



DSpace 5.x Documentation

Author: The DSpace Developer Team
Date: 26 February 2015
URL: <https://wiki.duraspace.org/display/DSDOC5x>

Table of Contents

1	Introduction	7
1.1	Release Notes	8
1.1.1	5.1 Release Notes	8
1.1.2	5.0 Release Notes	10
1.2	Functional Overview	14
1.2.1	Online access to your digital assets	16
1.2.2	Metadata Management	17
1.2.3	Licensing	20
1.2.4	Persistent URLs and Identifiers	21
1.2.5	Getting content into DSpace	22
1.2.6	Getting content out of DSpace	25
1.2.7	User Management	27
1.2.8	Access Control	28
1.2.9	Usage Metrics	30
1.2.10	Digital Preservation	31
1.2.11	System Design	32
2	Installing DSpace	35
2.1	For the Impatient	35
2.2	Hardware Recommendations	36
2.3	Prerequisite Software	36
2.3.1	UNIX-like OS or Microsoft Windows	36
2.3.2	Oracle Java JDK 7 or OpenJDK 7	37
2.3.3	Apache Maven 3.0.5+ (Java build tool)	37
2.3.4	Apache Ant 1.8 or later (Java build tool)	38
2.3.5	Relational Database: (PostgreSQL or Oracle)	38
2.3.6	Servlet Engine (Apache Tomcat 7 or later, Jetty, Caucho Resin or equivalent)	39
2.3.7	Perl (only required for [dspace]/bin/dspace-info.pl)	41
2.4	Installation Instructions	42
2.4.1	Overview of Install Options	42
2.4.2	Overview of DSpace Directories	43
2.4.3	Installation	44
2.5	Advanced Installation	54
2.5.1	'cron' jobs / scheduled tasks	54
2.5.2	Multilingual Installation	54
2.5.3	DSpace over HTTPS	55
2.5.4	The Handle Server	58
2.5.5	Google and HTML sitemaps	60
2.5.6	Statistics	61
2.6	Windows Installation	61

2.7	Checking Your Installation	62
2.8	Known Bugs	62
2.9	Common Problems	62
2.9.1	Common Installation Issues	63
2.9.2	General DSpace Issues	64
3	Upgrading DSpace	66
3.1	Backup your DSpace	67
3.2	Update Prerequisite Software (as necessary)	67
3.3	Upgrade Steps	68
3.4	Troubleshooting Upgrade Issues	73
3.4.1	Manually Upgrading Solr Indexes	73
3.4.2	"Property was circularly defined" errors	75
4	Using DSpace	76
4.1	Authentication and Authorization	76
4.1.1	Authentication Plugins	76
4.1.2	Embargo	97
4.1.3	Managing User Accounts	117
4.1.4	Request a Copy	121
4.2	Exporting Content and Metadata	130
4.2.1	OAI	130
4.2.2	Exchanging Content Between Repositories	149
4.2.3	SWORDv1 Client	150
4.2.4	Linked (Open) Data	152
4.3	Ingesting Content and Metadata	164
4.3.1	Submission User Interface	164
4.3.2	Configurable Workflow	192
4.3.3	Importing and Exporting Content via Packages	204
4.3.4	Importing and Exporting Items via Simple Archive Format	211
4.3.5	Registering Bitstreams via Simple Archive Format	224
4.3.6	Importing Items via basic bibliographic formats (Endnote, BibTex, RIS, TSV, CSV) and online services (OAI, arXiv, PubMed, CrossRef, CiNii)	227
4.3.7	Importing Community and Collection Hierarchy	234
4.3.8	SWORDv1 Server	236
4.3.9	SWORDv2 Server	242
4.3.10	Ingesting HTML Archives	253
4.4	Items and Metadata	254
4.4.1	Authority Control of Metadata Values	254
4.4.2	Batch Metadata Editing	258
4.4.3	DOI Digital Object Identifier	266
4.4.4	Item Level Versioning	275
4.4.5	Mapping Items	287
4.4.6	Metadata Recommendations	289
4.4.7	Moving Items	290

