DSPACE

DSpace Manual

Software Version 1.5

May 2008

TABLE OF CONTENTS

INTRODUCTION

This is the documentation for DSpace V1.5 – as of May 2008.

Documentation for other versions of DSpace is included with the source code.

DSpace is an open source software platform that enables organizations to:
- *Capture* and describe digital material using a submission workflow module, or a variety of programmatic ingest options
- *Distribute* an organization'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 personnel. 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 DSpace Wiki(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:

- A list of DSpace resources (http://wiki.dspace.org/DspaceResources) - Web sites, mailing lists etc.
- Technical FAQ (http://wiki.dspace.org/TechnicalFaq)
- Guidelines for contributing back to DSpace (http://wiki.dspace.org/ContributionGuidelines)

The DSpace website, www.dspace.org, list DSpace related events, news stories, a list of current users, documentation, training material, service providers and information to help organizations promote and develop their DSpace repository.

The DSpace community has several very active mailing lists where you can meet other DSpace users and developers, learn more about the community and the software, get answers to technical questions, and hear of upcoming events:

- The dspace-general list (http://sourceforge.net/account/login.php?return_to=%2Fmailarchive%2Fforum.php%3Fforum_id%3D13580) – Includes general questions, announcements, and discussions about non-technical aspects of DSpace, including services, policies, legal issues, features and functions, etc.
- The dspace-tech e-mail list on SourceForge (http://sourceforge.net/mailarchive/forum.php?forum_id=135800) is the recommended place to ask questions, since a large 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 dspace-devel e-mail list (https://sourceforge.net/mailarchive/forum.php?forum_id=39921), for those developing with the DSpace with a view to contributing to the core DSpace code

The DSpace Foundation was established in 2007 to be a legal entity to hold and protect the copyright to the DSpace code base so it would remain free and open to all under the BSD license agreement. The role of the foundation is to promote the use of DSpace and lead the collaboration of the open source software platform to enable organizations to provide permanent access to their digital works. More information about the Foundation can be found on the website.

FUNCTIONAL OVERVIEW

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

Data Model

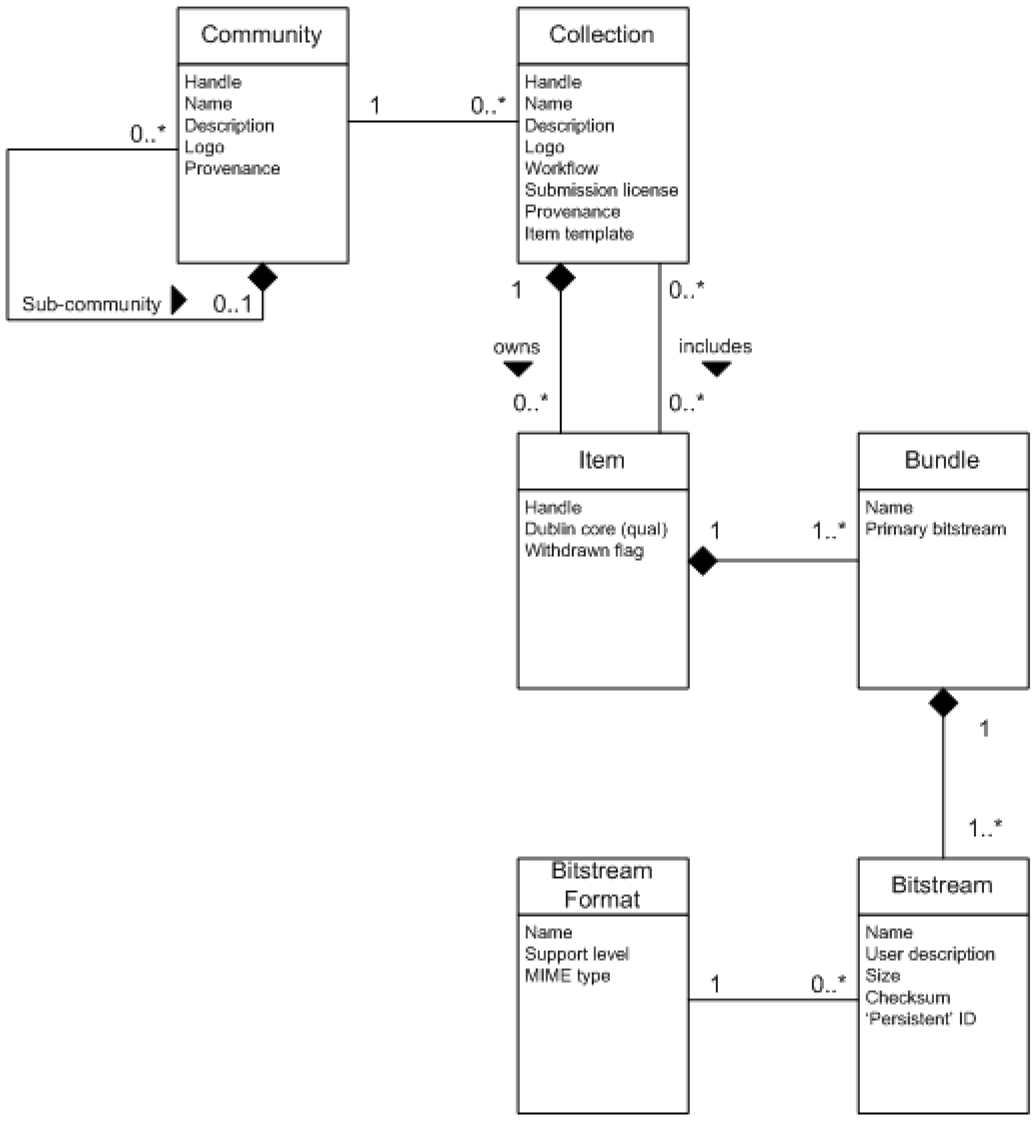

Figure 1: **Data Model Diagram**

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 organized 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 Creative Commons license (http://www.creativecommons.org/)) 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:

MIT Libraries' Definitions of Bitstream Format Support Levels:

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.

Objects in the DSpace Data Model

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

PLUGIN MANAGER

The Plugin Manager 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.

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 Library Application Profile (http://www.dublincore.org/documents/library-application-profile/) set of elements and qualifiers is provided by default. The set of elements and qualifiers used by MIT Libraries (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.

PACKAGER PLUGINS

PACKAGERS are software modules that translate between DSpace Item objects and a self-contained external representation, or "package". A PACKAGE INGESTER 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 IMS Content Package (http://www.imsglobal.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.

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.

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.

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

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.

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.

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:

Community

ADD/REMOVE	add or remove collections or sub-communities

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
COLLECTION_ADMIN	collection admins can edit items in a collection, withdraw items, and map other items into this collection.

Item

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

Bundle

ADD/REMOVE	add or remove bitstreams to a bundle

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.

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.

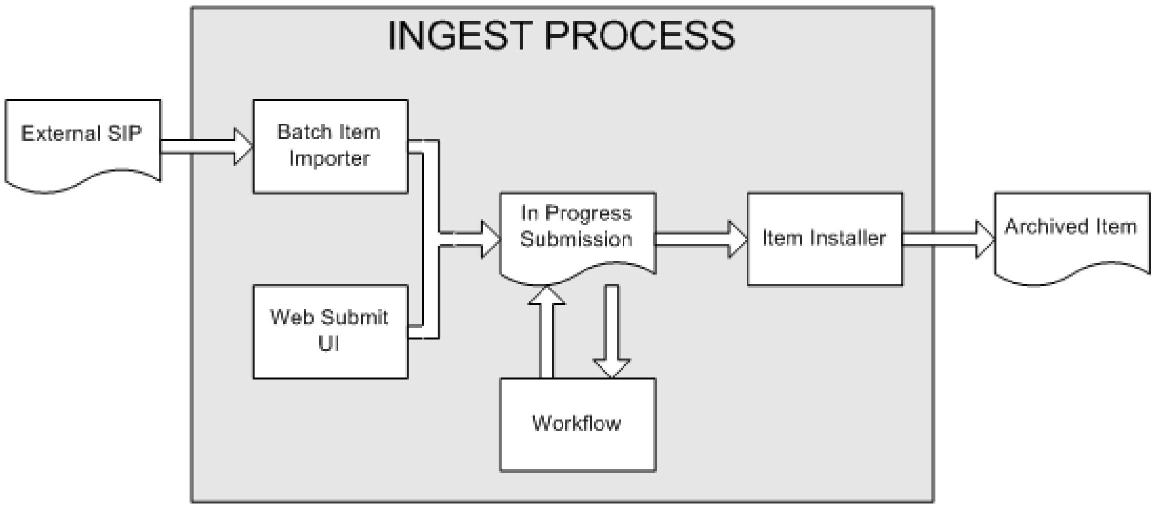

Figure 2 – Ingest Process

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 in 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. (The History system is also invoked, but provenance is easier for us to access at the moment.)

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.

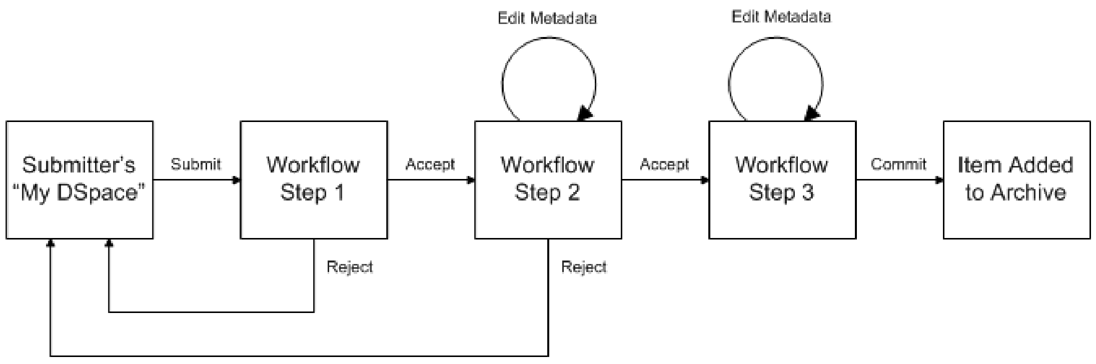

Figure 3 - Submission Workflow in DSpace

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.

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.

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 CNRI Handle System 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 CNRI's Handle Resolver plug-in (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.

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.

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 SRB (Storage Resource Broker) (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.

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 though 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 Lucene (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.

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 style sheets 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.

OAI SUPPORT

The Open Archives Initiative(http://www.openarchives.org/) has developed a protocol for metadata harvesting (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 OAICat (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.

OPENURL SUPPORT

DSpace supports the OpenURL protocol (http://www.sfxit.com/OpenURL/) from SFX (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.

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 their website (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.

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.

HISTORY

While provenance information in the form of prose is very useful, it is not easily programmatically manipulated. The History system captures a time-based record of significant changes in DSpace, in a manner suitable for later 'refactoring' or repurposing.

Currently, the History subsystem is explicitly invoked when significant events occur (e.g., DSpace accepts an item into the archive). The History subsystem then creates RDF data describing the current state of the object. The RDF data is modeled using Harmony/ABC (http://www.metadata.net/harmony/), an ontology for describing temporal-based data, and stored in the file system. Some simple indices for unwinding the data are available.

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.

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.

STATISTICS

Various statistical report about the contents and use of your system can be automatically generated by the system. These are generated by analyzing DSpace's log files. Statistics can be broken down monthly. At the time of publishing this feature is not available when using a Manakin based web interface.

The report includes data such as:

- A customizable 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
- Customizable 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.

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.

V1.5: INSTALLATION

I. 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.

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.

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

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 following location:
http://maven.apache.org/download.html

APACHE ANT 1.6.2 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

RELATIONAL DATABASE: (POSTGRESQL OR ORACLE).

