations that may be preformed on this field. The possible values are "add" and/or "delete". If both operations are possible then they should be provided as a space separated list. The "add" operations indicates that there may be multiple values for this field and the user may add to the set one at a time. The front-end should render a button that enables the user to add more fields to the set. The button must be named the field name appended with the string "_add", thus if the field's name is "firstName" the button must be called "firstName_add". The "delete" operation indicates that there may be multiple values for this field each of which may be removed from the set. The front-end should render a checkbox by each field value, except for the first, The checkbox must be named the field name appended with the string "_selected", thus if the field's name is "firstName" the checkbox must be called "firstName_selected" and the value of each successive checkbox should be the field name. The front-end must also render a delete button. The delete button name must be the field's name appended with the string "_delete".
- **rows**: optional The default number of rows that the text area should span. This applies only to textarea field types.
- **size**: optional The default size for a field. This applies to text, password, and select field types.

```
<p>
  <field id="XMLExample.field.name" n="name" type="text"
    required="yes">
    <params size="16"
      maxlength="32"/>
    <help>Some help text with <i18n>localized
      content</i18n>.</help>
    <default>Default value goes here</default>
  </field>
</p>
```

reference

Metadata Reference Element



reference is a reference element used to access information stored in an external metadata file. The *url* attribute is used to locate the external metadata file. The *type* attribute provides a short limited description of the referenced object's type.

reference elements can be both contained by *includeSet* elements and contain *includeSets* themselves, making the structure recursive.

Parent

referenceSet

Children

referenceSet

(zero or more)

Attributes

- **url**: required A url to the external metadata file.
- **repositoryIdentifier**: required A reference to the repositoryIdentifier of the repository.
- **type**: optional Description of the reference object's type.

```
<includeSet n="browse-list"
  id="XMLTest.includeSet.browse-list">
  <reference url="/metadata/handle/123/4/mets.xml"
    repositoryID="123" type="DSpace
    Item"/> <reference url="/metadata/handle/123/5/mets.xml"
    repositoryID="123" />
  ...
</includeSet>
```

referenceSet

Metadata Reference Element

The *referenceSet* element is a container of artifact or repository references.

Parent

div

reference

Children

head

(zero or one)

reference

(any)

Attributes

- **id**: required A unique identifier of the element
- **n**: required Local identifier used to differentiate the element from its siblings
- **orderBy**: optional A reference to the metadata field that determines the ordering of artifacts or repository objects within the set. When the Dublin Core metadata scheme is used this attribute should be the element.qualifier value that the set is sorted by. As an example, for a browse by title list, the value should be sortedBy=title, while for browse by date list it should be sortedBy=date.created



- **rend:** optional A rendering hint used to override the default display of the element.
- **type:** required Determines the level of detail for the given metadata. Accepted values are:
 - **summaryList:** Indicates that the metadata from referenced artifacts or repository objects should be used to build a list representation that is suitable for quick scanning.
 - **summaryView:** Indicates that the metadata from referenced artifacts or repository objects should be used to build a partial view of the referenced object or objects.
 - **detailList:** Indicates that the metadata from referenced artifacts or repository objects should be used to build a list representation that provides a complete, or near complete, view of the referenced objects. Whether such a view is possible or different from summaryView depends largely on the repository at hand and the implementing theme.
 - **detailView:** Indicates that the metadata from referenced artifacts or repository objects should be used to display complete information about the referenced object. Rendering of several references included under this type is up to the theme.

```

<div ...>
  <head> Example Division </head>
  <p> ... </p>
  <table> ... </table>
  <list>
    ...
  </list>
  <referenceSet n="browse-list"
id="XMLTest.referenceSet.browse-list" type="summaryView"
informationModel="DSpace">
  <head>A header for the includeset</head>
  <reference
url="/metadata/handle/123/34/mets.xml"/>
  <reference
url=" "metadata/handle/123/34/mets.xml/>
  </referenceSet>
  ...
</p>

```

repository

Metadata Element

The *repository* element is used to describe the repository. Its principal component is a set of structural metadata that carrier information on how the repository's objects under *objectMeta* are related to each other. The principal method of encoding these relationships at the time of this writing is a METS document, although other formats, like RDF, may be employed in the future.

Parent

repositoryMeta

Children

none

Attributes

- **repositoryID:** required A unique identifier assigned to a repository. It is referenced by the *object* element to signify the repository that assigned its identifier.
- **url:** required A url to the external METS metadata file for the repository.

```

<repositoryMeta>

```




```
<repository repositoryID="123456789"
url="/metadata/handle/1234/4/mets.xml" />
</repositoryMeta>
```

repositoryMeta

Metadata Element

The *repositoryMeta* element contains metadata references about the repositories used in the used or referenced in the document. It can contain any number of *repository* elements.

See the *repository* tag entry for more information on the structure of *repository* elements.

Parent

Meta

Children

repository

(any)

Attributes

None

```
<meta>
  <userMeta> ... </usermeta>
  <pageMeta> ... </pageMeta>
  <repositoryMeta>
    <repository repositoryIID="..." url="..."
  />
  </repositoryMeta>
</meta>
```

row

Structural Element

The row element is contained inside a *table* and serves as a container of *cell* elements. A required *role* attribute determines how the row and its cells are rendered.