4.4.8	ORCID Integration	291
4.4.9	PDF Citation Cover Page	304
4.4.10	Updating Items via Simple Archive Format	307
4.5	Managing Community Hierarchy	310
4.5.1	Sub-Community Management	310
4.6	Statistics and Metrics	312
4.6.1	DSpace Google Analytics Statistics	312
4.6.2	Elasticsearch Usage Statistics	314
4.6.3	SOLR Statistics	318
4.7	User Interfaces	335
4.7.1	Discovery	335
4.7.2	Localization L10n	357
4.7.3	JSPUI Configuration and Customization	362
4.7.4	XMLUI Configuration and Customization	364
5	System Administration	431
5.1	Introduction to DSpace System Administration	431
5.2	AIP Backup and Restore	432
5.2.1	Background & Overview	433
5.2.2	Running the Code	438
5.2.3	Command Line Reference	451
5.2.4	Configuration in 'dspace.cfg'	457
5.2.5	Common Issues or Error Messages	460
5.2.6	DSpace AIP Format	461
5.3	Ant targets and options	480
5.3.1	Options	481
5.3.2	Targets	482
5.4	Command Line Operations	482
5.4.1	Executing command line operations	483
5.4.2	Available operations	483
5.4.3	Executing streams of commands	485
5.4.4	Database Utilities	485
5.5	Legacy methods for re-indexing content	486
5.5.1	Overview	487
5.5.2	Re-Enabling the legacy Lucene Search and/or DBMS Browse providers	487
5.5.3	Creating the Browse & Search Indexes	488
5.5.4	Running the Indexing Programs	489
5.5.5	Indexing Customization	491
5.6	Mediafilters for Transforming DSpace Content	496
5.6.1	MediaFilters: Transforming DSpace Content	496
5.6.2	ImageMagic Media Filters	504
5.7	Performance Tuning DSpace	505
5.7.1	Give Tomcat (DSpace UIs) More Memory	506
5.7.2	Choosing the size of memory spaces allocated to DSpace	508

5.7.3	Give the Command Line Tools More Memory	509
5.7.4	Give PostgreSQL Database More Memory	510
5.7.5	SOLR Statistics Performance Tuning	510
5.8	Scheduled Tasks via Cron	511
5.8.1	Recommended Cron Settings	511
5.9	Search Engine Optimization	513
5.9.1	Ensuring your DSpace is indexed	513
5.9.2	Google Scholar Metadata Mappings	519
5.10	Troubleshooting Information	520
5.11	Validating CheckSums of Bitstreams	521
5.11.1	Checksum Checker	521
6	DSpace Development	526
6.1	Advanced Customisation	526
6.1.1	Additions module	526
6.1.2	Maven WAR Overlays	526
6.1.3	DSpace Source Release	526
6.1.4	DSpace Service Manager	527
6.2	REST API	530
6.2.1	What is DSpace REST API	530
6.2.2	Introduction to Jersey for developers	536
6.2.3	Configuration for DSpace REST	536
6.2.4	Recording Proxy Access by Tools	537
6.2.5	Deploying the DSpace REST API in your Servlet Container	537
6.2.6	Additional Information	537
6.3	Curation System	537
6.3.1	Changes in 1.8	538
6.3.2	Tasks	539
6.3.3	Activation	539
6.3.4	Writing your own tasks	540
6.3.5	Task Invocation	540
6.3.6	Asynchronous (Deferred) Operation	544
6.3.7	Task Output and Reporting	545
6.3.8	Task Properties	546
6.3.9	Task Annotations	547
6.3.10	Scripted Tasks	548
6.3.11	Bundled Tasks	550
6.3.12	Curation tasks in Jython	558
6.4	Date parser tester	560
7	DSpace Reference	561
7.1	Configuration Reference	561
7.1.1	General Configuration	564
7.1.2	The build.properties Configuration Properties File	565
7.1.3	The dspace.cfg Configuration Properties File	568