PostgreSQL 7.3 or greater

PostgreSQL can be downloaded from the following location: http://www.postgresql.org/

It's 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
```

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

Verify 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"
```

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]*/config/server.xml: 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" />
```

OVERVIEW OF INSTALL OPTIONS

With the advent of a new Apache Maven 2 (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.

Its 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. Most projects should find the non source release package adequate for their installation needs.

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 commiters 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 the dspace-xmlui, 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 source module
    - dspace-lni/ - Lightweight Network Interface source module
    - dspace-sword/ - SWORD (Simple Web-service Offering Repository Deposit) deposit service source module
    - pom.xml - DSpace Parent Project definition

Both approaches provide you with the same control over how DSpace builds itself (especially in terms of adding completely custom/3rd-party DSpace "modules" you wish to use). Both methods allow you the ability to create more complex user interface "overlays" in Maven. An interface "overlay" allows you to only manage your local custom code (in your local CVS or SVN), and automatically download the rest of the interface code from the maven central repository whenever you build DSpace. This reduces the amount of out-of-the-box DSpace interface code maintained in your local CVS/SVN.

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. **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. **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. **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 section of this document. NOTE THAT THE *[dspace-source]* AND *[dspace]* DIRECTORIES ARE ALWAYS SEPARATE!

III. UNIX-LIKE 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 latest DSpace release (http://sourceforge.net/projects/dspace/) and unpack it. Although there are two available releases (dspace-1.x-release.zip and dspace-1.x-src-release.zip), 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.zip release.

   ```
   unzip dspace-1.x-release.zip
   ```

   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.

3. Database Setup

   Postgres:

A PostgreSQL 8.1-404 jdbc3 driver is configure as part of the default DSpace build. You no longer need to copy any postgres jars to get postgres installed.

i. Create a dspace database, owned by the dspace PostgreSQL user:

ii. createuser -U postgres -d -A -P dspace
createdb -U dspace -E UNICODE dspace

iii. Enter a password for the DSpace database. (This isn't the same as the dspace user's UNIX password.)

4. 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}
   See the dspace.cfg file for examples.

5. 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.)

6. As the dspace UNIX user, generate the DSpace installation package in the *[dspace-source]*/dspace/target/dspace-[version].dir/ 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

7. As the dspace UNIX user, initialize the DSpace database and install DSpace to *[dspace]*:

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

   Note: 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 common problems section.

8. 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 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"/>
```

   Alternatively, you could copy only the DSpace Web application(s) you wish to use from [DSPACE]/webapps to the appropriate directory in your Tomcat/Jetty/Resin installation. For example:

   - cp -r *[dspace]*/webapps/jspui *[tomcat]*/webapps
   - cp -r *[dspace]*/webapps/oai *[tomcat]*/webapps

9. Create an initial administrator account:

   - *[dspace]*/bin/create-administrator

10. 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:

- o JSP USER INTERFACE - (e.g.) http://dspace.myu.edu:8080/jspui
- o XML USER INTERFACE (AKA. MANAKIN) - (e.g.) http://dspace.myu.edu:8080/xmlui
- o 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.

IV. 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.

'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
```

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.

Note that Perl (http://www.perl.com/download.csp) needs to be installed in order to run the statistical reports.

For information on customizing the output of this see configuring system statistical reports.

MULTILINGUAL INSTALLATION

In order to deploy a multilingual version of DSpace you have to configure two parameters in *[dspace]*/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 section in the configuration documentation.

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.

   1. 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
      ```

   2. 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
      ```

   3. 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
      ```

   4. 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.

   1. 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.

2. 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.

3. 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
```

4. 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 Apache HTTPD (http://httpd.apache.org/) as your primary HTTP server, you can have it forward requests to the Tomcat servlet container http://tomcat.apache.org/) via Apache Jakarta Tomcat Connector (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 Apache SSL/TLS Encryption (http://httpd.apache.org/docs/2.0/ssl/) for information about using mod_ssl (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 Apache Jakarta Tomcat Connector (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.

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 the central CNRI Handle site (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.

The Handle server code is included with the DSpace code in *[dspace]*/lib/handle.jar. **Note:** The latest version of the handle.jar file is not included in the release due to licensing conditions changing between the provided version and later versions. It is recommended you read the new license conditions (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]*/lib with the new version and rebuild your war files.

A script exists to create a simple Handle configuration - simply run *[dspace]*/bin/make-handle-config after you've set the appropriate parameters in dspace.cfg. You can also create a Handle configuration directly by following the installation instructions on handle.net (http://www.handle.net/hs_manual_18jan02/server_manual_2.html), but with these changes:

- Instead of running:

  ```
  java -cp /hs/bin/handle.jar net.handle.server.SimpleSetup /hs/svr_1
  ```

as directed in the Handle Server Administration Guide (http://hdl.handle.net/4263537/4093), you should run *[dspace]*/bin/dsrun net.handle.server.SimpleSetup *[dspace]*/handle-server ensuring that *[dspace]*/handle-server matches whatever you have in dspace.cfg for the handle.dir property.

- 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.

Whichever approach you take, start the Handle server with *[dspace]*/bin/start-handle-server, as the DSpace user. 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.

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.

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

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]/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
```

V. WINDOWS INSTALLATION

PRE-REQUISITE SOFTWARE

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

- Java SDK 1.5 (http://java.sun.com/) or later (standard SDK is fine, you don't need J2EE)
- PostgreSQL 8.x for Windows (http://www.postgresql.org/ftp/) OR Oracle 9 or later (http://www.oracle.com/database/).

    - If you install PostgreSQL, it's recommended to select to install the pgAdmin III tool

- Apache Ant 1.6.2 or later (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.

- Jakarta Tomcat 5.x or later (http://tomcat.apache.org/)

- Apache Maven 2.0.8 or later (http://maven.apache.org/)

INSTALLATION STEPS

1. Download the DSpace source from SourceForge (http://sourceforge.net/projects/dspace) and untar it (WinZip (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.

VI. CHECKING YOUR INSTALLATION

TODO

VII. 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 DSpace bug tracker (http://sourceforge.net/tracker/?atid=119984&group_id=19984&func=browse) 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.

VIII. 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(AbstractJdbc1
Connection.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 killing 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 killing the process, and stopping and restarting Tomcat.

UPDATING A DSPACE INSTALLATION

Upgrading an existing DSpace instance successfully requires a considerable amount of effort and time on the part of a “DSpace Application Manager”. While some guidance has been provided by developers, DSpace as a community driven project, does not provide a full featured an “automated” upgrade process. The upgrade process is very “hands on”. It cannot be stressed enough that it is important to preserve as much as your current installation as you reasonably can.

MAINTAIN YOU CHANGES IN AN SCM

One of the best ways to maintain your changes to DSpace (and currently most popular in the community) is to keep your local configuration and customizations in a source code management system like Subversion or CVS as a means to:

1. Maintain your changes to configuration over the long term.
2. Ease the process of merging those changes as new releases come out
3. Provide a solid foundation for your operations and production maintenance

Using an SCM as the initial building block in your “Staging, Development and Production” operations provides you with a mechanism to track your group’s development, deploy to staging services for testing and finally role out a production grade release.

With the 1.5 release of DSpace, we have worked to separate the configuration and customization of DSpace away from the core libraries that are maintained by the developers. The dspace-1.5.X-release distribution cleanly separates these two areas by only shipping that which you will need to maintain your customizations (i.e. just the [dspace-source]/dspace directory).

There is another distribution (dspace-1.5.X-source-release) that includes all the code projects as well. Please consider this larger release only as a means to access the existing source for the purpose of reviewing and making your own modifications to those libraries in your local [dspace-source]/dspace configuration. This distribution takes significantly longer to build.

We highly recommend that you download this distribution and start your upgrade process by preserving it in a local SCM and migrating your local changes onto it rather than trying to merge these changes into your live installation directly. Based on the experience of several application managers, maintaining your DSpace will become significantly less stressful if you take this approach.

UPDATING 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 programmatic 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. **Download dspace-1.5.X-release.zip and preserve it in a local SCM**

   Get the new DSpace 1.5 source code either as a download from SourceForge (http://sourceforge.net/project/showfiles.php?group_id=19984) or check it out directly from the SVN code repository (http://sourceforge.net/svn/?group_id=19984). If you downloaded DSpace do not unpack it on top of your existing installation. This downloaded release of dspace will be refered to as [dspace-source] from here on.

2. **Apply any local customizations to about [dspace-source]**

   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 in the DSpace documentation for more information.

   If you have customized sourcecode, these will need to be handled on a case by case basis. More than likely, if it JSP-UI related changes, you can copy the modified class into [dspace-source]/dspace/modules/jspui/src/main/java/ making sure to preserve the package names.

3. **Update dspace.cfg**

   This document originally suggested copying new settings out of the 1.5 [dspace-source]/dspace/config/dspace.cfg into your existing [dspace]/config/dspace.cfg. However, the magnitude of differences is enormous and you would be much better served to bring your customizations to 1.4.2 into the 1.5 configuration, it will be much simpler and your minor updates via 1.5.x maintenance releases will be easier in the future.

   - Note all your customizations to [dspace]/config/dspace.cfg out and edit the [dspace-source]/dspace/config/dspace.cfg to contain these settings.

   - If you do have a merge tool like Eclipse, Try to work with it and the Comparison viewer to identify those configurations.

There are two important areas of the configuration that you need to pay attention to. The first is that Stackable Authentications package name has changed from "org.dspace.eperson" to "org.dspace.authentication" You want to verify these settings are correct in any custom authentications you may have configured.

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

Second, the Browse system has been replaced with a new and quite differently configured. Youe will want to make sure your settings here are adjusted/customized appropriately.

```
# 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
```

4. **Package your dspace-1.5.X-release plus your customizations**

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.

5. **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

Basically, anything in "bin, "config", "etc", "lib" for configuration and "assetstore", "search", "reports", "logs" and "history" for data/state.

6. **Stop Tomcat**

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

7. **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_142-15.sql contains the commands necessary to upgrade your database schema on oracle.

8. **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.

   The new statistics scripts will be installed on top of the existing scripts, it is important to get the details of your configuration present in those script prior to running the installation.

   Then edit your statistics configuration file with the start details. Add the follwing to [dspace-source]/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

9. **Update DSpace**

   After assuring you do have your *[dspace]/config, bin, etc,* directories properly backed up and all your configuration settings properly migrated into the new dspace.cfg, you will need to reinitialize your configurations to contain any new config from the source.

   ```
   cp [dspace-source]/dspace/config/xmlui.xconf [dspace]/config/xmlui.xconf
   cd [dspace-source]/dspace/target/dspace-1.5-build.dir/;
   ant init_config
   ```

   Next 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 update
   ```

10. **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
    ```

11. **Deploy webapplications**

    Adjust your Servlet Container configuration to use “[dspace]/webapps” as its webapplication directory (I.E. by editing the tomcat server.xml)

    Otherwise, copy the webapplications files from your [dspace]/webapps directory to the subdirectory of your servlet container (e.g. Tomcat):

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

12. **Restart Tomcat**

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

UPDATING FROM EARLIER VERIONS

For information on how to update from previous versions of DSpace, please visit the DSpace website, at:
http://www.dspace.org/index.php?option=com_content&task=view&id=156

CONFIGURATION AND CUSTOMIZATION

There are a number of ways in which DSpace can be configured and/or customized:

- Altering the configuration files in *[dspace]*/config
- Creating a new XMLUI (Manakin) theme to change the look-and-feel of the repository
- Creating modified versions of the JSP pages for local changes in the JSPUI interface
- Implementing a custom 'plug-in' class -- for example, an 'authenticator' class, so that user authentication in the Web UI can be adapted and integrated with any existing mechanisms your organization might use, or a 'media filter' to generate thumbnails or extract full text from a new file format
- Editing the source code

Of these methods, only the last is likely to cause any headaches; if you update the DSpace source code directly, particularly core class files in org.dspace.* or org.dspace.storage.*, it may make applying future updates difficult. Before doing this, it is strongly recommended that you e-mail the DSpace developer community (http://wiki.dspace.org/DspaceResources) to find out the best way to proceed, and the best way to implement your change in a way that can be contributed back to DSpace (http://wiki.dspace.org/HowToContribute) for everyone's benefit.

GENERAL CONFIGURATION

These are general configuration options that apply to the core of DSpace regardless of which interface you are using (JSPUI or XMLUI).

THE DSPACE.CFG CONFIGURATION PROPERTIES FILE

The primary way of configuring DSpace is to edit the dspace.cfg. You'll definitely have to do this before you can operate DSpace properly. dspace.cfg contains basic information about a DSpace installation, including system path information, network host information, and other things like site name.