Parent

table

Children

cell

(any)

Attributes

- **id**: optional A unique identifier of the element
- **n**: optional A local identifier used to differentiate the element from its siblings
- **rend**: optional A rendering hint used to override the default display of the element.



- **role:** required Indicates what kind of information the row carries. Possible values include "header" and "data".

```

<table n="table-example" id="XMLExample.table.table-example" rows="2"
cols="3">
  <row
role="head">
  <cell cols="2">Data Label One and
Two</cell>
  <cell>Data Label Three</cell>
  ...
</row> <row>
  <cell> Value One </cell>
  <cell> Value Two </cell>
  <cell> Value Three </cell>
  ...
</row>
  ...
</table>

```

table

Structural Element

The *table* element is a container for information presented in tabular format. It consists of a set of *row* elements and an optional *header*.

Parent

div

Children

head

(zero or one)

row

(any)

Attributes

- **cols:** required The number of columns in the table.
- **id:** required A unique identifier of the element
- **n:** required A local identifier used to differentiate the element from its siblings
- **rend:** optional A rendering hint used to override the default display of the element.
- **rows:** required The number of rows in the table.

```

<div n="division-example"
id="XMLExample.div.division-example">

```



```

    <table n="table1" id="XMLExample.table.table1" rows="2"
  cols="3">

    <row role="head">

      <cell cols="2">Data Label One and
  Two</cell>

      <cell>Data Label Three</cell>

      ...

    </row>

    <row>

      <cell> Value One </cell>

      <cell> Value Two </cell>

      <cell> Value Three </cell>

      ...

    </row>

    ...

  </table>
  ...
</div>

```

trail

Text Container

Metadata Element

The *trail* element carries information about the user's current location in the system relative of the repository's root page. Each instance of the element serves as one link in the path from the root to the current page.

Parent

pageMeta

Children

none

Attributes

- **rend:** optional A rendering hint used to override the default display of the element.
- **target:** optional An optional attribute to specify a target URL for a trail element serving as a hyperlink. The text inside the element will be used as the text of the link.

```

<pageMeta>

  <metadata element="title">Examlpe DRI
  page</metadata>

  <metadata
  element="contextPath">/xmlui/</metadata>

  <metadata ...> ... </metadata>

```



```

...
    <trail target="/myDSpace"> A bread crumb item pointing to a
page. </trail> <trail ...> ... </trail>
...
</pageMeta>

```

userMeta

Metadata Element

The *userMeta* element contains metadata associated with the user that requested the document. It contains generic *metadata* elements, which in turn carry the information. Required and suggested values for *metadata* elements contained in *userMeta* include but not limited to:

- identifier (suggested): A unique identifier associated with the user.
- identifier.email (suggested): The requesting user's email address.
- identifier.firstName (suggested): The requesting user's first name.
- identifier.lastName (suggested): The requesting user's last name.
- identifier.logoutURL (suggested): The URL that a user will be taken to when logging out.
- identifier.url (suggested): A url reference to the user's page within the repository.
- language.RFC3066 (suggested): The requesting user's preferred language selection code as describe by RFC3066
- rights.accessRights (required): Determines the scope of actions that a user can perform in the system. Accepted values are:
 - none: The user is either not authenticated or does not have a valid account on the system
 - user: The user is authenticated and has a valid account on the system
 - admin: The user is authenticated and belongs to the system's administrative group
See the *metadata* tag entry for more information on the structure of *metadata* elements.

ParentmetaChildrenmetadata (any)Attributes

- **authenticated**: required Accepted values are "yes", "no". Determines whether the user has been authenticated by the system.

```

<meta>
  <userMeta>
    <metadata element="identifier" qualifier="email">
      bobJones@tamu.edu
    </metadata>
    <metadata element="identifier" qualifier="firstName"> Bob
  </metadata>
    <metadata element="identifier" qualifier="lastName"> Jones
  </metadata>
    <metadata element="rights"

```



```

qualifier="accessRights">user</metadata>

  <metadata ...> ... </metadata>

  ...

  <trail source="123456789/6"> A bread crumb item
</trail>

  <trail ...> ... </trail>

  ...

</userMeta>

<pageMeta> ... </pageMeta>

</meta>

```

value

Rich Text Container

Structural Element

The value element contains the value associated with a form field and can serve a different purpose for various field types. The value element is comprised of two subelements: the raw element which stores the unprocessed value directly from the user or other source, and the interpreted element which stores the value in a format appropriate for display to the user, possibly including rich text markup.

Parent

field

Children

hi

(any)

xref

(any)

figure

(any)

Attributes

- **optionSelected**: optional An optional attribute for select, checkbox, and radio fields to determine if the value is to be selected or not.
- **optionValue**: optional An optional attribute for select, checkbox, and radio fields to determine the value that should be returned when this value is selected.
- **type**: required A required attribute to specify the type of value. Accepted types are:
 - **raw**: The raw type stores the unprocessed value directly from the user or other source.
 - **interpreted**: The interpreted type stores the value in a format appropriate for display to the user, possibly including rich text markup.
 - **default**: The default type stores a value supplied by the system, used when no other values are provided.

```
<p>
```



```

<hi> ... </hi>
<xref> ... </xref>
<figure> ... </figure>
<field id="XMLExample.field.name" n="name" type="text"
required="yes">
  <params size="16" maxlength="32"/>
  <help>Some help text with <i18n>localized
content</i18n>.</help>
  <value type="default">Author,
John</value>
</field>
</p>

```

xref

Text Container

Structural Element

The *xref* element is a reference to an external document. It can be mixed freely with text, and any text within the tag itself will be used as part of the link's visual body.