7.1.4	Optional or Advanced Configuration Settings	644
7.2	Directories and Files	652
7.2.1	Overview	652
7.2.2	Source Directory Layout	653
7.2.3	Installed Directory Layout	655
7.2.4	Contents of JSPUI Web Application	655
7.2.5	Contents of XMLUI Web Application (aka Manakin)	656
7.2.6	Log Files	656
7.3	Metadata and Bitstream Format Registries	658
7.3.1	Default Dublin Core Metadata Registry (DC)	659
7.3.2	Dublin Core Terms Registry (DCTERMS)	662
7.3.3	Default Bitstream Format Registry	666
7.4	Architecture	668
7.4.1	Overview	668
7.4.2	Application Layer	670
7.4.3	Business Logic Layer	681
7.4.4	DSpace Services Framework	714
7.4.5	Storage Layer	720
7.5	History	726
7.5.1	Changes in 5.x	727
7.5.2	Changes in 4.x	733
7.5.3	Changes in 3.x	744
7.5.4	Changes in 1.8.x	753
7.5.5	Changes in 1.7.x	759
7.5.6	Changes in 1.6.x	767
7.5.7	Changes in 1.5.x	774
7.5.8	Changes in 1.4.x	781
7.5.9	Changes in 1.3.x	784
7.5.10	Changes in 1.2.x	786
7.5.11	Changes in 1.1.x	791
7.6	DSpace Item State Definitions	792

1 Introduction

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

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

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

For people actually running a DSpace service, there is an [installation guide](#), and sections on [configuration](#) and the [directory structure](#).

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](#) 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](#) (Web sites, mailing lists etc.)
 - [Technical FAQ](#)
 - [A list of projects using DSpace](#)
 - [Guidelines for contributing back to DSpace](#)
- www.dspace.org has announcements and contains useful information about bringing up an instance of DSpace at your organization.
- The [DSpace General List](#). Join DSpace-General to ask questions or join discussions about non-technical aspects of building and running a DSpace service. It is open to all DSpace users. Ask questions, share news, and spark discussion about DSpace with people managing other DSpace sites. Watch DSpace-General for news of software releases, user conferences, and announcements from the DSpace Federation.
- The [DSpace Technical List](#). DSpace developers help answer installation and technology questions, share information and help each other solve technical problems through the DSpace-Tech mailing list. Post questions or contribute your expertise to other developers working with the system.

- The [DSpace Development List](#). Join Discussions among DSpace Developers. The DSpace-Devel listserv is for DSpace developers working on the DSpace platform to share ideas and discuss code changes to the open source platform. Join other developers to shape the evolution of the DSpace software. The DSpace community depends on its members to frame functional requirements and high-level architecture , and to facilitate programming, testing, documentation and to the project.

1.1 Release Notes

Online Version of Documentation also available

This documentation was produced with [Confluence](#) software. A PDF version was generated directly from Confluence. An online, updated version of this 5.x Documentation is also available at: <https://wiki.duraspace.org/display/DSDOC5x>

Welcome to Release 5.1, a security and bug-fix release for the DSpace 5.x platform. For information on upgrading to DSpace 5, please see [Upgrading DSpace](#).

1.1.1 5.1 Release Notes

We highly recommend any users of DSpace 5.x upgrade to 5.1

DSpace 5.1 contains security fixes for both the XMLUI and JSPUI. To ensure your 5.x site is secure, **we highly recommend all DSpace 5.x users upgrade to DSpace 5.1.**

We also highly recommend removing any "allowLinking=true" settings from your Tomcat's <Context> configuration. Previously our installation documentation erroneously listed examples which included "allowLinking=true", while the [Tomcat documentation lists it as a possible security concern](#). The XMLUI Directory Traversal Vulnerability (see below) is also exacerbated by this setting.

DSpace 1.x.x, 3.x or 4.x users may wish to consider upgrading directly to DSpace 5.1

Several of the security vulnerabilities patched in DSpace 5.1 (and backported to 4.3 and 3.4) also affect sites running unsupported DSpace 1.x.x releases. In order to ensure your site is patched, we highly recommend upgrading to [DSpace 3.4](#), [DSpace 4.3](#) or DSpace 5.1.

If you are considering an upgrade from DSpace 1.x.x, note that, as of DSpace 5, your existing data (i.e. database contents, search/browse indexes) will now be automatically upgraded from ANY prior version of DSpace. Therefore, you may wish to consider upgrading directly to DSpace 5.1, as the [5.x upgrade process](#) is simplified.