The default dspace.cfg is a good source of information, and contains comments for all properties. It's 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
```

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
```

DSPACE.CFG MAIN PROPERTIES (NOT COMPLETE)

Property	**Example Values**	**Notes**
dspace.dir	/dspace	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. assetstore.dir.
dspace.url	http://dspace.myu.edu http://dspacetest.myu.edu:8080	Main URL at which DSpace Web UI webapp is deployed. Include any port number, but do not include a trailing '/'
dspace.hostname	dspace.myu.edu	Fully qualified hostname; do not include port number
dspace.name	DSpace at My University	Short and sweet site name, used throughout Web UI, e-mails and elsewhere (such as OAI protocol)
config.template.foo	/opt/othertool/cfg/foo	When install-configs is run, the file *[dspace]*/config/templates/foo file will be filled out with values from dspace.cfg and copied to the value of this property, in this example /opt/othertool/cfg/foo. See here for more information.

plugin.sequence.org.dspace.authenticate.AuthenticationMethod	org.dspace.eperson.X509Authentication, org.dspace.authenticate.PasswordAuthentication	Comma-separated list of classes implementing the org.dspace.authenticate.AuthenticationMethod interface, which make up the authentication stack. Authentication methods are called on in the order listed.
authentication.x509.keystore.path	/tomcat/conf/keystore	Path to Java keystore containing Client CA's certificiate for client X.509 certificates (Optional; only needed if X.509 user authentication is used.)
authentication.x509.keystore.password	changeit	Password to Java keystore configured above in authentication.x509.keystore.path
handle.prefix	1721.1234	The Handle prefix for your site, see the Handle section
assetstore.dir	/bigdisk/store	The location in the file system for asset (bitstream) store number zero. This should be a directory for the sole use of DSpace.
assetstore.dir.n	/anotherdisk/store1	The location in the file system of asset (bitstream) store number n. When adding additional stores, start with 1 (assetstore.dir.1 and count upwards. Always leave asset store zero (assetstore.dir). For more details, see the Bitstream Storage

		section.
assetstore.incoming	1	The asset store number to use for storing new bitstreams. For example, if assetstore.dir.1 is /anotherdisk/store1, and assetstore.incoming is 1, new bitstreams will be stored under /anotherdisk/store1. A value of 0 (zero) corresponds to assetstore.dir. For more details, see the Bitstream Storage section.
srb.xxx srb.xxx.n	/zone/home/user.domain	The sets of SRB access parameters (see dspace.cfg) if one or more SRB accounts are used. The srb.xxx set would correspond to asset (bitstream) store number zero. The srb.xxx.n set would correspond to asset (bitstream) store number n. For more details, see the Bitstream Storage section.
webui.submit.enable-cc	true	Enable the Creative Commons license step in the submission process for the JSPUI interface. Submitters are given an opportunity to select a Creative Commons license to accompany the Item. Creative Commons licenses govern the use of the

		content. For more details, see the Creative Commons website (http://creativecommons.org/).
default.locale	en	The default Locale your Installation is working with.
webui.browse.thumbnail.maxheight	80	Determines the maximum height of any system generated thumbnails.
webui.browse.thumbnail.maxwidth	80	Determines the maximum width of any system generated thumbnails.
webui.feed.enable	true	Set the value of this property to true to enable RSS feeds. If false, feeds will not be generated, and the feed links will not appear.
webui.feed.cache.size	100	If caching is desired, set the value of this property to a positive number, which represents the total number of feeds kept in the cache at one time, for all communities and collections. A value of 0 disables caching, and the feed is generated on demand for each request.
webui.cache.age	48	This property specifies the age in hours that a cache web feed may remain valid for. A value of 0 will force a check with each request.
webui.feed.formats	rss_1.0,rss_2.0	The RSS feature

		supports several different syndication formats.
webui.feed.localresolve	false	By default, the RSS feed will return global handle server-based URLs to items, collections and communities (e.g. http://hdl.handle.net/123456789/1). This means if you have not registered your DSpace installation with the CNRI Handle Server (e.g. development or testing instance) the URLs returned by the feed will return an error if accessed. Setting webui.feed.localresolve = true will result in the RSS feed returning localized URLs (e.g. http://myserver.myorg/handle/123456789/1). If webui.feed.localresolve is set to false or not present the default global handle URL form is used.
webui.feed.item.title	dc.title	Specify which metadata field you want to be displayed as an item's title in the RSS feed.
webui.feed.item.date	dc.date.issued	Specify which metadata field you want to be displayed as an item's date in the RSS feed.
webui.feed.item.description	dc.title, dc.creator,dc.description.abstract	Specify which metadata fields should be displayed in an item's description field in the

		RSS feed. You can specify as many fields as you wish here.

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

Whenever you edit dspace.cfg, you should then run *[dspace]*/bin/install-configs 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.

CONFIGURING LUCENE SEARCH INDEXES

Search Indexes can be configured via the dspace.cfg file. This allows institutions to choose which DSpace metadata fields are indexed by Lucene.

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

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

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

- <id> is an incremental number to distinguish each search index entry
- <search field> is an identifier for the search field this index will correspond to
- <metadata field> is the DSpace metadata field to be indexed

So in the example above, search.indexes1, 2 and 6 are configured as the author search field. The author index is created by Lucene indexing all contributor, creator and description.statementofresponsibility medatata fields.

After changing the configuration, run index-all to recreate the indexes.

NOTE: While the indexes are created, this only affects the search results and has no effect on the search components of the user interface. To add new search capability (e.g. to add a new search category to the Advanced Search) requires local customisation to the user interface.

BROWSE CONFIGURATION

The browse indices for DSpace can be extensively configured. This section of the configuration allows you to take control of the indices you wish to browse on, and how you wish to present the results. This configuration is broken down into several parts: defining the indices, defining the fields upon which users can sort results, defining truncation for potentially long fields (e.g. author lists), 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.

DEFINING THE INDICES

The form is:

```
webui.browse.index.<n> = <index name> : \
                         <schema prefix>.<element>[.<qualifier>|.*] : \
                         (date | title | text) : \
                         (full | single) \
```

index name

The name by which the index will be identified. This may be used in later configuration or to locate the message key for this index.

<schema prefix>.<element>[.<qualifier>|.*]

The metadata field declaration for the field to be indexed. This will be something like dc.date.issued or dc.contributor.* or dc.title.

(date | title | text)

This refers to the datatype of the field:

- date: the index type will be treated as a date object
- title: the index type will be treated like a title, which will include a link to the item page
- text: the index type will be treated as plain text. If single mode is specified then this will link to the full mode list

(full | single)

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 webui.itemlist.columns; "single" will be a single list of only the indexed term.

If you are customising this list beyond the default you will need to insert the text you wish to appear in the navigation and on link and buttons describing the browse index into the Messages.properties file. The system uses parameters of the form:

```
browse.type.<index name>
```

The Index numbers denoted by <n> must start from 1 and increment by 1 continuously thereafter. Deviation from this rule will cause an error during installation or during configuration update

This is an example configuration, as it appears by default in dspace.cfg.

```
webui.browse.index.1 = dateissued:dc.date.issued:date:full
webui.browse.index.2 = author:dc.contributor.*:text:single
webui.browse.index.3 = title:dc.title:title:full
webui.browse.index.4 = subject:dc.subject.*:text:single
webui.browse.index.5 = dateaccessioned:dc.date.accessioned:date:full
```

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 webui.itemlist.columns

The format is:

```
webui.browse.sort-option.<n> = <option name> : \
                        <schema prefix>.<element>[.<qualifier>|.*] : \
                        (date | text)
```

option name

The name by which the sort option will be identified. This may be used in later configuration or to locate the message key for this index.

<schema prefix>.<element>[.<qualifier>|.*]

The metadata field declaration for the field to be sorted on. This will be something like dc.title or dc.date.issued.

(date | text)

This refers to the datatype of the field:

- date: the sort type will be treated as a date object
- text: the sort type will be treated as plain text.

This is the example configuration as it appears in the default dspace.cfg:

```
webui.browse.sort-option.1 = title:dc.title:text
webui.browse.sort-option.2 = date:dc.date.issued:date
```

AUTHOR (MULTIPLE METADATA VALUE) DISPLAY

(Note: this section actually applies to any field with multiple values, but authors are the defined case)

You can define which field is the author (or editor, or other repeated field) which this configuration will deal with thus:

```
webui.browse.author-field = dc.contributor.*
```

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 data type text, otherwise the functionality will be overriden by the specific data type features.

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

```
webui.browse.author-limit = <n>
```

Where <n> is an integer number of values to be displayed. Use -1 for unlimited (default).

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.

The form is:

```
webui.browse.link.<n> = <index name>:<display column metadata>
```

This should associated the name of one of the browse indices (webui.browse.index.n) with a metadata field listed in webui.itemlist.columns above. If this condition is not fulfilled, cross-linking will not work. Note also that cross-linking only works for metadata fields not tagged as title in webui.itemlist.columns.

The following example shows the default in dspace.cfg which links author names to lists of their publications:

```
webui.browse.link.1 = author:dc.contributor.*
```

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

First define the sort name (from webui.browse.sort-option) to use for # displaying recent submissions. For example:

```
recent.submissions.sort-option = dateaccessioned
```

Define how many recent submissions should be displayed at any one time, for example:

```
recent.submissions.count = 5
```

Now we need to set up the processors that the PluginManager will load to actually perform the recent submissions query on the relevant pages.

Tell the community and collection pages that we are using the Recent Submissions code

```
plugin.sequence.org.dspace.plugin.CommunityHomeProcessor = \
        org.dspace.app.webui.components.RecentCommunitySubmissions

plugin.sequence.org.dspace.plugin.CollectionHomeProcessor = \
        org.dspace.app.webui.components.RecentCollectionSubmissions
```

This is already configured by default in dspace.cfg so there should be no need for you to worry about it

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 indices defines the author browse (or equivalent) so that the mapper can list authors' items for mapping

Define the the index name (from webui.browse.index) to use for displaying items by author

```
itemmap.author.index = author
```

So if you change the name of your author browse field, you will also need to update this configuration.

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.

```
        #### Media Filter / Format Filter plugins (through PluginManager) ####

        #Names of the enabled MediaFilter or FormatFilter plugins
        filter.plugins = PDF Text Extractor, HTML Text Extractor, \
        Word Text Extractor, JPEG Thumbnail
        # to enable branded preview: remove last line above, and uncomment 2 lines
below
        #  Word Text Extractor, JPEG Thumbnail, \
        #  Branded Preview JPEG

        #Assign 'human-understandable' names to each filter
        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

        #Configure each filter's input format(s)
        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.inputFormats =
GIF, JPEG, image/png
```

The enabled Media/Format Filters are named in the filter.plugins field above.

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 vaild input formats which each filter can be applied to. 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. More information is provide below in Creating a new Media/Format Filter

WORDING OF E-MAIL MESSAGES

Sometimes DSpace automatically sends e-mail messages to users, for example to inform them of a new workflow 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

THE METADATA AND BITSTREAM FORMAT REGISTRIES

The *[dspace]*/config/registries directory contains three XML files. These are used to load the INITIAL contents of the Metadata Schema Registry, Dublin Core Metadata registry and Bitstream Format 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 allready 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 difficult. The changes made via admin UI are not reflected in the XML files.

METADATA SCHEMA REGISTRY

The default metadata schema in DSpace is Dublin Core, so it is distributed with a single entry in the source XML file for that namespace. If you wish to add more schemas you can do this in one of two ways. Via the DSpace admin UI you may define new Metadata Schemas, edit existing schemas and move elements between 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.adminster.SchemaImporter -f [xml file]
```

The XML file should be structured as follows:

```
<metadata-schemas>
        <schema>
                <name>[schema name]</name>
                <namespace>http://myu.edu/some/namespace</namespace>
        </schema>
</metadata-schemas>
```

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.adminster.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 administation 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

Note: Deleting a format will cause any existing bitstreams of this format to be reverted to the unknown bitstream format.

THE DEFAULT SUBMISSION LICENSE

For each submitted item, a license must be granted. The license will be stored along with the item in the bundle LICENSE in order to keep the information under which terms an items has been published.

You may define a license for each collection seperately, when creating/editing a collection. If no collection specific license is defined, the default license is used.

The default license can be found in *[dspace]*/config/default.license and can be edited via the dspace-admin interface.

DSpace comes with a demo license, which you must adopt to your institutional needs and the legal regulations of your country.

If in doubt, contact the law department of your institution.

POSSIBLE POINTS IN A LICENSE

Note, that this is no legal advice, just some starting thoughts for creating you own license.

- Non-exclusive or exclusive right to

    - capture and store
    - distribute
        - worldwide
        - restricted (e.g. institutional wide
    - translate
    - transform to other formats or mediums

    without changing the content

- Make sure no rights (copyright or any other) are violated by this publication
- In case the type of submission (e.g. thesis) needs approval, make sure it is the final and approved version.
- Distinguish between the document itself and the metadata
- Point out that the license granted and the information who granted it will be stored.

SUBMISSION CONFIGURATION

Instructions for customizing and configuring the Item Submission user interface for either the JSP-UI or XML-UI are contained in the separate Customizing and Configuring Submission User Interface page.

XMLUI INTERFACE CUSTOMIZATIONS (MANAKIN)

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 parameters which are specific to the XMLUI interface based upon the Cocoon framework.

XMLUI CONFIGURATION PROPERTIES

There are several options effecting how the XMLUI user interface for DSpace operates. Listed below are the major elements and their description, refere to the dspace.cfg file itself for the exhaustive list of configuration parameters.

Property	**Example Values**	**Notes**
xmlui.supportedLocales	en, de	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 seperated list of Locales. All types of Locales country, country_language, country_language_variant Note that that if the approprate files are not present (i.e. Messages_XX_XX.xml) then Manakin will fall back through to a more general language.
xmlui.user.registration	true	Determine if new users should be allowed to register.This parameter is usefull in congunction with shibboleth where you want to disallow registration because shibboleth will automatically register the user. Default value is true.
xmlui.user.editmetadata	true	Determine if users should be allowed to edit their own metadata. This parameter is usefull in congunction with shibboleth where you want to disable the user's ability to edit their metadata because it came from Shibboleth. Default value is true.
xmlui.user.loginredirect	/profile	Determine where a user is directed after logging into the system. Leave this parameter blank or undefined to direct users to the repository homepage, or "/profile" for the user's profile, or another reasonable choice is "/submissions" to see if the user has any tasks awaiting their attention. The default is the repository home page.
xmlui.force.ssl	true	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. Default value is false.
xmlui.theme.allowoverrides	false	If set to true, then 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"
xmlui.bundle.upload	ORIGINAL, METADATA, THUMBNAIL, LICENSE, CC_LICENSE	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 & write) on the bundle then that bundle will not be shown to the user as an option.
xmlui.community-list.render.full	True	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.
xmlui.community-list.cache	12 hours	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.
xmlui.bitstream.mods	true	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 either MODS.xml. If this option is set to true and the bitstream is present then it is made available to the theme for display.
xmlui.bitstream.mets	true	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 either METS.xml. If this option is set to true and the bitstream is present then the stored METS file is merged with the METS file generated by Manakin for each item. Thus if the bitstream contains a dmdSec then there will be two dmdSec one from the bitstream and another generated from the Dublin Core stored inside the database.

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* & *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.xmap 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.

MULTILINGUAL SUPPORT

The XMLUI user interface supports multiple languages through the use of internationalization catalogues as defined by the Cocoon Internationalization Transformer (http://cocoon.apache.org/2.1/userdocs/i18nTransformer.html). Each catalogue contains the translation of all user-displayed strings into a particular language or variant. Each catalogue 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"

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 & deploy DSpace as described in the installation portion of the manual, and ensure the theme has been installed as described in the previous section "Configuring Themes and Aspects".

JSPUI INTERFACE CUSTOMIZATIONS

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 parameters which are specific to the JSPUI interface.

JSPUI CONFIGURATION PROPERTIES

There are many options effecting how the JSP-based user interface for DSpace operates. Listed below are the major elements and their description, refere to the dspace.cfg file itself for the exhaustive list of configuration parameters.

Property	**Example Values**	**Notes**
webui.mydspace.showgroupmemberships	false	Determine if the MyDSpace page should list all groups a user belongs too. The default behavior, if omitted, is false.
webui.strengths.show	true	Determine if communities and collections should display item counts when listed. The default behavior, if omitted, is true.
webui.licence_bundle	true	Setting this parameter to ture will result in a hyperlink being rendered on the item View page that points to the item's licence.
webui.browse.thumbnail.show	true	Determine if thumbnails should be displayed on browe-by pages and item view pages when available. The default behavior, if omitted, is false.
webui.browse.thumbnail.linkbehaviour	item	Direct the target when a thumbnail is clicked. Currently the values item and bitstream

		are allowed. If this configuration item is not set, or set incorrectly, the default is item.
webui.suggest.enable	true	Set the value of this property to true to expose the link to the recommendation form. If false, the link will not display.
webui.suggest.loggedinusers.only	true	Enables only logged in users to suggest an item. The default value is false.

CONFIGURING CONTROLLED VOCABULARIES

DSpace now supports controlled vocabularies to confine the set of keywords that users can use while describing items.

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 organised in an 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>
```