Parent

cell

p

item

hi

Children

none

Attributes

- **target:** required A target for the reference, using either a URL or an id of an existing element as a destination for the *xref*.

```

<p>
  <xref target="/url/link/target">This text is shown as a
  link.</xref>
</p>

```

13.3. History

13.3.1. Changes in DSpace 1.6.0

New Features

- DS-161 - Bulk Metadata Editing (Batch Metadata Editing)
- DS-194 - Give METS ingester configuration option to make use of collection templates
- DS-195 - Allow the primary bitstream to be set in the item importer / exporter
- DS-204 - New -zip option for item exporter and importer



- DS-205 - Creative Commons - option to set legal jurisdiction
- DS-228 - Community Admin XMLUI: Delegated Admins Patch
- DS-288 - Hide metadata from full item view
- DS-317 - Embargo feature
- DS-323 - ItemUpdate - new feature to batch update metadata and bitstreams
- DS-324 - Add support for OpenSearch syndicated search conventions
- DS-330 - Create new session on login / invalidate sessions on logout
- DS-359 - Add alternate file appender for log4j
- DS-377 - Add META tags identifying DSpace source version to Web UIs
- DS-388 - Item importer - new option to enable workflow notification emails

General Improvements

- DS-196 - METS exposed via OAI-PMH includes description.provenance information
- DS-201- handle.jar 6.2 needs adding to DSpace Maven repository
- DS-213 - IPAuthentication extended to allow negative matching
- DS-219 - Internal Server error - include login details of user
- DS-221 - XMLUI 'current activity' recognises Google Chrome as Safari
- DS-234 - Configurable passing of Javamail parameter settings
- DS-238 - Move item function in xmlui
- DS-252 - Interpolate variables in the Subject: line of email templates as well
- DS-261 - Community Admin JSPUI: porting of the DS-228 patch
- DS-270 - Make delegate admin permissions configurable
- DS-271 - Make the OAI DC crosswalk configurable
- DS-291 - README update for top level of dspace 1.6.0 package directory
- DS-297 - Refactor SQL source and Ant script to avoid copying Oracle versions over PostgreSQL
- DS-299 - Allow long values to be specified for the max upload request (for uploading files greater than 2Gb)
- DS-306 - Option to disable mailserver
- DS-307 - Offer access in AbstractSearch to QueryResults for subclasses
- DS-315 - Enhance readability of embedded metadata in html head
- DS-316 - Make SWORD app:accepts configurable
- DS-319 - Replace `/dspace/bin/dsrun org.dspace.browse.ItemCounter` with `/dspace/bin/itemcounter`
- DS-333 - Adjust SWORD ingest crosswalk to store bibliographic citation



- DS-356 - Antispam for suggest item feature
- DS-372 - New verbose option for [dspace]/bin/dspace cleanup script
- DS-382 - Add '*dc.creator*' to Author browse index by default
- DS-386 - Allow user to specify which `<dmdSec>` is used by the METS Ingestor when importing METS from Packager script
- DS-389 - Misleading label: "Submit to This Collection" is corrected
- DS-52 - Factor out common webapp installation - ID: 2042160

Bug fixes

- DS-44 - Monthly statistics skip first and last of month - ID: 2541435
- DS-114 - Links not working due to trailing white space in `dspace.url`
- DS-118 - File preview link during submission leads to page not found
- DS-128 - Anchor in submission doesn't work
- DS-156 - File description not available in XMLUI
- DS-191 - `metadataschemaregistry_seq` is not initialized correctly under Oracle
- DS-193 - OAI RDF crosswalk fails when DC value is null
- DS-197 - Deleting a primary bitstream does not clear the `primary_bitstream_id` on the bundle table
- DS-198 - File descriptions can not be removed/cleared in XMLUI
- DS-199 - SWORD module doesn't accept X-No-Op header (dry run)
- DS-200 - SWORD module requires the X-Packaging header
- DS-206 - Input form visibility restriction doesn't work properly
- DS-209 - `Context.java turnOffAuthorisationSystem()` can throw a NPE
- DS-212 - NPE thrown during Harvest of non-items when visibility restriction is enabled
- DS-216 - Migrating items that use additional metadata schemas causes an NPE
- DS-218 - Cannot add/remove email subscriptions from Profile page in XMLUI
- DS-222 - Email alerts due to internal errors are not sent, if context is missing
- DS-223 - Submission process show previous button in JSPUI also if the step is the first "visible" step
- DS-225 - *dc.description.provenance* - public display
- DS-226 - confirmation page of edit profile has an invalid link
- DS-227 - Values with double apos doesn't work in dropdown and list input type
- DS-231 - Missig file for index-init
- DS-232 - `DCPersonName` parses name incorrectly (fix included)
- DS-240 - Item `validityKey` not complete