DSpace 5.1 is a security and bug fix release to resolve several issues located in DSpace 5.0. As it only provides only bug fixes, DSpace 5.1 should constitute an easy upgrade from DSpace 5.0 for most users. No database changes or additional configuration changes should be necessary when upgrading from DSpace 5.0 to 5.1.

This release addresses the following security issues discovered in DSpace 5.x and below:

- XMLUI Security Fixes
 - *[HIGH SEVERITY] XMLUI Directory Traversal Vulnerabilities* ([DS-2445](#) - requires a JIRA account to access for two weeks, and then will be public): These vulnerabilities allow someone to potentially access *any file* on your local filesystem which is readable to the Tomcat user account. This includes files which are unrelated to DSpace or Tomcat, but are readable to all users on the filesystem (e.g. `/etc/passwd`, `/etc/hosts`, etc.). This also includes Tomcat configuration files (which may or may not contain passwords). These vulnerabilities have existed since DSpace 1.5.2.
 - Discovered by: [Khalil Shreateh](#), with additional (related) vulnerabilities discovered by the DSpace Committer Team
 - In some configurations of Tomcat, simply removing any "allowLinking=true" settings from your Tomcat's `<Context>` configuration will limit the directory traversal vulnerability's severity to only allow access to files within the XMLUI web application directory. In addition, the [Tomcat documentation details "allowLinking=true" as a possible security concern](#). However, you still must upgrade or patch your DSpace in order to completely resolve this vulnerability.

- JSPUI Security Fixes
 - *[MEDIUM SEVERITY] JSPUI Directory Traversal Vulnerability (DS-2448 - requires a JIRA account to access for two weeks, and then will be public):* This vulnerability allows someone to potentially access any file within the JSPUI web application directory (e.g. WEB-INF/web.xml). This vulnerability is believed to have existed in all prior versions of DSpace.
 - Discovered by [Khalil Shreateh](#)
 - *[LOW SEVERITY] Cross-site scripting (XSS injection) is possible in JSPUI Recent Submissions listings (DS-1702 - requires a JIRA account to access for two weeks, and then will be public):* This vulnerability could allow a depositor/submitter to embed dangerous Javascript code into the metadata of a new submission, thus causing that code to be run across other user accounts. However, this vulnerability is only possible by someone with privileges to add content to your DSpace site. This vulnerability has existed since DSpace 1.5.x.
 - Discovered by: Jean-Paul Zhao of [University of Toronto](#)
 - *[LOW SEVERITY] Cross-site scripting (XSS injection) is possible in JSPUI Discovery search form (DS-2044 - requires a JIRA account to access for two weeks, and then will be public):* This vulnerability could allow someone to embed dangerous Javascript code into links to search results. If a user was emailed such a link and clicked it, the javascript would be run in their local browser. This vulnerability has existed since DSpace 3.x
 - 4.x / 5.x vulnerability discovered by Gabriela Mircea of [McMaster University](#) and [Khalil Shreateh](#)
 - 3.x vulnerability discovered by Iyas Orak of [Biznet Bilisim A.S.](#)

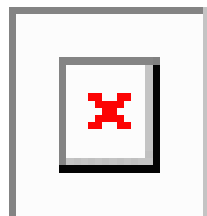
In addition, this release fixes a variety of minor bugs in the 5.0 release. For more information, see the [Changes in 5.x](#) page.

1.1.2 5.0 Release Notes

The following is a list of the new features included for the 5.x platform (not an exhaustive list):

DSpace 5.0 ships with a number of new features. Certain features are automatically enabled by default while others require deliberate activation.





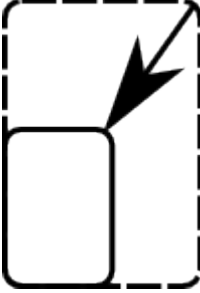


The following non-exhaustive list contains the major new features in 5.0






Easier [Upgrading to 5.x](#) from ANY previous DSpace version (1.x.x, 3.x or 4.x).

- Your underlying DSpace database now upgrades itself automatically when you first run a newer version of DSpace (see [DS-2167](#), by Tim Donohue with support/feedback from 5.0 Release Team).
- Solr/Lucene indexes now upgrade automatically during the "ant update" step of the Upgrade process (see [DS-2297](#) by Tim Donohue, with support/feedback from Ivan Masár and 5.0 Release Team).