Your 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 at:
http://www.dspace.org/images/onepointfivedocs/controlledvocabulary.zip.
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 java script 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

CONFIGURING MULTILINGUAL SUPPORT

SETTING THE DEFAULT LANGUAGE FOR THE APPLICATION

The default language for the application is set via the *[dspace]*/config/dspace.cfg parameter default.locale.
This is a locale according to i18n and might consist of country, country_language or country_language_variant,
e. g.: default.locale=en. If not default locale is specified the server locale will be used instead.

SUPPORTING MORE THAN ONE LANGUAGE

CHANGES IN DSPACE.CFG

With the *[dspace]*/config/dspace.cfg parameter webui.supported.locales you may provide a comma seperated list of supported (including the default locale) locales.
The locales might have the form country, country_language or country_language_variant, e. g.:
webui.supported.locales = en, de or webui.supported.locales = en, en_ca, de.
This will result in:

- a language switch in the default header
- the user will able to choose his preferred language, this will be part of his 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]*/modules/jspui/src/main/resources/Messages.properties
- *[dspace]*/modules/jspui/src/main/resources/Messages_en.properties
- *[dspace]*/modules/jspui/src/main/resources/Messages_de.properties
- *[dspace]*/modules/jspui/src/main/resources/Messages_fr.properties

Files to be localized:

- *[dspace]*/modules/jspui/src/main/resources/Messages_LOCALE.properties
- *[dspace]*/config/input-forms_LOCALE.xml
- *[dspace]*/config/default_LOCALE.license *should be pure ascii*
- *[dspace]*/config/news-top_LOCALE.html
- *[dspace]*/config/news-side_LOCALE.html
- *[dspace]*/config/emails/change_password_LOCALE
- *[dspace]*/config/emails/feedback_LOCALE
- *[dspace]*/config/emails/internal_error_LOCALE
- *[dspace]*/config/emails/register_LOCALE
- *[dspace]*/config/emails/submit_archive_LOCALE
- *[dspace]*/config/emails/submit_reject_LOCALE
- *[dspace]*/config/emails/submit_task_LOCALE
- *[dspace]*/config/emails/subscription_LOCALE
- *[dspace]*/config/emails/suggest_LOCALE
- *[dspace]*/jsp/help/collection-admin_LOCALE.html *in html keep the jump link as original*
- *[dspace]*/jsp/help/index_LOCALE.html
- *[dspace]*/jsp/help/site-admin_LOCALE.html

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 dictionary 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.

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 recommeded 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**
[jsp.dir]/community-list.jsp	*[dspace-source]*/dspace/modules/jspui/src/main/webapp/community-list.jsp
[jsp.dir]/mydspace/main.jsp	*[dspace-source]*/dspace/modules/jspui/src/main/webapp/mydspace/main.jsp

Heavy use is made of a style sheet, styles.css.jsp. If you make edits, copy the local version to *[dspace-source]*/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-*.jsp and /layout/footer-*.jsp. You can provide modified versions of these (in *[dspace-source]*/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.

After you've customized your JSPs, **you must rebuild the DSpace Web application**. If you haven't already built and installed it, follow the install directions. Otherwise, follow the steps below:

1. Rebuild the DSpace installation package by running the following command from your *[dspace-source]*/dspace/ directory:

   mvn package

2. Re-install the DSpace WAR(s) to *[dspace]*/webapps by running the following command from your *[dspace-source]*/dspace/target/dspace-[version].dir directory:

3.
4. ant -Dconfig=*[dspace]*/config/dspace.cfg update

5. Depending on your setup with Tomcat, you may also need to do the following:

   - Shut down Tomcat, and delete any existing *[tomcat]*/webapps/dspace directories.
   - Copy the new .war file(s) to the Tomcat webapps directory:
   - Restart Tomcat.

When you restart the web server you should see your customized JSPs.

ADVANCED DSPACE CUSTOMIZATIONS

Some customizations to the DSpace platform require advanced skills or knowledge to complete. The options list here will require either knowledge in system administration or may involve light programming.

CHECKSUM CHECKER

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

1. the execution mode
2. the logging output
3. the policy for removing old checksum results from the database

CHECKER EXECUTION MODE

Execution mode can be configured using command line options. Information on the options can be found at any time by running *[dspace]*/bin/checker --help. The different modes are described below; see the "Which to use" section that follows for details on the various pros and cons.

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.)

Limited Count Mode

To check a specific number of bitstreams, use the -c option followed by an integer number of bitstreams to check:

```
bin/checker -c 10
```

Limited count mode 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 -c 1 option had been given.

Limited Duration Mode

To run the Checker for a specific period of time, use the -d option with a time argument:

```
bin/checker -d 10m
bin/checker -d 2h
```

Valid options for specifying duration are s for seconds, m for minutes, h for hours, d for days, w for weeks, and y for years (OK, so we're optimists).

The checker will keep starting new bitstream checks for the specified duration, so actual execution duration will be slightly longer than the specified duration. Bear this in mind when scheduling checks.

Check Specific Bitstreams

To check one or more particular bitstreams by ID, use the -b option followed by one or more bitstream IDs:

```
bin/checker -b 1 2 3 4
```

This mode is useful when analyzing problems reported in the logs and when verifying that a resolution has been successful.

Check Specific Handles

Use the -a option followed by a handle:

```
bin/checker -a 123456/123
```

This will check all the bitstreams inside an item, collection or community.

Continuous Looping

There are two looping modes:

```
bin/checker -l     # Loops once through the repository
bin/checker -L              # Loops continuously through the repository
```

The -l option can be used if your repository is relatively small and your backup strategy requires it to be completely validated at a particular point. The -L option might be useful if you have a large repository, and you don't mind (or can avoid) the IO load caused by the checker.

Which to Use

The Checksum Checker was designed with the idea that most sys admins will run it from the cron. For small repositories we recommend using the -l option in the cron. For larger repositories that cannot be completely checked in a couple of hours, we recommend the -d option in the cron.

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:

```
bin/checker -l -v    #Loop through the repository once and
  report in detail about every bitstream checked.
```

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

CHECKER RESULTS PRUNING

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 command line option is used (unsuccessful ones will be retained indefinitely). 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 OR
2. passing in a properties file containing retention policies when using the -p option.

Pruning is controlled by a number of properties, each of which describes a checksum result code, and the length of time for which results with that code should be

retained. The format is checker.retention.[RESULT CODE]=[duration]. For example:
-

checker.retention.CHECKSUM_MATCH=8w

indicates that successful checksum matches will be retained for eight weeks. Supported units of time are

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

(Note that these units are also used for describing durations for the -d limited duration mode.)

There is a special property, checker.retention.default, that is used to assign a default retention period.

To execute the pruning you must use the -p command line option (with or without a properties file). Checksum Checker will prune the history table before beginning new checks. We recommend that you use this option regularly, as the checksum_history table can grow very large without it.

CUSTOM AUTHENTICATION

Since many institutions and organizations have exisiting 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.

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 regiser. 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.:*

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

6. 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

```
7.  authentication.x509.keystore.path = PATH TO JAVA KEYSTORE FILE
8.  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.
```

9. 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.IPAuthentication

You are than 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
```

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:

- **ldap.enable**
  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.

- **webui.ldap.autoregister**
  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.provider_url = ldap://ldap.myu.edu/o=myu.edu**
  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.

- **ldap.id_field = uid**
  This is the unique identifier field in the LDAP directory where the username is stored.

- **ldap.object_context = ou=people,o=myu.edu**
  This is the object context used when authenticating the user. It is appended to the ldap.id_field and username. For example uid=username,ou=people,o=myu.edu. You will need to modify this to match your ldap configuration.

- **ldap.search_context = ou=people**
  This is the search context used when looking up a user's ldap object to retrieve their data for autoregistering. With ldap.autoregister 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 uid=username,ou=people,o=byu.edu we now search in ou=people for filtering on [uid=username]. Often the ldap.search_context is the same as the ldap.object_context parameter. But again this depends on your ldap server configuration.

- **ldap.email_field = mail**
  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.

- **ldap.surname_field = sn**
  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.

- **ldap.givenname_field = givenName**
  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.

- **ldap.phone_field = telephoneNumber**
  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.

CONFIGURING SYSTEM STATISTICAL REPORTS

Currently the statistic's engine is only available for the JSP-based user interface

Statistics for the system can be made available at http://www.mydspaceinstance.edu/statistics. To use the system statistics you will have to initialise them as per the installation documentation, but before you do so you need to perform the customisations discussed here in order to ensure that the reports are generated correctly.

CONFIGURATION FILE

Configuration for the statistics system are in [dspace]/config/dstat.cfg and the file should guide you to correctly filling in the details required. For the most part you will not need to change this file.

CUSTOMISING SHELL SCRIPTS

To customise the supplied perl scripts to do monthly and general report generation it is necessary to modify the scripts themselves sightly. This is because these scripts were developed to speed up the process of using DStat at Edinburgh University Library and were not particularly intended for external use. They appear here for the convenience of others and in order to bridge the gap between the report generation and the inclusion of those reports into the DSpace UI, which is currently a clunky process.

In order to get these scripts to work for you, open each of the following in turn:

stat-general
stat-initial
stat-monthly
stat-report-general
stat-report-initial
stat-report-monthly

scripts eding with -general do the work for building reports spanning the entire history of the archive; scripts ending -initial are to initialise the reports by doing monthly reports from some start date up to the present; scripts ending -monthly generate a single monthly report FOR THE CURRENT MONTH. These scripts are just designed to make life easier, and are not particularly clever or elegant.

In each file you will find a section:

```
# Details used
##################################################

... some perl ...