- DS-242 - Special groups shown for logged in user rather than for user being examined
- DS-246 - Fix configurable browse parameter encoding (XMLUI)
- DS-248 - Missing admin column in community table in database-schema.sql - community admin patch
- DS-249 - sub-daily utility script does not pass arguments to Java (fix included)
- DS-250 - Invalid identifiers are not escaped
- DS-254 - Bitstream (and item-export) download service does not correctly sense authenticated user
- DS-255 - CompleteStep in submission LOSES SUBMISSION if an exception is thrown
- DS-256 - Item Export ignores metadata language qualifier
- DS-258 - Item View Thumbnails not displaying in XMLUI
- DS-260 - Template item some times has owningCollection filled and some times not
- DS-262 - Bug in DS-118, new patch included
- DS-265 - IndexBrowse dies fatally when confronting badly-formatted date
- DS-269 - Oracle JDBC connection string wrong in dspace.cfg - ID: 2722093
- DS-274 - Typo in XSL breaks rendering of dri:xref with class
- DS-275 - License files not listed on Item Summary page; XSL bug with patch
- DS-276 - Patch to fix spelling error in Exception page
- DS-280 - build.xml fails for ant versions below 1.7 (patch included)
- DS-281 - Invalid Link to "Go to DSpace Home" on Page Not Found
- DS-285 - Item and Bitstream pages do not provide Last-Modified HTTP header, nor recognize If-Modified-Since
- DS-290 - `[dspace]/exports` is not created during fresh install
- DS-303 - Export migrate option incorrectly removes non-handle identifier.uris
- DS-309 - Shibboleth default roles are applied also to anonymous user and user logged-in with other methods
- DS-310 - UTF-8 encoding in community and collection text
- DS-318 - JSPUI: Left over text in edit item about format
- DS-320 - `java.util.NoSuchElementException: Timeout waiting for idle object`
- DS-327 - SWORD temp upload directory missing trailing slash
- DS-328 - SWORD service documents do not include `atom:generator` element
- DS-337 - A bug related with adding new `-EPersons`
- DS-338 - Bitstream download allows caching of content that requires authorization to read
- DS-344 - Apostrophe in email address prevents `EPerson` from being selected
- DS-349 - Edit Item in admin UI does not allow setting Bitstream to an `Internal BitstreamFormat`



- DS-353 - Missing commits in XMLUI server-side javascript code.
- DS-354 - Make-handle-server configuration fails. New command created using dspace launcher.
- DS-370 - E Mail Sent On Item Export Error Message
- DS-373 - "Letter" links have broken URLs in 2nd-stage Browse
- DS-378 - XMLUI Submission Interface messes up in IE7 after an empty <hint> in input_forms.xml
- DS-379 - open-search in jsui won't return description.xml
- DS-381 - community and collection homepage
- DS-385 - Packager script is unable to import the same METS + DIM package that was exported
- DS-392 - Error messages in the submission do not disappear if e.g. one of the two errors are solved
- DS-393 - The issue date in the submission lowers each time the describe page is being displayed

13.3.2. Changes in DSpace 1.5.2

General Improvements

- The History System has been removed since DSpace 1.5. The `[dspace]/history` directory and it's contents can be completely removed if you so choose as it is non functional.

Bug fixes and smaller patches

- TBD

13.3.3. Changes in DSpace 1.5.1

General Improvements

- TBD

Bug fixes and smaller patches

- TBD

13.3.4. Changes in DSpace 1.5

General Improvements

- Highly configurable and theme-able new user interface (Manakin).
- Apache Maven-based modular build system.
- LNI (Lightweight Network Interface) service. Allows programmatic ingest of content via WebDAV or SOAP.
- SWORD (Simple Web-service Offering Repository Deposit): repository-standard ingest service using Atom Publishing Protocol.



- Highly configurable item web submission system. All submission steps are configurable not just metadata pages.
- Browse functionality allowing customisation of the available indexes via `dspace.cfg` and pluggable normalisation of the sort strings. Integration with both JSP-UI and XML-UI included.
- Extensible content event notification service.
- Generation of Google and HTML sitemaps

Bug fixes and smaller patches

- New options for ItemImporter to support bitstream permissions and descriptions.
- 1824710 Fix - Change in Creative Commons RDF.
- 1794700 Fix - Stat-monthly and stat-report-monthly
- 1566820 Patch - Authentication code moved to new `org.dspace.authenticate` package, add IP Auth
- 1670093 Patch - More stable metadata and schema registry import Option to generate community and collection "strength" as a batch job
- 1659868 Patch - Improved database level debugging
- 1620700 Patch - Add Community and Sub-Community to OAI Sets
- 1679972 Fix - OAIDCCrosswalk NPE and invalid character fix, also invalid output prevented
- 1549290 Fix - Suggest Features uses hard coded strings
- 1727034 Fix - Method `MetadataField.unique()` is incorrect for null values
- 1614546 Fix - Get rid of unused `mets_bitstream_id` column
- 1450491 Patch - i18n configurable multilingualism support
- 1764069 Patch - Replace "String" with "Integer" in `PreparedStatement` where needed
- 1743188 Patch - for Request #1145499 - Move Items
- 179196 Patch - Oracle SQL in Bitstream Checker
- 1751638 Patch - Set http disposition header to force download of large bitstreams
- 1799575 Patch - New `EPersonConsumer` event consumer
- 1566572 Patch - Item metadata in XHTML head elements
- 1589429 Patch - "Self-Named" Media Filters (i.e. `MediaFilter Plugins`) (updated version of this patch)
- 1888652 Patch - Statistics Rewritten In Java
- 1444364 Request - Metadata registry exporter
- 1221957 Request - Admin browser for withdrawn items
- 1740454 Fix - Concurrency
- 1552760 Fix - Submit interface looks bad in Safari
- 1642563 Patch - `bin/update-handle-prefix` rewritten in Java