	<p>Perform Batch Imports from the User Interface (in both XMLUI and JSPUI)</p> <ul style="list-style-type: none"> XMLUI version (XMLUI Documentation, also see DS-1641) by Peter Dietz with the support of Ohio State University Libraries and Longsight JSPUI version (JSPUI Documentation, also see DS-2177) by the Greek National Documentation Centre/EKT
   	<p>XMLUI new features</p> <ul style="list-style-type: none"> Mirage 2 Responsive Theme, based on Bootstrap (<i>disabled by default</i>, see DS-2052 for screenshots) by @mire ORCID Integration (<i>disabled by default</i>, see DS-2049) by @mire Report Google Analytics statistics from Admin UI (<i>disabled by default</i>, see DS-2108) by Robin Taylor Track file downloads in Google Analytics statistics (see DS-2008) by Robin Taylor Autogenerate PDF citation "cover pages" for all PDFs (<i>disabled by default</i>, see DS-2175) by Peter Dietz with the support of Ohio State University Libraries and Longsight Sherpa/Romeo lookup during item submission process, see DS-2053 by Kevin Van de Velde and Bert Vanderhallen with the support of @mire (previously available only in JSPUI) Rendering MathML code in abstracts using MathJax, see DS-635 by Peter Dietz with the support of Ohio State University Libraries and Longsight Ensure "page not found" error pages use configured theme, see DS-1596 by Tim Donohue with the support of DuraSpace Performance improvements for "Select Collection" dropdown in submission process, see DS-682 by Peter Dietz with the support of Ohio State University Libraries and Longsight
	<p>JSPUI new features</p> <ul style="list-style-type: none"> Drag and drop file upload (using HTML5), see DS-1994 by Pascal-Nicolas Becker with the support of TU Berlin Item Visual Indicators in Browse/Search results, see DS-2162 by Greek National Documentation Centre/EKT Track file downloads in Google Analytics statistics (see DS-2008) by Robin Taylor

 	<p>REST API new features</p> <ul style="list-style-type: none"> • DSpace REST API now includes CRUD (Create/Read/Update/Delete) endpoints, see DS-2168 by the Czech Technical University in Prague
	<p>RDF Interface to support Linked (Open) Data (NEW)</p> <ul style="list-style-type: none"> • DSpace can now provide its content as Linked (Open) Data via a new RDF interface (provided as an "rdf" webapp), see DS-2061 by Pascal -Nicolas Becker
	<p>OAI-PMH interface enhancements / bug fixes</p> <ul style="list-style-type: none"> • OpenAIRE v3 compliance (operators over filters) • OAI respects item READ rights • /oai displays the list of available contexts; contexts have descriptions <p>See DS-1649 by João Melo</p>
	<p>Enhanced Thumbnail Quality (<i>disabled by default</i>)</p> <ul style="list-style-type: none"> • Enhanced image thumbnails can now be generated using ImageMagick • Enhanced PDF Thumbnails can now be generated using ImageMagick and Ghostscript <p>See DS-2105 by Terry Brady with the support of Georgetown University</p>
	<p>Bug fixes / improvements to Biblio-Transformation-Engine (BTE)</p> <ul style="list-style-type: none"> • BTE: batch import from various bibliographic formats was upgrade to the latest version (see DS-2183) <p>Kindly contributed by the Greek National Documentation Centre/EKT</p>
	<p>Enhancements to DOI Support (<i>disabled by default</i>)</p> <ul style="list-style-type: none"> • Enhanced EZID IdentifierProvider Metadata Mapping via XSLT, see DS-2119 by Mohamed Mohideen Abdul Rasheed