##################################################
```

the perl between the lines of hashes defines the variables which will be used to do all of the processing in the report. The following explains what the variables mean and what they should be set to for each of the scripts

stat-initial:
$out_prefix: prefix to place in front of each output file.
$out_suffix: suffix for output file. A date will be inserted between the prefix and suffix
$start_year: year to start back-analysing monthly logs from
$start_month: month to start back-analysing monthly logs from
$dsrun: path to your dsrun script, usually [dspace]/bin/dsrun
$out_directory: directory into which to place analysis files, for example [dspace]/bin/log/

stat-monthly:
$out_prefix: prefix to place in front of each output file.

$out_suffix: suffix for output file. A date will be inserted between the prefix and suffix
$dsrun: path to your dsrun script, usually [dspace]/bin/dsrun
$out_directory: directory into which to place analysis files, for example [dspace]/bin/log/

stat-general:
$out_prefix: prefix to place in front of each output file.
$out_suffix: suffix for output file. Today's date will be inserted between the prefix and suffix
$dsrun: path to your dsrun script, usually [dspace]/bin/dsrun
$out_directory: directory into which to place analysis files, for example [dspace]/bin/log/

stat-report-initial:
$in_prefix: the prefix of the files generated by stat-initial
$in_suffix: the suffix of the files generated by stat-initial
$out_prefix: the report file prefix. Should be "report-" in order to work with DSpace UI
$out_suffix: the report file suffix. Should be ".html" in order to work with DSpace UI
$start_year: the start year used in stat-initial
$start_month: the start month used in stat-initial
$dsrun: path to your dsrun script, usually [dspace]/bin/dsrun
$in_directory: directory where analysis files were placed in stat-initial
$out_directory: the live reports directory: [dspace]/reports/

stat-report-monthly:
$in_prefix: the prefix of the files generated by stat-monthly
$in_suffix: the suffix of the files generated by stat-monthly
$out_prefix: the report file prefix. Should be "report-" in order to work with DSpace UI
$out_suffix: the report file suffix. Should be ".html" in order to work with DSpace UI
$dsrun: path to your dsrun script, usually [dspace]/bin/dsrun
$in_directory: directory where analysis files were placed in stat-monthly
$out_directory: the live reports directory: [dspace]/reports/

stat-report-general:
$in_prefix: the prefix of the files generated by stat-general
$in_suffix: the suffix of the files generated by stat-general
$out_prefix: the report file prefix. Should be "report-general-" in order to work with DSpace UI
$out_suffix: the report file suffix. Should be ".html" in order to work with DSpace UI
$dsrun: path to your dsrun script, usually [dspace]/bin/dsrun
$in_directory: directory where analysis files were placed in stat-general
$out_directory: the live reports directory: [dspace]/reports/

If you want additional customisations, you will need to modify the lines which build the command to be executed and change the parameters passed to the java processes which actually carry out the analysis. For more information on these processes either build the javadocs or run:

```
[dspace]/bin/dsrun org.dspace.app.statistics.LogAnalyser -help
[dspace]/bin/dsrun org.dspace.app.statistics.ReportGenerator -help
```

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** - Qualfied 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:

- Uncomment the oai.didl.maxresponse item in dspace.cfg
- Uncomment the DIDL Crosswalk entry from the config/templates/oaicat.properties file
- Run the bin/install-configs script
- Restart Tomcat
- Verify the Crosswalk is activated by accessing a URL such as http://mydspace/oai/request?verb=ListRecords&metadataPrefix=didl

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.

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.

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. So, a line like:

```
 crosswalk.mods.properties.MODS = crosswalks/mods.properties
```
defines a crosswalk named MODS whose configuration comes from the file [dspace]/config/crosswalks/mods.properties.

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:
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.

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 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. So, a line like:

```
  crosswalk.qdc.properties.QDC = crosswalks/qdc.properties
```

defines a crosswalk named QDC whose configuration comes from the file [dspace]/config/crosswalks/qdc.properties.

You'll also need to configure the namespaces and schema location strings for the XML output generated by this crosswalk. The namespaces property names are of the format:

```
crosswalk.qdc.namespace.PREFIX = URI
```

where PREFIX is the namespace prefix and URI is the namespace URI.

For example, this shows how a crosswalk named "QDC" would be configured:

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

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>
```

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.

A submission crosswalk is described by a configuration key starting with 'crosswalk.submission.", like

```
  crosswalk.submission.PluginName.stylesheet = path
```

The PLUGINNAME is, of course, the plugin's name. The PATH value is the path to the file containing the crosswalk stylesheet (relative to DSPACE.DIR/config).

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

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

A dissemination crosswalk is described by a configuration key starting with 'crosswalk.dissemination.", like

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

The PLUGINNAME is, of course, the plugin's name. The PATH value is the path to the file containing the crosswalk stylesheet (relative to DSPACE.DIR/config).

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

```
    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, i.e.:

```
   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
```

DSPACE INTERMEDIATE METADATA (DIM) FORMAT

XSLT crosswalk plugins translate between the external metadata format and an XML format called DSPACE INTERMEDIATE METADATA, which exists ONLY for the purpose of XSLT crosswalks. It is NEVER to be exported from DSpace, since it is not an acknowledged metadata format, it is simply an expression of the way DSpace stores its metadata fields internally.

All the elements in a DIM document are in the namespace http://www.dspace.org/xmlns/dspace/dim.

The root element is named dim. It has zero or more children, all field elements. It may have an attribute dspaceType, which identifies the type of object ("ITEM", "COLLECTION", or "COMMUNITY") this metadata describes. This attribute is only guaranteed to be set for dissemination crosswalks.

Each field element may have the following attributes:

- mdschema (Required) The metadata schema, e.g. "dc".
- element (Required) Element name, such as "contributor".
- qualifier Qualifier name, such as "author".
- lang Language code describing language of this entry.

The value of field is the value of that metadata field. Fields with the same qualifiers may be repeated.

Here is an example of the DIM format:

```
  <dim:dim xmlns:dim="http://www.dspace.org/xmlns/dspace/dim"
dspaceType="ITEM">
    <dim:field mdschema="dc" element="title" lang="en_US">
      The Endochronic Properties of Resublimated Thiotimonline
    </dim:field>
    <dim:field mdschema="dc" element="contributor" qualifier="author">
      Isaac Asimov
    </dim:field>
    <dim:field mdschema="dc" element="language" qualifier="iso">
      eng
    </dim:field>
    <dim:field mdschema="dc" element="subject" qualifier="other" lang="en_US">
      time-travel scifi hoax
    </dim:field>
    <dim:field element="publisher">
      Boston University Department of Biochemistry
    </dim:field>
  </dim:dim>
```

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 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.

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. bitstreamformatregistry 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 Dynamic / Self-Named Format FilterXXX.

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 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.inputFormats =
Microsoft Word

filter.org.dspace.app.mediafilter.MyComplexMediaFilter.Excel2CSV.inputFormats =
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.outputFormat = 
Adobe PDF

filter.org.dspace.app.mediafilter.MyComplexMediaFilter.Excel2CSV.outputFormat = 
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");
```

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.

BROWSE INDEX CREATION

To create all the various browse indices that you define in the configuration as described in the section Browse Configuration there are a variety of options available to you. You can see these options at any time by running the indexer without any arguments, thus:

```
[dspace]/bin/dsrun org.dspace.browse.IndexBrowse
```

This will show you the following options are available to you:

-r,--rebuild
: should we rebuild all the indices, which removes old index tables and creates new ones. For use with -f. Mutually exclusive with -d

-s,--start
: [-s <int>] start from this index number and work upward (mostly only useful for debugging). For use with -t and -f

-x,--execute
: execute all the remove and create SQL against the database. For use with -t and -f

-i,--index
: actually do the indexing. Mutually exclusive with -t and -f

-o,--out
: [-o <filename>] write the remove and create SQL to the given file. For use with -t and -f

-p,--print
: write the remove and create SQL to the stdout. For use with -t and -f

-t,--tables
: create the tables only, do not attempt to index. Mutually exclusive with -f and -i

-f,--full
: make the tables, and do the indexing. This forces -x. Mutually exclusive with -t and -i

-v,--verbose
: print extra information to the stdout. If used in conjunction with -p, you cannot use the stdout to generate your database structure

-d,--delete
: delete all the indices, but don't create new ones. For use with -f. This is mutually exclusive with -r

-h,--help
: show this help documentation. Overrides all other arguments

The following, then, are examples of what you want to achieve and how this is done with the command line options

DO A FULL BROWSE RE-INDEX, TEARING DOWN ALL OLD TABLES AND RECONSTRUCTING WITH THE NEW CONFIGURATION

```
[dspace]/bin/dsrun org.dspace.browse.IndexBrowse -f -r
```

DO A FULL BROWSE RE-INDEX WITHOUT MODIFYING THE TABLE STRUCTURE (This should be your default approach if indexing, for example, via a cron job periodically)

```
[dspace]/bin/dsrun org.dspace.browse.IndexBrowse -i
```

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

```
[dspace]/bin/dsrun org.dspace.browse.IndexBrowse -r -t -p -v -x -o myfile.sql
```

During installation you will have run the ant target:

```
ant index
```

This creates the index tables as per the configuration, and will produce your initial indexed state. From this point on, you should not use ant to generate your indices, as it is not a very good execution environment. Instead, if you feel the need, or your local customisations demand regular full indexing you should set up a regular script to execute:

```
[dspace]/bin/dsrun org.dspace.browse.IndexBrowse -i
```

DIRECTORIES AND FILES

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.

SOURCE DIRECTORY LAYOUT

- *[dspace-source]*
    - dspace/ - Directory which contains all build and configuration information for DSpace
        - build.xml - The Build file for Ant -- used to preform a fresh_install, upgrade, or deploy new changes.
        - 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 MODS(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.
    - language-packs/ - Contains "dictionary files" -- Java properties files that contain user interface text in different languages
    - 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.

    - templates/ - Configuration files for libraries and external applications (e.g. Apache, Tomcat) are kept and edited here. They can refer to properties in the main DSpace configuration - have a look at a couple. When they're updated, a command line tool fills out these files with appropriate values from dspace.cfg, and copies them to their appropriate location (hence "templates".)

- docs/ - DSpace system documentation. The technical documentation for functionality, installation, configuration, etc.

- etc/ - Miscellaneous configuration need to install DSpace that isn't really to do with system configuration - e.g. the PostgreSQL database schema, and a couple of configuration files that are used during the build process but not by the live system. Also contains the deployment descriptors (web.xml files) for the Web UI and OAI-PMH support .war files.

    - 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.

        - src/main/webapp/ - The overlay for JSPUI Web Application. This is the location to place any custom JSPs to be used by DSpace.

    - lni - 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
            - themes/ - The location to place custom Themes for the XMLUI
- 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)

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

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
        - fmt.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

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

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.

DSpace Log File Locations

Log File	**What's In It**
[dspace]/log/dspace.log	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 *[dspace]*/config/templates/log4j.properties file and then running *[dspace]*/bin/install-configs.
[tomcat]/logs/catalina.out	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 (dspace.jar), it would be logged in catalina.out.
[tomcat]/logs/hostname_log.yyyy-mm-dd.txt	If you're running Tomcat stand-alone (without Apache), it logs some information and errors for

	specific Web applications to this log file. hostname will be your host name (e.g. dspace.myu.edu) and yyyy-mm-dd will be the date.
[tomcat]/logs/apache_log.yyyy-mm-dd.txt	If you're using Apache, Tomcat logs information about Web applications running through Apache (mod_webapp) in this log file (yyyy-mm-dd being the date.)
[apache]/error_log	Apache logs to this file. If there is a problem with getting mod_webapp working, this is a good place to look for clues. Apache also writes to several other log files, though error_log tends to contain the most useful information for tracking down problems.
[dspace]/log/handle-plug.log	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 *[dspace]*/config/templates/log4j-handle-plugin.properties.
[dspace]/log/handle-server.log	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.
[dspace]/handle-server/error.log	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.

ARCHITECTURE

OVERVIEW

The DSpace system is organized into three layers, each of which consists of a number of components.