- 1724330 Fix - Removes "null" being displayed in community-home.jsp
- 1763535 Patch - Alert DSpace administrator of new user registration
- 1759438 Patch - Multilingualism Language Switch - DSpace Header

13.3.5. Changes in DSpace 1.4.1

General Improvements

- Error pages now return appropriate HTTP status codes (e.g. 404 not found)
- Bad filenames in /bitstream/ URLs now result in 404 error – prevents infinite URL spaces confusing crawlers and bad "persistent" bitstream IDs circulating
- Prevent infinite URL spaces in HTMLServlet
- InstallItem no longer sets dc.format.extent, dc.format.mimetype; no longer sets default value for dc.language.iso if one is not present
- Empty values in drop-down submit fields are not added as empty metadata values
- API methods for searching epeople and groups
- Support stats from both 1.3 and 1.4
- [dspace]/bin/update-handle-prefix now runs index-all
- Remove cases of System.out from code executed in webapp
- Change "View Licence" to "View License" in Messages.properties
- dspace.cfg comments changed to indicate what default.language actually means
- HandleServlet and BitstreamServlet support If-Modified-Since requests
- Improved sanity-checking of XSL-based ingest crosswalks
- Remove thumbnail filename from alt-text
- Include item title in HTML title element
- Improvements to help prevent spammers and sploggers
- Make cleanup() commit outstanding work every 100 iterations
- Better handling where email send failed due to wrong address for new user
- Include robots.txt to limit bots navigating author, date and browse by subject pages
- Add css styles for print media
- RSS made more configurable and provide system-wide RSS feed, also moves text to Messages.properties
- Jar file updates (includes required code changes for DSIndexer and DSQuery and new jars fontbox.jar and serializer.jar)
- Various documentation additions and cleanups
- XHTML compliance improvements



- Move w3c valid xhtml boiler image into local repository
- Remove unnecessary Log4j Configuration in CheckerCommand
- Include Windows CLASSPATH in dsrun.bat

Bug fixes

- 1604037 - UIUtil.encodeBitstream() now correctly encodes URLs (no longer incorrectly substitutes '+' for spaces in non-query segment)
- 1592984 - Date comparisons strip time in org.dspace.harvest.Harvest
- 1589902 - Duplicate [field] checking error [on input-forms.xml]
- 1596952 - Collection Wizard create Template missing schema
- 1596978 - View unfinished submissions - collection empty
- 1588625 - Incorrect text on item mapper screen
- 1597805 - DIDL Crosswalk: wrong resource management
- 1605635 - NPE in Utils.java
- 1597504 - Search result page shows shortened query string
- 1532389 - Item Templates do not work for non-dc fields
- 1066771 - Metadata edit form dropping DC qualifier
- 1548738 - Multiple Metadata Schema, schema not shown on edit item page
- 1589895 - Not possible to add unqualified Metadata Field
- 1543853 - Statistics do not work in 1.4
- 1541381 - Browse-by-date and browse-by-title not working
- 1556947 - NullPointerException when no user selected to del/edit
- 1554064 - Fix exception handling for ClassCastException in BitstreamServlet
- 1548865 - Browse errors on withdrawn item
- 1554056 - Community/collection handle URL with / redirects to homepage
- 1571490 - UTF-8 encoded characters in licence
- 1571519 - UTF-8 in statistics
- 1544807 - Browse-by-Subject/Author paging mechanism broken
- 1543966 - "Special" groups inside groups bug
- 1480496 - Cannot turn off "ignore authorization" flag!
- 1515148 - Community policies not deleting correctly
- 1556829 - Docs mention old SiteAuthenticator class
- 1606435 - Workflow text out of context



- Fix for bitstream authorization timeout
- Fix to make sure cleanup() doesn't fail with NullPointerException
- Fix for removeBitstream() failing to update primary bitstream
- Fix for Advanced Search ignoring conjunctions for arbitrary number of queries
- Fix minor bug in Harvest.java for Oracle users
- Fix missing title for news editor page
- Small Messages.properties modification (change of DSpace copyright text)
- fix PDFBox tmp file issue
- Fix HttpServletRequest encoding issues
- Fix bug in TableRow toString() method where NPE is thrown if tablename not set
- Update DIDL license and change coding style to DSpace standard

13.3.6. Changes in DSpace 1.4

General Improvements

- Content verification through periodic checksum checking
- Support for branded preview image
- Add/replace Creative Commons in 'edit item' tool
- Customisable item listing columns and browse indices
- Script for updating handle prefixes (e.g. for moving from development to production)
- Configurable boolean search operator
- Controlled vocabulary patch to provide search on classification terms, and addition of terms during submission.
- Add 'visibility' element to input-forms.xml
- Browse by subject feature
- Log4J enhancement to use XML configuration
- QueryArgs class can support any number of fields in advanced search.
- Community names no longer have to be unique
- Enhanced Windows support
- Support for multiple (flat) metadata schemas
- Suggest an item page
- RSS Feeds
- Performance enhancements



- Stackable authentication methods
- Plug-in manager
- Pluggable SIP/DIP support and metadata crosswalks
- Nested groups of e-people
- Expose METS and MPEG-21 DIDL DIPs via OAI-PMH
- Configurable Lucene search analyzer (e.g. for Chinese metadata)
- Support for SMTP servers requiring authentication