	<p>Apache Solr libraries were upgraded for all interfaces (JSPUI, XMLUI, and OAI)</p> <p>See DS-2253 by Roeland Dillen with the support of @mire</p>
	<p>Add a place for third-party JARs / plugins to be "found" by DSpace (<i>disabled by default</i>)</p> <ul style="list-style-type: none"> • DSpace will now look for JARs / plugins in the locations specified by "plugin.classpath" value specified in <code>dspace.cfg</code>. <p>See DS-2107 by Mark H. Wood with the support of IUPUI University Library</p>
	<p>All objects now have metadata support</p> <ul style="list-style-type: none"> • All DSpace objects (Communities, Collections, Items, EPeople, Groups) now have metadata, and most now use the default "dc" (Dublin Core) metadata schema. <ul style="list-style-type: none"> • NOTE: The only exception is EPeople metadata, which is stored in a new "eperson" metadata schema. • The User Interfaces don't yet take advantage of this enhancement in DSpace 5.0. Instead, this is an internal restructuring of data within DSpace. In the future, this provides the potential to create more enhanced metadata (or even more configurable metadata) on all objects <p>See DS-1582 by Mark H. Wood with the support of IUPUI University Library and Kevin Van de Velde with the support of @mire</p>

A full list of all changes / bug fixes in 5.x is available in the [Changes in 5.x](#) section.

The following individuals have contributed directly to this release of DSpace: Adan Roman, Àlex Magaz Graça , Andrea Bollini, Andrea Schweer, Antoine Snyers, Art Lowel, Artur Konczak, Bavo Van Geit, Bram Luyten, Christian Scheible, Christian Völker, Christos Rhodosthenous, Claudia Jürgen , CTU Developers, Denis Fdz, Ed Goulet, Eliana de Mattos Pinto Coelho, Elvi S. Nemiz, Emilio Lorenzo, George Simeonov, Graham Triggs , Hardy Pottinger, Ivan Masár, James Halliday, João Melo, Jon Gibson , Jordan Pišanc, Jozef M. , Keiji Suzuki, Kevin Van de Velde, Kostas Stamatis, Luigi Andrea Pascarelli, Marina Muilwijk, Mark Diggory, Mark H. Wood, Mohamed Mohideen Abdul Rasheed, Monika Mevenkamp, Ondej Košarko, Panagiotis Koutsourakis , Pascal-Nicolas Becker, Pauline Ward, Paulo Graça , Peter Dietz, Petya Kohts, Philip Vissenaekens, Robert Faling, Robin Taylor, Roeland Dillen, Royopa, Sonmez CELIK, Terry Brady, Thanos Kyritsis, Thomas Misilo, Tiago Murakami, Tim Donohue, and others who reviewed and commented on their work. Many of these could not do this work without the support (release time and financial) of their associated institutions. We offer thanks to those institutions for supporting their staff to take time to contribute to the DSpace project.

A big thank you also goes out to the [DSpace Community Advisory Team](#) (DCAT), who helped the developers to prioritize and plan out several of the new features that made it into this release. The current DCAT members include: Augustine Gitonga, Bram Luyten, Bharat Chaudhari, Claire Bundy, Dibyendra Hyoju, Elin Stangeland, Felicity A Dykas, Iryna Kuchma, James Evans, Jim Ottaviani, Kate Dohe, Kathleen Schweitzberger, Leonie Hayes, Lilly Li, Maureen Walsh, Pauline Ward, Roger Weaver, Sarah Molloy, Sarah Potvin, Sarah Shreeves, Steve Van Tuyl, Terry Brady, Valorie Hollister and Yan Han.

We apologize to any contributor accidentally left off this list. DSpace has such a large, active development community that we sometimes lose track of all our contributors. Our ongoing list of all known people/institutions that have contributed to DSpace software can be found on our [DSpace Contributors page](#). Acknowledgments to those left off will be made in future releases.

Want to see your name appear in our list of contributors? All you have to do is report an issue, fix a bug, improve our documentation or help us determine the necessary requirements for a new feature! Visit our [Issue Tracker](#) to report a bug, or join [dspace-devel mailing list](#) to take part in development work. If you'd like to help improve our current documentation, please get in touch with one of our [Committers](#) with your ideas. You don't even need to be a developer! Repository managers can also get involved by volunteering to join the [DSpace Community Advisory Team](#) and helping our developers to plan new features.

The Release Team consisted of:

- Peter Dietz (Longsight)
- Hardy Pottinger (U of Missouri)
- Ivan Masár
- Mark H. Wood (Indiana University)
- Robin Taylor (University of Edinburgh)
- Pascal-Nicolas Becker (Technische Universität Berlin)

Additional thanks to Tim Donohue from DuraSpace for keeping all of us focused on the work at hand, for calming us when we got excited, and for the general support for the DSpace project.