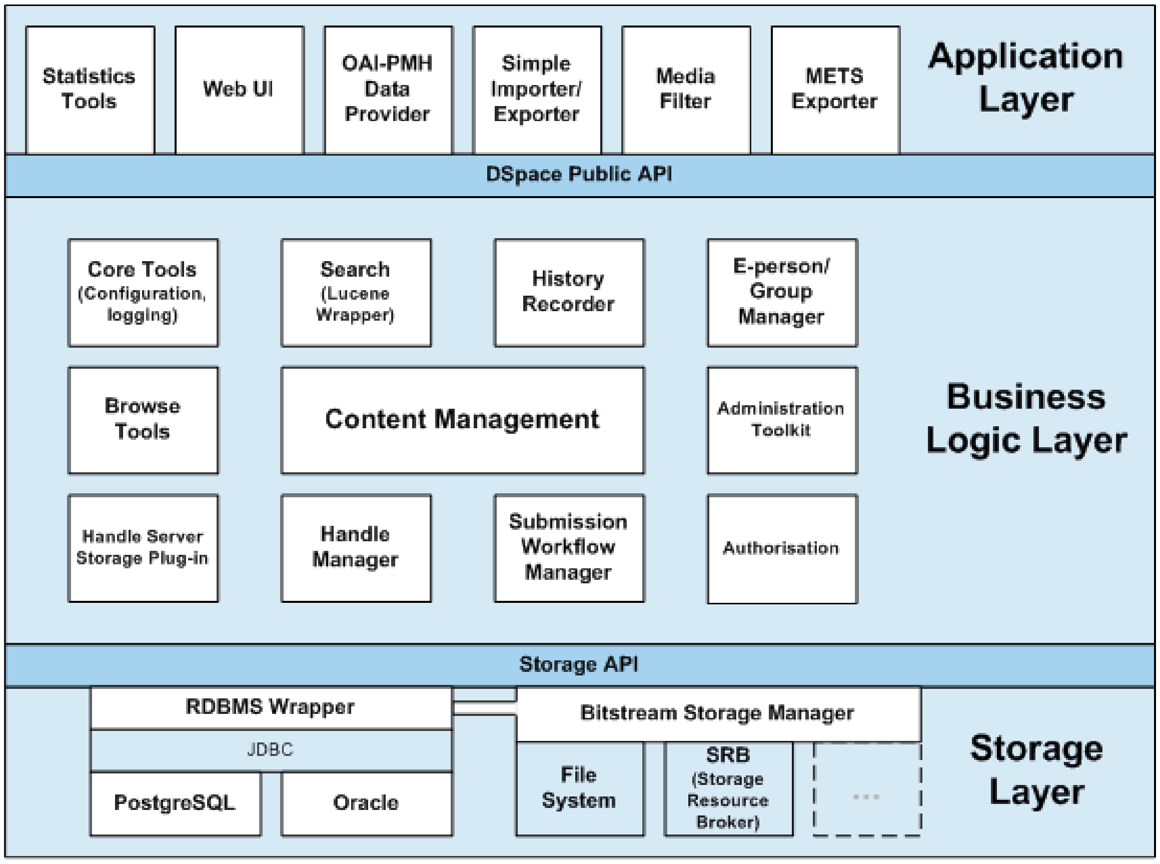

Figure 4 – DSpace System Architecture

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 Open Archives Initiative (http://www.openarchives.org/) protocol for metadata harvesting service.

Each layer only invokes the layer below it; the application layer may not used 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 is 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.

Source Code Packages

Packages within	**Correspond to components in**
org.dspace.app	Application layer
org.dspace	Business logic layer (except storage and app)
org.dspace.storage	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 source directory and running:

```
ant public_api
```

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, Business Logic Layer, and Application Layer.

STORAGE LAYER

RDBMS

DSpace uses a relational database to store all information about the organization of content, metadata about the content, information about e-people and authorization, and the state of currently-running workflows. The DSpace system also uses the relational database in order to maintain indices that users can browse.

Graphical visualization of the relational database

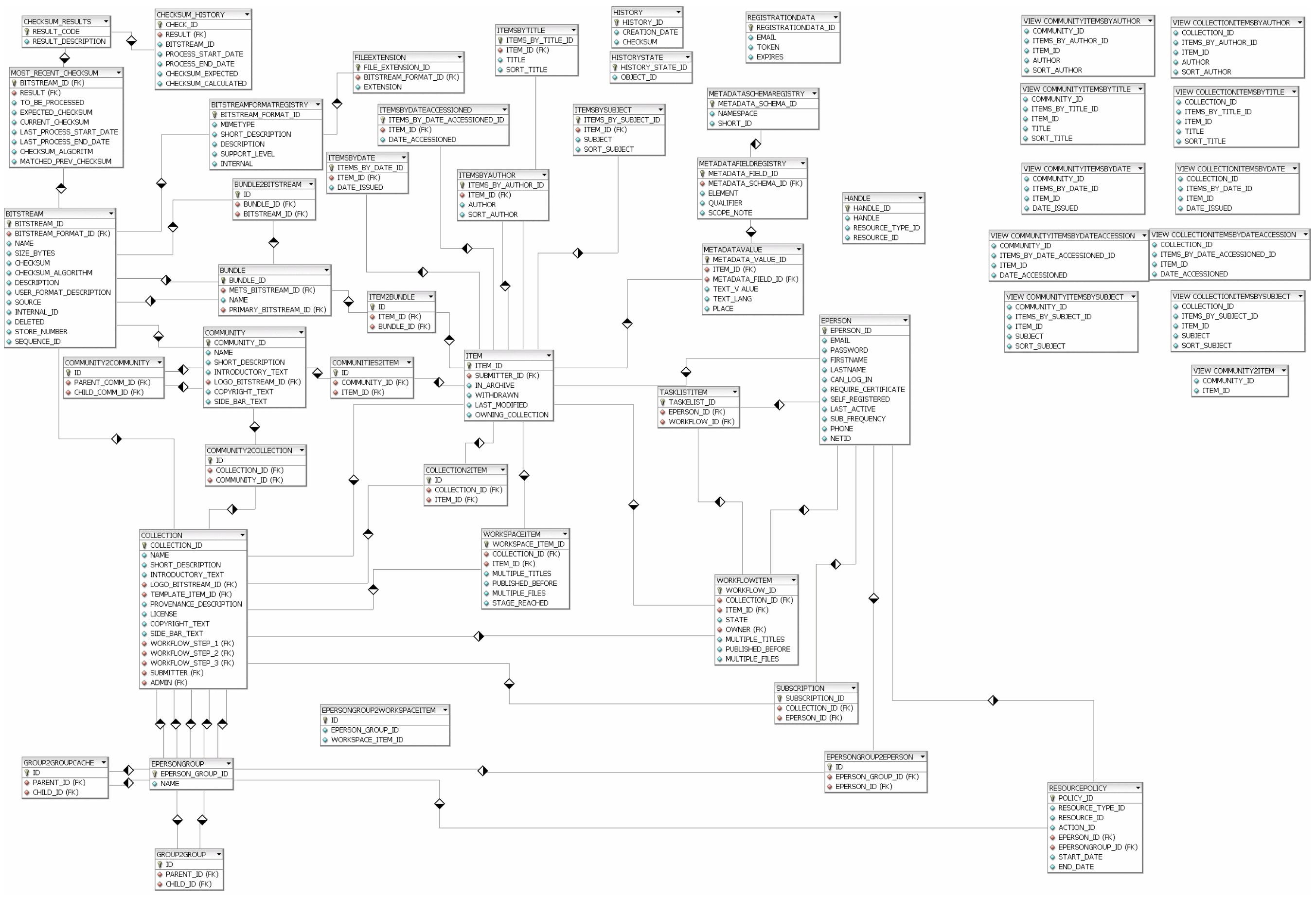

Figure 5 – Graphical Visualization of the Relational Database.

Most of the functionality that DSpace uses can be offered by any standard SQL database that supports transactions. Presently, the browse indices use some features specific to PostgreSQL (http://www.postgresql.org/) and Oracle, (http://www.oracle.com/database/) so some modification to the code would be needed before DSpace would function fully with an alternative database back-end.

The org.dspace.storage.rdbms package provides access to an SQL database in a somewhat simpler form than using JDBC directly. The main class is DatabaseManager, which executes SQL queries and returns TableRow or TableRowIterator objects. The InitializeDatabase class is used to load SQL into the database via JDBC, for example to set up the schema.

All calls to the Database Manager require a DSpace Context object. Example use of the database manager API is given in the org.dspace.storage.rdbms package Javadoc.

The database schema used by DSpace (for PostgreSQL) is stored in *[dspace-source]*/dspace/etc/database_schema.sql in the source distribution. It is stored in the form of SQL that can be fed straight into the DBMS to construct the database. The schema SQL file also directly creates two e-person groups in the database that are required for the system to function properly.

Also in *[dspace-source]*/dspace/etc 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.

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.

CONFIGURING THE RDBMS COMPONENT

The database manager is configured with the following properties in dspace.cfg:

db.url	The JDBC URL to use for accessing the database. This should not point to a connection pool, since DSpace already implements a connection pool.
db.driver	JDBC driver class name. Since presently, DSpace uses PostgreSQL-specific features, this should be org.postgresql.Driver.
db.username	Username to use when accessing the database.
db.password	Corresponding password ot use when accessing the database.

BITSTREAM STORE

DSpace offers two means for storing content. The first is in the file system on the server. The second is using SRB (Storage Resource Broker) (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 committed, 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 stored

None 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.

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.

CONFIGURING THE BITSTREAM STORE

Both traditional and SRB bitstream stores are configured in dspace.cfg.

CONFIGURING TRADITONAL 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. The there will be a set of SRB account parameters like this:

srb.host.1 = mysrbmcathost.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 to 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.

BUSINESS LOGIC LAYER

CORE CLASSES

The org.dspace.core package provides some basic classes that are used throughout the DSpace code.

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.

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.

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 responsiblity 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.

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.

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 (FATAL, WARN, INFO or DEBUG)	INFO
Java class	org.dspace.app.webui.servlet.DSpaceServlet
	@
User email or anonymous	anonymous
	:
Extra log info from context	session_id=BD84E7C194C2CF4BD0EC3A6CAD0142BB

	:
Action	view_item
	:
Extra info	handle=1721.1/1686

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.

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.

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.

Items 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.

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.

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 take place 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:

Context context = new Context();
Bitstream b =
Bitstream.find(context, 1234);
b.setName("newfile.txt");
b.update();
context.complete();
	Will change storage

Context context = new Context();
Bitstream b =
Bitstream.find(context, 1234);
b.setName("newfile.txt");
b.update();
context.abort();
	Will not change storage (context aborted)

Context context = new Context();
Bitstream b =
Bitstream.find(context, 1234);
b.setName("newfile.txt");
context.complete();
	The new name **will not** be stored since update was not invoked

Context context = new Context();
Bitstream bs =
Bitstream.find(context, 1234);
	The bitstream **will** be included in the bundle, since update doesn't need to be called

```
Bundle bnd = Bundle.find(context,
5678);
bnd.add(bs);
context.complete();
```

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 BitstreamFormats) to be instantiated.

Instantiating an Item object causes the appropriate Bundle objects (etc.) and hence BitstreamFormats 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 inconsistenties 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).

DUBLIN CORE METADATA

The DCValue class is a simple container that represents a single Dublin Core element, optional qualifier, value and language. Note that as of 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 Library Application Profile (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
date	Any or unqualified	ISO 8601 in the UTC time zone, with either year, month, day, or second precision. Examples: 2000 2002-10 2002-08-14 1999-01-01T14:35:23Z	DCDate
contributor	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: Doe, John Smith, John Jr. van Dyke, Dick Massachusetts Institute of Technology	DCPersonName
language	iso	A two letter code taken ISO 639, followed optionally by a two letter country code taken from ISO 3166. Examples: en fr en_US	DCLanguage

relation	ispartofseries	The series name, following by a semicolon followed by the number in that series. Alternatively, just free text. MIT-TR; 1234 My Report Series; ABC-1234 NS1234	DCSeriesNumber

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 where ever 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.

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);
```

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.

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.

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 implementations 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 configuration 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 it 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 Objects 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.

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 PluginConfigurationError;

  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 PluginConfigurationError 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 no 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.

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.SimpleDis
patcher

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 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.xsl
   crosswalk.dissemination.stylesheet.MODS = xwalk/mods.xsl

   plugin.selfnamed.crosswalk.org.dspace.content.metadata.DisseminationCross
   walk = \
         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
```

VALIDATING THE CONFIGURATION

The Plugin Manager is very sensitive to mistakes in the DSpace configuration. Subtle errors can have unexpected consequnces 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().