Bug fixes

- 1358197 - Edit Item, empty DC fields not removable
- 1363633 - Submission step 1 fails when there are no collections
- 1255264 - Resource policy eperson value was set to wrong column
- 1380494 - Error deleting an item with multiple metadata schema support
- 1443649 - Cannot configure unqualified elements for advanced search index
- 1333687 - Browse-(title|date) fails on withdrawn item
- 1066713 - Two (sub)communities cannot have one name
- 1284055 - Two Communities of same name throws error
- 1035366 - Bitstream size column should be bigint
- 1352257 - Selecting a Group for GroupToGroup while Creating Collection
- 1352226 - Navigation and Sorting in Group List (Select Groups) fails
- 1348276 - Null in collection name causes OAI ListSets to fail
- 1160898 - dspace_migrate removes Date.Issued from prev published items
- 1261191 - Malformed METS metadata exported

13.3.7. Changes in DSpace 1.3.2

General Improvements

- DSpace UI XHTML/WAI compliant
- Configure metadata fields shown on simple item display
- Supervisor/workspace help documentation

Bug fixes

- Oracle compatibility fixes
- Item exporter now correctly exports metadata in UTF-8



- fixed to handle 'null' values passed in

13.3.8. Changes in DSpace 1.3.1

Bug fixes

- 1252153 - Error on fresh install

13.3.9. Changes in DSpace 1.3

General Improvements

- Initial i18n Support for JSPs - Note: the implementation of this feature required changes to almost all JSP pages
- LDAP authentication support
- Log file analysis and report generation
- Configurable item licence viewing
- Supervision order/collaborative workspace administrative tools
- Basic workspace for submissions in progress, with support for supervision
- SRB storage system option
- Updated handle server system
- Database optimisations
- Latest versions of Xerces, Xalan and OAICAT jars
- Various documentation additions and cleanups

Bug fixes

- 1161459 - ItemExporter fails with Too many open files
- 1167373 - Email date field not populated
- 1193948 - New item submit problem
- 1188132 - NullPointerException when Adding EPerson
- 1188016 - Cannot Edit an Eperson
- 1219701 - Unable to open unfinished submission
- 1206836 - community strengths not reflecting sub-community
- 1238262 - Submit UI nav/progress buttons no longer show progress
- 1238276 - Double quote problem in some fields in submit UI
- 1238277 - format support level not shown in "uploaded file" page
- 1242548 - Uploading non-existing files



- 1244743 - Bad lookup key for special case of DC Title in ItemTag.java
- 1245223 - Subscription Emailer fails
- 1247508 - Error when browsing item with no content/bitstream collections
- Set the content type in the HTTP header
- Fix issue where EPerson edit would not work due to form indexing (partial fix)
- POST handling in HTMLServlet
- Missing ContentType directives added to some JSPs
- Name dependency on Collection Admin and Submitter groups fixed
- Fixed OAI-PMH XML encoding

13.3.10. Changes in DSpace 1.2.2

General Improvements

- Customisable submission forms added
- Configurable number of index terms in Lucene for full-text indexing
- Improved scalability in media filter
- Submit button on collection pages only appears if user has authorisation
- PostgreSQL 8.0 compatibility
- Search scope retention to improve browsing
- Community and collection strengths displayed
- Upgraded OAICat software

Bug fixes

- Fix for Oracle too many cursors problem.
- Fix for UTF-8 encoded searches in advanced search.
- Fix for handling "\" in bitstream names.
- Fix to prevent delete of "unknown" bitstream format
- Fix for ItemImport creating new handles for replaced items

Changes in JSPs

- *collection-home.jsp* changed
- *community-home.jsp* changed
- *community-list.jsp* changed
- *home.jsp* changed



- *dspace-admin/list-formats.jspchanged*
- *dspace-admin/wizard-questions.jspchanged*
- *search/results.jspchanged*
- *submit/cancel.jspchanged*
- *submit/change-file-description.jspchanged*
- *submit/choose-file.jspchanged*
- *submit/complete.jspchanged*
- *submit/creative-commons.jspchanged*
- *submit/edit-metadata.jspnew*
- *submit/get-file-format.jspchanged*
- *submit/initial-questions.jspchanged*
- *submit/progressbar.jspchanged*
- *submit/review.jspchanged*
- *submit/select-collection.jspchanged*
- *submit/show-license.jspchanged*
- *submit/show-uploaded-file.jspchanged*
- *submit/upload-error.jspchanged*
- *submit/upload-file-list.jspchanged*

13.3.11. Changes in DSpace 1.2.1

General Improvements

- Oracle support added
- Thumbnails in item view can now be switched off/on
- Browse and search thumbnail options
- Improved item importer
 - can now import to multiple collections
 - added --test flag to simulate an import, without actually making any changes
 - added --resume flag to try to resume the import in case the import is aborted
- Configurable fields for the search index
- Script for transferring items between DSpace instances
- Sun library JARs (JavaMail, Java Activation Framework and Servlet) now included in DSpace source code bundle



Bug fixes

- A logo to existing collection can now be added. Fixes SF bug #1065933
- The community logo can now be edited. Fixes SF bug #1035692
- MediaFilterManager doesn't 'touch' every item every time. Fixes SF bug #1015296
- Supported formats help page, set the format support level to "known" as default
- Fixed various database connection pool leaks