1.2 Functional Overview

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

- 1 [Online access to your digital assets](#)
 - 1.1 [Full-text search](#)
 - 1.2 [Navigation](#)
 - 1.3 [Supported file types](#)
 - 1.4 [Optimized for Google Indexing](#)
 - 1.5 [OpenURL Support](#)
 - 1.6 [Support for modern browsers](#)

- 2 [Metadata Management](#)
 - 2.1 [Metadata](#)
 - 2.2 [Choice Management and Authority Control](#)
- 3 [Licensing](#)
 - 3.1 [Collection and Community Licenses](#)
 - 3.2 [License granted by the submitter to the repository](#)
 - 3.3 [Creative Commons Support for DSpace Items](#)
- 4 [Persistent URLs and Identifiers](#)
 - 4.1 [Handles](#)
 - 4.2 [Bitstream 'Persistent' Identifiers](#)
- 5 [Getting content into DSpace](#)
 - 5.1 [The Manual DSpace Submission and Workflow System](#)
 - 5.1.1 [Workflow Steps](#)
 - 5.1.2 [Submission Workflow in DSpace](#)
 - 5.2 [Command line import facilities](#)
 - 5.3 [Registration for externally hosted files](#)
- 6 [Getting content out of DSpace](#)
 - 6.1 [OAI Support](#)
 - 6.2 [SWORD Support](#)
 - 6.3 [Command Line Export Facilities](#)
 - 6.4 [Packager Plugins](#)
 - 6.5 [Crosswalk Plugins](#)
 - 6.6 [Supervision and Collaboration](#)
- 7 [User Management](#)
 - 7.1 [User Accounts \(E-Person\)](#)
 - 7.2 [Subscriptions](#)
 - 7.3 [Groups](#)
- 8 [Access Control](#)
 - 8.1 [Authentication](#)
 - 8.2 [Authorization](#)
- 9 [Usage Metrics](#)
 - 9.1 [Item, Collection and Community Usage Statistics](#)
 - 9.2 [System Statistics](#)
- 10 [Digital Preservation](#)
 - 10.1 [Checksum Checker](#)
- 11 [System Design](#)
 - 11.1 [Data Model](#)
 - 11.2 [Storage Resource Broker \(SRB\) Support](#)

1.2.1 Online access to your digital assets

The online presentation of your content in an organized tree of Community and Collections is a main feature of DSpace. Users can access pages for individual items, these are metadata descriptions together with files available for download.

Full-text search

DSpace can process uploaded text based contents for full-text searching. This means that not only the metadata you provide for a given file will be searchable, but all of its contents will be indexed as well. This allows users to search for specific keywords that only appear in the actual content and not in the provided description.

Navigation

DSpace allows users to find their way to relevant content in a number of ways, including:

- **Searching** for one or more keywords in metadata or extracted full-text
- **Faceted browsing** through any field provided in the item description.
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](#). 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.
- Through **external reference**, such as a Handle

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.

Supported file types

DSpace can accommodate any type of uploaded file. While DSpace is most known for hosting text based materials including scholarly communication and electronic theses and dissertations (ETDs), there are many stakeholders in the community who use DSpace for multimedia, data and learning objects. While some restrictions apply, DSpace can even serve as a store for [HTML Archives](#).

Files that have been uploaded to DSpace are often referred to as "Bitstreams". The reason for this is mainly historic and tracks back to the technical implementation. After ingestion, files in DSpace are stored on the file system as a stream of bits without the file extension.

Optimized for Google Indexing

The Duraspace community fosters a close relation with Google to ensure optimal indexing of DSpace content, primarily in the Google Search and Google Scholar products. For the purpose of Google Scholar indexing, DSpace added specific metadata in the page head tags facilitating indexing in Scholar. More information can be retrieved on the [Google Scholar Metadata Mappings page](#). Popular DSpace repositories often generate over 60% of their visits from Google pages.

OpenURL Support

DSpace supports the [OpenURL protocol](#) from [SFX](#), 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.