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.SimpleDis
patcher
```

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.xsl
crosswalk.dissemination.stylesheet.MODS = xwalk/mods.xsl

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 guide for further details.

WORKFLOW SYSTEM

The primary classes are:

org.dspace.content.WorkspaceItem	contains an Item before it enters a workflow
org.dspace.workflow.WorkflowItem	contains an Item while in a workflow
org.dspace.workflow.WorkflowManager	responds to events, manages the WorkflowItem states
org.dspace.content.Collection	contains List of defined workflow steps
org.dspace.eperson.Group	people who can perform workflow tasks are defined in EPerson Groups

org.dspace.core.Email	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.

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.

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.

AUTHORIZATION

The primary classes are:

org.dspace.authorize.AuthorizeManager	does all authorization, checking policies against Groups
org.dspace.authorize.ResourcePolicy	defines all allowable actions for an object
org.dspace.eperson.Group	all policies are defined in terms of EPerson 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.)

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.

MISCELLANEOUS AUTHORIZATION NOTES

Where do items get their read policies? From the 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.

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.

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 is invoked by the dspace/bin/index-all 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.

OUR 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.

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	contributor.* creator.* description.statementofresponsibility
Titles	title.*
Keywords	subject.*
Abstracts	description.abstract description.tableofcontents
Series	relation.ispartofseries
MIME types	format.mimetype
Sponsors	description.sponsorship

Identifiers	identifier.*

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.

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 flavor 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 save 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.

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/index-all shell script) that causes the indices to be regenerated from scratch.

CAVEATS

Presently, the browse API is not tremendously efficient. 'Indexing' takes the form of simply extracting the relevant Dublin Core value, normalizing 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.

HISTORY RECORDER

The purpose of the history subsystem is to capture a time-based record of significant changes in DSpace, in a manner suitable for later refactoring or repurposing. Note that the history data is not expected to provide current information about the archive; it simply records what has happened in the past.

The Harmony project (http://www.metadata.net/harmony/) describes a simple and powerful approach for modeling temporal data. The DSpace history framework adopts this model. The Harmony model is used by the serialization mechanism (and ultimately by agents who interpret the serializations); users of the History API need not be aware of it. The content management API handles invocations of the history system. Users of the DSpace public API do not generally need to use the history API.

When anything of archival interest occurs in DSpace, the saveHistory method of the HistoryManager is invoked. The parameters contain a reference to anything of archival interest. Upon reception of the object, it serializes the state of all archive objects referred to by it, and creates Harmony-style objects and associations to describe the relationships between the objects. (A simple example is given below). Note that each archive object must have a unique identifier to allow linkage between discrete events; this is discussed under "Unique IDs" below.

The serializations (including the Harmony objects and associations) are persisted as files in the /dspace/history (or other configured) directory. The history and historystate tables contain simple indicies into the serializations in the file system.

ARCHIVAL EVENTS

The following events are significant enough to warrant history records:

Communities
- create/modify/delete
- add/remove Collection to/from Community

Collections
- create/modify/delete
- add/remove Item to/from Collection

Items
- create/modify/delete
- assign Handle to Item
- modify Item contents (Bundles, Bitstreams, metadata fields, etc)

EPerson
- create/modify/delete

Workflow
- Workflow completed

SERIALIZATIONS

The serialization of an archival object consists of:

- Its instance fields (i.e., non-static, non-transient fields)
- The serializations of associated objects (or references to these serializations).

UNIQUE IDS

To be able to trace the history of an object, it is essential that the object have a unique identifier. Since not all objects in the system have Handles, the unique identifiers are only weakly tied to the Handle system. Instead, the identifier consists of:

- an identifier for the project

- a site id (using the handle prefix)
- an RDBMS-based id for objects

STORAGE

When an archive object is serialized, an object ID and MD5 checksum are recorded. When another object is serialized, the checksum for the serialization is matched against existing checksums for that object. If the checksum already exists, the object is not stored; a reference to the object is used instead. Note that since none of the serializations are deleted, reference counting is unnecessary.

The history data is not initially stored in a query-able form. Two simple RDBMS tables give basic indications of what is stored, and where. The history table is an index of serializations with checksums and dates. The history_id column corresponds to the file in which a serialization is stored. For example, if the history ID is 123456, it will be stored in the file:

/dspace/history/00/12/34/123456

The table also contains the date the serialization was written and the MD5 checksum of the serialization.

The historystate table is supposed to indicate the most recent serialization of any given object.

EXAMPLE

An item is submitted to a collection via bulk upload. When (and if) the item is eventually added to the collection, the history method is called, with references to the item, its collection, the e-person who performed the bulk upload, and some indication of the fact that it was submitted via a bulk upload.

When called, the HistoryManager does the following: It creates the following new resources (all with unique ids):
- An event
- A state
- An action

It also generates the following relationships:

```
event  --atTime-->     time
event  --hasOutput-->  state
Item   --inState-->    state
state  --contains-->   Item
action --creates-->    Item
event  --hasAction-->  action
action --usesTool-->   DSpace Upload
action --hasAgent-->   User
```

The history component serializes the state of all archival objects involved (in this case, the item, the e-person, and the collection). It creates entries in the history database tables which associate the archival objects with the generated serializations.

CAVEATS

This history system is a largely untested experiment. It also needs further documentation. There have been no serious efforts to determine whether the information written by the history system, either to files or the database tables, is accurate. In particular, the historystate table does not seem to be correctly written.

CHECKSUM CHECKER

The architecture of the checker is documented in the package javadocs, run ant public_api, and look in build/public_api/index.html.

APPLICATION LAYER

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.

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 1.2, the deployment mechanism has changed; the build process creates easy-to-deploy Web application archives (.war files).

Locations of Web UI Source Files

Location	**Description**
org.dspace.app.webui	Web UI source files
org.dspace.app.webui.filter	Servlet Filters (Servlet 2.3 spec)
org.dspace.app.webui.jsptag	Custom JSP tag class files
org.dspace.app.webui.servlet	Servlets for main Web UI (controllers)
org.dspace.app.webui.servlet.admin	Servlets that comprise the administration part of the Web UI
org.dspace.app.webui.util	Miscellaneous classes used by the servlets and filters
[DSPACE-SOURCE]/jsp	The JSP files
[DSPACE-SOURCE]/jsp/local	This is where you can place customized versions of JSPs -- see the configuration section XXX
[DSPACE-SOURCE]/jsp/WEB-INF/dspace-tags.tld	Custom DSpace JSP tag descriptor
[DSPACE-SOURCE]/etc/dspace-web.xml	The Web application deployment descriptor. Before including in the .war file, the text @@dspace.dir@@ will be replaced with the DSpace installation directory (referred to as [DSPACE] elsewhere in this system documentation). This allows the Web application to pick up the DSpace configuration and environment.

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.

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.

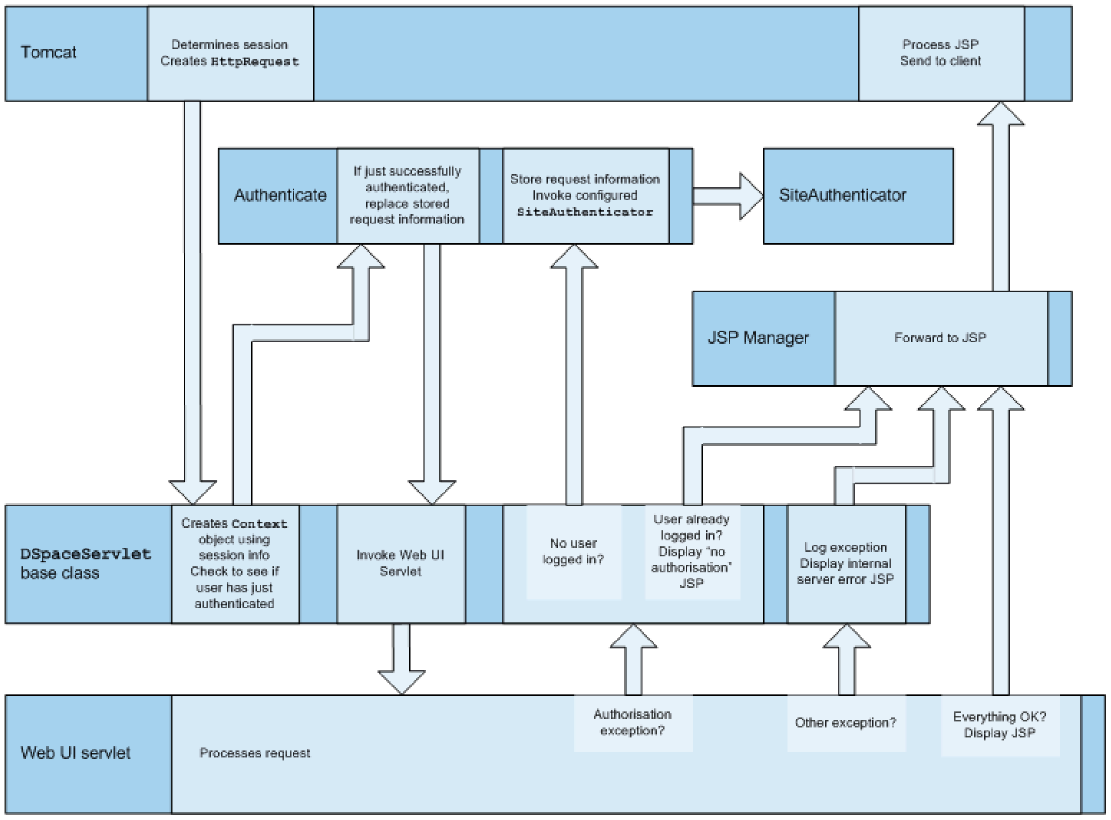
Tomcat
Determines session
Creates HttpRequest
Process JSP
Send to client
Authenticate
If just successfully authenticated, replace stored request information
Store request information
Invoke configured SiteAuthenticator
SiteAuthenticator
JSP Manager
Forward to JSP
DSpaceServlet base class
Creates Context object using session info
Check to see if user has just authenticated
Invoke Web UI Servlet
No user logged in?
User already logged in? Display "no authorisation" JSP
Log exception
Display internal server error JSP
Web UI servlet
Processes request
Authorisation exception?
Other exception?
Everything OK? Display JSP

Figure 6 – Flow of Control During HTTP Request Processing

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 anyway. 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.

selecteperson

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.

INTERNATIONALIZATION

The Java Standard Tag Library v1.0 (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 atring parameters:

```
jsp.submit.show-uploaded-file.size-in-bytes = {0} bytes
```

In the JSP using this key can be used in the way belov:

```
<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 = {0,number} bytes. Setting the parameter as <fmt:param value="${variable}" /> workes 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 ResourceBundle.getBundle (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="/mydspace"
            parenttitle="My DSpace">
```

You now do:

```
<dspace:layout titlekey="jsp.page.title"
            parentlink="/mydspace"
            parenttitlekey="jsp.mydspace">
```

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/mydspace/main.jsp use:

jsp.mydspace.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 i18n page (http://wiki.dspace.org/I18nSupport) of the DSpace Wiki.

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 contraints. 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.

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.

OAI-PMH DATA PROVIDER

The DSpace platform supports the Open Archives Initiative Protocol for Metadata Harvesting (http://www.openarchives.org/) (OAI-PMH) version 2.0 as a data provider. This is accomplished using the OAICat framework from OCLC (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 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 oaicat.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 &gt;)

In addition to the implementations of the OAICat interfaces, there are two configuration files relevant to OAI support:

oaicat.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.

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.DSpaceOAICatalog 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.)

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.

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.

'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.

DELETIONS

DSpace keeps track of deletions (withdrawals). These are exposed via OAI, which has a specific mechansim 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.

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.)

COMMUNITY AND COLLECTION STRUCTURE IMPORTER

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/structure-builder -f [source xml] -o [output xml file] -e [administrator email]
```

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.

The source xml document needs to be 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>
```

LIMITATION

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

PACKAGE IMPORTER AND EXPORTER

This command-line tool gives you access to the Packager plugins. It can INGEST a package to create a new DSpace 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.

INGESTING

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

```
[DSPACE]/bin/packager -e USER -c 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 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 -
```

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
```

METS PACKAGES

DSpace 1.4 includes a package disseminator and matching ingester for the DSpace METS SIP (Submission Information Package) format. They were created to help end users prepare sets of digital resources and metadata for submission to the archive using well-defined standards such as METS (http://www.loc.gov/standards/mets/), MODS (http://www.loc.gov/standards/mods/), and PREMIS (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.

ITEM IMPORTER AND EXPORTER

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

DSPACE SIMPLE ARCHIVE FORMAT

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

```
archive_directory/
    item_000/
        dublin_core.xml -- qualified Dublin Core metadata
        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 file has the following format, where each Dublin Core element has its own entry within a <dcvalue> tagset. There are currently three tag elements 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></dublin_core>
    <dcvalue element="title" qualifier="alternate" language="fr" ">J'aime les
Printemps</dcvalue>
</dublin_core>
```

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

The contents file simply enumerates, one file per line, the bitstream file names. 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.

IMPORTING ITEMS

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

The item importer is in org.dspace.app.itemimport.ItemImport, and is run with the import utility in the dspace/bin directory. Running it with -h gets the current command-line arguments. Another very important flag is the --test flag, which you can use with any command to simulate all of the actions it will perform without actually making any changes to your DSpace instance - very useful for validating your item directories before doing an import. In the importer's arguments you can use either the user's database ID or email address and the eperson ID, and the collection's database ID or handle as arguments. Currently with the importer you can add, remove, and replace items in a collection. If you specify more than one collection argument then the items will be imported to multiple collections, and the first collection specified becomes the "owning" collection. If there is an error and the import is aborted, there is a --resume flag that you can try to resume the import where you left off after you fix the error.

To add items to a collection with an EPerson as the submitter, type:

```
[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
```

which would then 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 then 'unimport' with the command:

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

The imported items listed in the map file would then be deleted. If you wish to replace previously imported items, you can give the command:

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

Replacing items uses the map file to replace the old items and still retain their handles.

The importer usually bypasses any workflow assigned to a collection, but adding the --workflow option will route the imported items through the workflow system.

The importer also has a --test flag that will simulate the entire import process without actually doing the import. This is extremely useful for verifying your import files before doing the import step.