Changed JSPs

- *collection-homechanged*
- *community-homechanged*
- *display-itemchanged*
- *dspace-admin/confirm-delete-collectionmoved to tools/ and changed*
- *dspace-admin/confirm-delete-communitymoved to tools/ and changed*
- *dspace-admin/edit-collectionmoved to tools/ and changed*
- *dspace-admin/edit-communitymoved to tools/ and changed*
- *dspace-admin/indexchanged*
- *dspace-admin/upload-logochanged*
- *dspace-admin/wizard-basicinfochanged*
- *dspace-admin/wizard-default-itemchanged*
- *dspace-admin/wizard-permissionschanged*
- *dspace-admin/wizard-questionschanged*
- *help/formats.htmlremoved*
- *help/formatschanged*
- *indexchanged*
- *layout/navbar-adminchanged*

13.3.12. Changes in DSpace 1.2

General Improvements

- Communities can now contain sub-communities
- Items may be included in more than one collection
- Full text extraction and searching for MS Word, PDF, HTML, text documents
- Thumbnails displayed in item view for items that contain images



- Configurable MediaFilter tool creates both extracted text and thumbnails
- Bitstream IDs are now persistent - generated from item's handle and a sequence number
- Creative Commons licenses can optionally be added to items during web submission process

Administration

- If you are logged in as administrator, you see admin buttons on item, collection, and community pages
- New collection administration wizard
- Can now administer collection's submitters from collection admin tool
- Delegated administration - new 'collection editor' role - edits item metadata, manages submitters list, edits collection metadata, links to items from other collections, and can withdraw items
- Admin UI moved from /admin to /dspace-admin to avoid conflict with Tomcat /admin JSPs
- New EPerson selector popup makes Group editing much easier
- 'News' section is now editable using admin UI (no more mucking with JSPs)

Import/Export/OAI

- New tool that exports DSpace content in AIPs that use METS XML for metadata (incomplete)
- OAI - sets are now collections, identified by Handles ('safe' with /, : converted to _)
- OAI - contributor.author now mapped to oai_dc:creator

Miscellaneous

- Build process streamlined with use of WAR files, symbolic links no longer used, friendlier to later versions of Tomcat
- MIT-specific aspects of UI removed to avoid confusion
- Item metadata now rendered to avoid interpreting as HTML (displays as entered)
- Forms now have no-cache directive to avoid trouble with browser 'back' button
- Bundles now have 'names' for more structure in item's content

JSP file changes between 1.1 and 1.2

This list generated with `cvs -Q rdiff -s -r dspace-1_1 dspace` and a sprinkling of perl.

- Changed: dspace/jsp/collection-home.jsp
- Changed: dspace/jsp/community-home.jsp
- Changed: dspace/jsp/community-list.jsp
- Changed: dspace/jsp/display-item.jsp
- Changed: dspace/jsp/index.jsp
- Changed: dspace/jsp/home.jsp



- Changed: dspace/jsp/styles.css.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/authorize-advanced.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/authorize-collection-edit.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/authorize-community-edit.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/authorize-item-edit.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/authorize-main.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/authorize-policy-edit.jsp
- Moved to dspace-admin: dspace/jsp/admin/collection-select.jsp
- Moved to dspace-admin: dspace/jsp/admin/community-select.jsp
- Moved to dspace-admin: dspace/jsp/admin/confirm-delete-collection.jsp
- Moved to dspace-admin: dspace/jsp/admin/confirm-delete-community.jsp
- Moved to dspace-admin: dspace/jsp/admin/confirm-delete-dctype.jsp
- Moved to dspace-admin: dspace/jsp/admin/confirm-delete-eperson.jsp
- Moved to dspace-admin: dspace/jsp/admin/confirm-delete-format.jsp
- Moved to dspace/jsp/tools: dspace/jsp/admin/confirm-delete-item.jsp
- Moved to dspace/jsp/tools: dspace/jsp/admin/confirm-withdraw-item.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/edit-collection.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/edit-community.jsp
- Moved to dspace/jsp/tools and changed: dspace/jsp/admin/edit-item-form.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/eperson-browse.jsp
- Moved to dspace-admin: dspace/jsp/admin/eperson-confirm-delete.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/eperson-edit.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/eperson-main.jsp
- Moved to dspace/jsp/tools and changed: dspace/jsp/admin/get-item-id.jsp
- Moved to dspace/jsp/tools and changed: dspace/jsp/admin/group-edit.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/group-eperson-select.jsp
- Moved to dspace/jsp/tools and changed: dspace/jsp/admin/group-list.jsp
- Moved to dspace-admin: dspace/jsp/admin/index.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/item-select.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/list-communities.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/list-dc-types.jsp
- Removed: dspace/jsp/admin/list-epeople.jsp