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. To export a collection's items you type:

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

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/export --type=ITEM --id=itemID --dest=dest_dir --number=seq_num
```

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.

TRANSFERRING ITEMS BETWEEN DSPACE INSTANCES

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 <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. Use

dspace_migrate --help

for instructions on use of the script.

REGISTERING (NOT IMPORTING) BITSTREAMS

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

ACCESSIBLE STORAGE

To register an item its bitstreams must reside on storage accessible to DSpace and therefore referenced by an *asset store number* in dspace.cfg. The configuration file dspace.cfg establishes one or more asset stores through the use of an integer asset store number. This number relates to a directory in the DSpace host's file system or a set of SRB account parameters. This asset store number is described in The dspace.cfg Configuration Properties FileXXX 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 that you will want to mix the bitstreams of normally ingested and imported items and registered items.

REGISTERING ITEMS USING THE ITEM IMPORTER

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

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

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

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

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

where

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

The bundle, that is everything after the filepath, is optional and is normally not used.

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

```
dsrun org.dspace.app.itemimport.ItemImport --add --eperson=joe@user.com  --
collection=collectionID --source=items_dir --mapfile=mapfile
```

(or by using the short form)

```
dsrun org.dspace.app.itemimport.ItemImport -a -e joe@user.com  -c collectionID -s
items_dir -m mapfile
```

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

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

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

INTERNAL IDENTIFICATION AND RETRIEVAL OF REGISTERED ITEMS

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

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

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

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

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

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

EXPORTING REGISTERED ITEMS

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

METS EXPORT OF REGISTERED ITEMS

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

DELETING REGISTERED ITEMS

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

METS TOOLS

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

THE EXPORT TOOL

The METS export tool is invoked via the command line like this:

```
[DSPACE]/bin/dsrun org.dspace.app.mets.METSExport --help
```

The tool can export an individual item, the items within a given collection, or everything in the DSpace instance. To export an individual item, use:

```
[DSPACE]/bin/dsrun org.dspace.app.mets.METSExport --item [HANDLE]
```

To export the items in collection hdl:123.456/789, use:

```
[DSPACE]/bin/dsrun org.dspace.app.mets.METSExport --collection hdl:123.456/789
```

To export all the items DSpace, use:

```
[DSPACE]/bin/dsrun org.dspace.app.mets.METSExport --all
```

With any of the above forms, you can specify the base directory into which the items will be exported, using --destination [DIRECTORY]. If this parameter is omitted, the current directory is used.

THE AIP FORMAT

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 Metadata Object Description Schema (MODS) (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 DCMI Metadata Terms (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 ID 123_456_789_24. This is because ID and GROUPID attributes must be of type xsd:id.

LIMITATIONS

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

MEDIAFILTERS: TRANSFORMING DSPACE CONTENT

DSpace can apply filters to content/bitstreams, creating new content. Filters are included that extract text for **full-text searching**, and create **thumbnails** for items that contain images. The media filters are controlled by the MediaFilterManager which traverses the asset store, invoking the MediaFilter or 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/filter-media -h
  - Display help message describing all command-line options.
- **Force mode**: [dspace]/bin/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/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/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/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-all elsewhere.
- **Plugin mode**: [dspace]/bin/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/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), and call it similar to the following:
        - [dspace]/bin/filter-media -s `less filter-skiplist.txt`
- **Verbose mode**: [dspace]/bin/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 FilterXXX topic and comments in the source file FormatFilter.java for more information. In theory filters could be implemented in any programming language (C, Perl, etc.) However, they need to be invoked by the Java code in the Media Filter class that you create.

SUB-COMMUNITY MANAGEMENT

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

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

Using the dsrun utility in the dspace/bin directory, the establish operation looks like this:

```
dsrun org.dspace.administer.CommunityFiliator --set --parent=parentID --
child=childID
```

(or using the short form)

```
dsrun org.dspace.administer.CommunityFiliator -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:

```
dsrun org.dspace.administer.CommunityFiliator --remove --parent=parentID --
child=childID
```

(or using the short form)

```
dsrun org.dspace.administer.CommunityFiliator -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.

DRI SCHEMA REFERENCE

Digital **R**epository **I**nterface (**DRI**) is a schema that governs the structure of a Manakin DSpace page when encoded as an XML Document. It determines what elements can be present in the Document and the relationship of those elements to each other. This section explains the purpose of DRI, provides a broad architectural overview, and explains common design patterns. Also included is 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.

INTRODUCTION

This manual describes the Digital Repository Interface (DRI) as it applies to the DSpace digital repository and Manakin web-based user 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, consists of several components, written using Java, XML, and XSL, and is implemented in Cocoon (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 section, the terms XML Document, DRI Document, and Document will be used interchangeably.

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 section.

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 Cocoon's i18n schema (http://cocoon.apache.org/2.1/userdocs/i18nTransformer.html) for internationalization, DCMI's Dublin Core (http://dublincore.org/), and the Library of Congress's METS schema (http://www.loc.gov/standards/mets/). The design of structural elements was derived primarily from TEI (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.

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 singe 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.

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 elementÃs 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 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 following 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.

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.

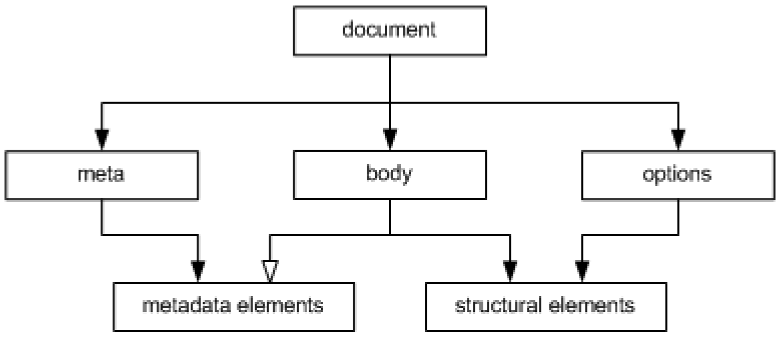

Figure 7 – The two content types across three major divisions of a DRI page.

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 repositoryMeta, 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 Manakin to apply uniform templates and styling rules to all DSpace pages and is the fundamental difference from the JSP approach used exclusively in previous versions of 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.

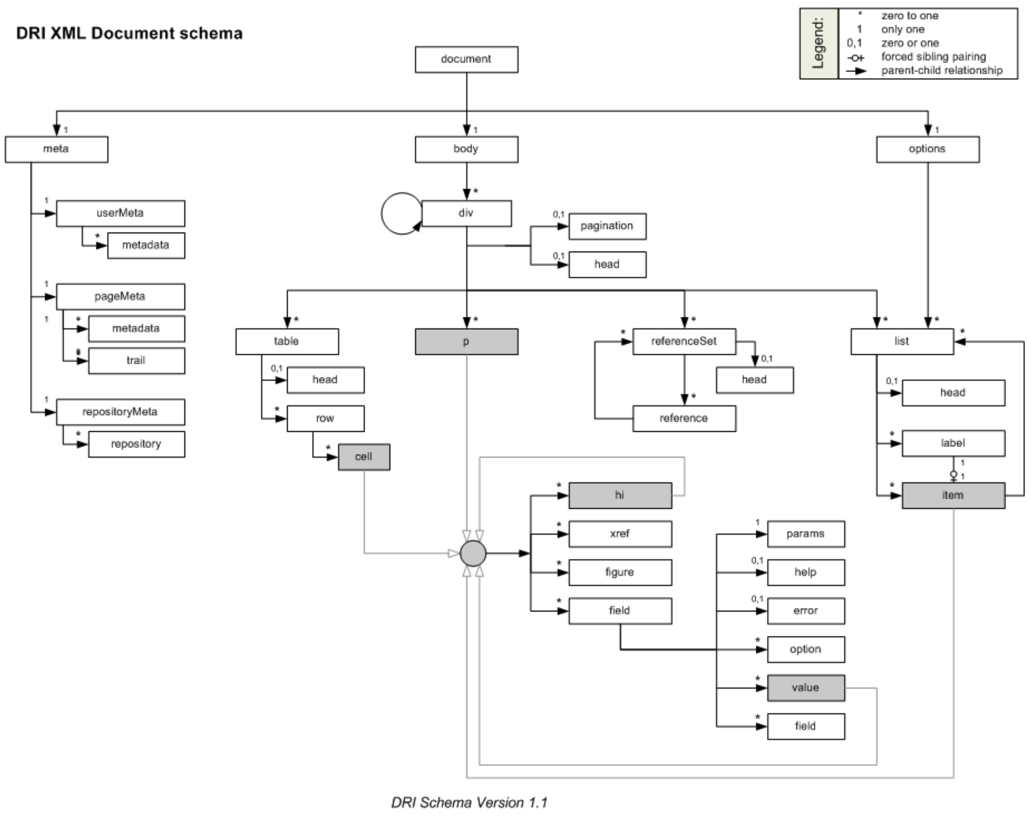

Figure 8 – All the elements in the DRI schema

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.

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. Originaly 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 are included. Thus, a reference element has been added along with referenceSet. These new elements operate like their counterparts in the previous version except refrencing 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.

Manakin produces an independent METS document for all communities, collections, and of course items in the DSpace repository. These documents are generated dynamically and located at the URL "/metadata/handle/<handle-prefix>/<handle-postfix>/mets.xml". As an example with DRI version 1.1, a reference to an item's specific METS document is included. The theme's XSLT will use the document() function to retrieve this document and render it for display on the page within the same XSLT process.

ELEMENT REFERENCE

Element	Attributes (if required, noted)
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
refrence	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 without 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 exit 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 {pageNum} string in the URL mask to generate a full URL to that page.
pagination	Optional Accepted values are "simple", "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 numbers are "1.0" and "1.1". Future iterations of this schema may increment the version number.

```
<document version="1.1">
   <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 posses 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 consisting 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: “available” (meaning the user may jump to the step using the provided xref), “unavailable” (the user has not meet the requirements to jump to the step), or “current” (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 on 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>…</meta>

    <body>…</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>
        <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.

Parent meta

Children metadata (any) trail (any)

Attributes None

```
<meta>

    <userMeta> ... </userMeta>

    <pageMeta>

        <metadata element="title">Examlpe 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 performed 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" operation 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

The reference element is 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">
        <refrence url="/metadata/handle/123/4/mets.xml"
repositoryID="123" type="DSpace Item"/>
        <refrence 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>
<refrence url="/metadata/handle/123/34/mets.xml"/>
```

```
<refrence 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.

Parent meta

Children metadata (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 sub-elements: the raw element which stores the unprocessed value directly from the user of 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 of 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>
```

VERSION HISTORY

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 customization of the available indexes via dspace.cfg and pluggable normalization 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 <meta> 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

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 license
- 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

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
- Customizable 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 previously published items
- 1261191 - Malformed METS metadata exported

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

CHANGES IN DSPACE 1.3.1

BUG FIXES

- 1252153 - Error on fresh install

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 license 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 optimizations
- 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

CHANGES IN DSPACE 1.2.2

GENERAL IMPROVEMENTS

- Customizable 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 authorization
- 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.jsp CHANGED
- dspace-admin/wizard-questions.jsp CHANGED
- search/results.jsp CHANGED
- submit/cancel.jsp CHANGED
- submit/change-file-description.jsp CHANGED
- submit/choose-file.jsp CHANGED
- submit/complete.jsp CHANGED
- submit/creative-commons.jsp CHANGED
- submit/edit-metadata.jsp NEW
- submit/get-file-format.jsp CHANGED
- submit/initial-questions.jsp CHANGED
- submit/progressbar.jsp CHANGED
- submit/review.jsp CHANGED
- submit/select-collection.jsp CHANGED
- submit/show-license.jsp CHANGED
- submit/show-uploaded-file.jsp CHANGED
- submit/upload-error.jsp CHANGED
- submit/upload-file-list.jsp CHANGED

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-home CHANGED
- community-home CHANGED
- display-item CHANGED

- dspace-admin/confirm-delete-collection MOVED TO TOOLS/ AND CHANGED
- dspace-admin/confirm-delete-community MOVED TO TOOLS/ AND CHANGED
- dspace-admin/edit-collection MOVED TO TOOLS/ AND CHANGED
- dspace-admin/edit-community MOVED TO TOOLS/ AND CHANGED
- dspace-admin/index CHANGED
- dspace-admin/upload-logo CHANGED
- dspace-admin/wizard-basicinfo CHANGED
- dspace-admin/wizard-default-item CHANGED
- dspace-admin/wizard-permissions CHANGED
- dspace-admin/wizard-questions CHANGED
- help/formats.html REMOVED
- help/formats CHANGED
- index CHANGED
- layout/navbar-admin CHANGED

CHANGES IN DSPACE 1.2

GENERAL IMPROVMENTS

- 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

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

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.)

APPENDICES

DEFAULT DUBLIN CORE METADATA REGISTRY

Element	**Qualifier**	**Scope Note**
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	lcsh	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.

[1]Used by system: do not remove

Back to contents

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, mpega
audio/x-pn-realaudio	RealAudio	RealAudio file	Known	false	ra, ram
audio/x-wav	WAV	Broadcase Wave Format	Known	false	wav
image/gif	GIF	Graphics Interchange Format	Known	false	gif
image/jpeg	JPEG	Joint Photographic Experts Group/JPEG File Interchange 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 Quicktime	Video Quicktime	Known	false	mov, qt

[1] Used by system: do not remove