- Moved to dspace-admin and changed: dspace/jsp/admin/list-formats.jsp
- Moved to dspace/jsp/tools: dspace/jsp/admin/upload-bitstream.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/upload-logo.jsp
- Moved to dspace-admin: dspace/jsp/admin/workflow-abort-confirm.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/workflow-list.jsp
- Changed: dspace/jsp/browse/authors.jsp
- Changed: dspace/jsp/browse/items-by-author.jsp
- Changed: dspace/jsp/browse/items-by-date.jsp
- Changed: dspace/jsp/browse/no-results.jsp
- New: dspace-admin/eperson-deletion-error.jsp
- New: dspace/jsp/dspace-admin/news-edit.jsp
- New: dspace/jsp/dspace-admin/news-main.jsp
- New: dspace/jsp/dspace-admin/wizard-basicinfo.jsp
- New: dspace/jsp/dspace-admin/wizard-default-item.jsp
- New: dspace/jsp/dspace-admin/wizard-permissions.jsp
- New: dspace/jsp/dspace-admin/wizard-questions.jsp
- Changed: dspace/jsp/components/contact-info.jsp
- Changed: dspace/jsp/error/internal.jsp
- New: dspace/jsp/help/formats.jsp
- Changed: dspace/jsp/layout/footer-default.jsp
- Changed: dspace/jsp/layout/header-default.jsp
- Changed: dspace/jsp/layout/navbar-admin.jsp
- Changed: dspace/jsp/layout/navbar-default.jsp
- Changed: dspace/jsp/login/password.jsp
- Changed: dspace/jsp/mydspace/main.jsp
- Changed: dspace/jsp/mydspace/perform-task.jsp
- Changed: dspace/jsp/mydspace/preview-task.jsp
- Changed: dspace/jsp/mydspace/reject-reason.jsp
- Changed: dspace/jsp/mydspace/remove-item.jsp
- Changed: dspace/jsp/register/edit-profile.jsp
- Changed: dspace/jsp/register/inactive-account.jsp
- Changed: dspace/jsp/register/new-password.jsp



- Changed: `dspace/jsp/register/registration-form.jsp`
- Changed: `dspace/jsp/search/advanced.jsp`
- Changed: `dspace/jsp/search/results.jsp`
- Changed: `dspace/jsp/submit/cancel.jsp`
- New: `dspace/jsp/submit/cc-license.jsp`
- Changed: `dspace/jsp/submit/choose-file.jsp`
- New: `dspace/jsp/submit/creative-commons.css`
- New: `dspace/jsp/submit/creative-commons.jsp`
- Changed: `dspace/jsp/submit/edit-metadata-1.jsp`
- Changed: `dspace/jsp/submit/edit-metadata-2.jsp`
- Changed: `dspace/jsp/submit/get-file-format.jsp`
- Changed: `dspace/jsp/submit/initial-questions.jsp`
- Changed: `dspace/jsp/submit/progressbar.jsp`
- Changed: `dspace/jsp/submit/review.jsp`
- Changed: `dspace/jsp/submit/select-collection.jsp`
- Changed: `dspace/jsp/submit/show-license.jsp`
- Changed: `dspace/jsp/submit/show-uploaded-file.jsp`
- Changed: `dspace/jsp/submit/upload-error.jsp`
- Changed: `dspace/jsp/submit/upload-file-list.jsp`
- Changed: `dspace/jsp/submit/verify-prune.jsp`
- New: `dspace/jsp/tools/edit-item-form.jsp`
- New: `dspace/jsp/tools/eperson-list.jsp`
- New: `dspace/jsp/tools/itemmap-browse.jsp`
- New: `dspace/jsp/tools/itemmap-info.jsp`
- New: `dspace/jsp/tools/itemmap-main.jsp`

13.3.13. Changes in DSpace 1.1.1

Bug fixes

- non-administrators can now submit again
- installations now preserve file creation dates, eliminating confusion with upgrades
- authorization editing pages no longer create null entries in database, and no longer handles them poorly (no longer gives blank page instead of displaying policies.)



- registration page Invalid token error page now displayed when an invalid token is received (as opposed to internal server error.) Fixes SF bug #739999
- eperson admin 'recent submission' links fixed for DSpaces deployed somewhere other than at / (e.g. /dspace).
- help pages Link to help pages now includes servlet context (e.g. '/dspace'). Fixes SF bug #738399.

Improvements

- *bin/dspace-info.pl* now checks jsp and asset store files for zero-length files
- *make-release-package* now works with SourceForge CVS
- eperson editor now doesn't display the spurious text 'null'
- item exporter now uses Jakarta's cli command line arg parser (much cleaner)
- item importer improvements:
 - now uses Jakarta's cli command line arg parser (much cleaner)
 - imported items can now be routed through a workflow
 - more validation and error messages before import
 - can now use email addresses and handles instead of just database IDs
 - can import an item to a collection with the workflow suppressed

13.3.14. Changes in DSpace 1.1

- Fixed various OAI-related bugs; DSpace's OAI support should now be correct. Note that harvesting is now based on the new Item 'last modified' date (as opposed to the Dublin Core *date.available* date.)
- Fixed Handle support--DSpace now responds to naming authority requests correctly.
- Multiple bitstream stores can now be specified; this allows DSpace storage to span several disks, and so there is no longer a hard limit on storage.
- Search improvements:
 - New fielded searching UI
 - Search results are now paged
 - Abstracts are indexed
 - Better use of Lucene API; should stop the number of open file handles getting large
- Submission UI improvements:
 - now insists on a title being specified
 - fixed navigation on file upload page
 - citation & identifier fields for previously published submissions now fixed
- Many Unicode fixes to the database and Web user interface
- Collections can now be deleted



- Bitstream descriptions (if available) displayed on item display page
- Modified a couple of servlets to handle invalid parameters better (i.e. to report a suitable error message instead of an internal server error)
- Item templates now work
- Fixed registration token expiration problem (they no longer expire.)