Support for modern browsers

The DSpace developer community aims to rely on modern web standards and well tested libraries where possible. As a rule of thumb, users can expect that the DSpace web interfaces work on modern web browsers. DSpace developers routinely test new interface developments on recent versions of Firefox, Safari and Chrome. Because of fast moving, automatic, incremental updates to these browsers, support is no longer targeted at specific versions of these browsers. The community attempts to support the latest official version and up to 2 older versions of Microsoft's Internet Explorer.

In some cases, modern interfaces are developed alongside older interfaces that no longer receive active maintenance or improvements. This is particularly true for the original themes for the XML User Interface such as "Kubrick", "Classic" and "Reference". These themes still reside in the code base but are not optimized for modern browsers.

1.2.2 Metadata Management

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](#) set of elements and qualifiers is provided by default. The [set of elements and qualifiers used by MIT Libraries](#) 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 relational 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, a bitstream also has 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.

Choice Management and Authority Control

This is a configurable framework that lets you define plug-in classes to control the choice of values for a given DSpace metadata fields. It also lets you configure fields to include "authority" values along with the textual metadata value. The choice-control system includes a user interface in both the Configurable Submission UI and the Admin UI (edit Item pages) that assists the user in choosing metadata values.

Introduction and Motivation

Definitions

Choice Management

This is a mechanism that generates a list of choices for a value to be entered in a given metadata field. Depending on your implementation, the exact choice list might be determined by a proposed value or query, or it could be a fixed list that is the same for every query. It may also be closed (limited to choices produced internally) or open, allowing the user-supplied query to be included as a choice.

Authority Control

This works in addition to choice management to supply an authority key along with the chosen value, which is also assigned to the Item's metadata field entry. Any authority-controlled field is also inherently choice-controlled

About Authority Control

The advantages we seek from an authority controlled metadata field are:

1. **There is a simple and positive way to test whether two values are identical**, by comparing authority keys.
 - Comparing plain text values can give false positive results e.g. when two different people have a name that is written the same.
 - It can also give false negative results when the same name is written different ways, e.g. "J. Smith" vs. "John Smith".
2. **Help in entering correct metadata values.** The submission and admin UIs may call on the authority to check a proposed value and list possible matches to help the user select one.
3. **Improved interoperability.** By sharing a name authority with another application, your DSpace can interoperate more cleanly with other applications.
 - For example, a DSpace institutional repository sharing a naming authority with the campus social network would let the social network construct a list of all DSpace Items matching the shared author identifier, rather than by error-prone name matching.
 - When the name authority is shared with a campus directory, DSpace can look up the email address of an author to send automatic email about works of theirs submitted by a third party. That author does not have to be an EPerson.
4. Authority keys are normally invisible in the public web UIs. They are only seen by administrators editing metadata. The value of an authority key is not expected to be meaningful to an end-user or site visitor. Authority control is different from the controlled vocabulary of keywords already implemented in the submission UI:

1. **Authorities are external to DSpace.** The source of authority control is typically an external database or network resource.
 - Plug-in architecture makes it easy to integrate new authorities without modifying any core code.
2. This authority proposal impacts all phases of metadata management.
 - The keyword vocabularies are only for the submission UI.
 - Authority control is asserted everywhere metadata values are changed, including unattended/ batch submission, LNI and SWORD package submission, and the administrative UI.

Some Terminology

Authority	An authority is a source of fixed values for a given domain, each unique value identified by a key.
.	For example, the OCLC LC Name Authority Service.

Authority Record	The information associated with one of the values in an authority; may include alternate spellings and equivalent forms of the value, etc.
Authority Key	An opaque, hopefully persistent, identifier corresponding to exactly one record in the authority.

1.2.3 Licensing

DSpace offers support for licenses on different levels

Collection and Community Licenses

Each community and collection in the hierarchy of a DSpace repository can contain its own license terms. This allows an institution to use the repository both for collections where certain rights are reserved and others from which the content may be accessed and distributed more freely.

License granted by the submitter to the repository

At the end of the manual submission process, the submitter is asked to grant the repository service an appropriate distribution license. This license can be easily customized on a per collection basis. In its most common form, the submitter grants to the repository service a non-exclusive distribution license, meaning that he officially gives the repository service the right to share his or her work with the world.

Creative Commons Support for DSpace Items

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](#). Support for license selection is controlled by a site-wide configuration option, and since license selection involves interaction with 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, metadata and (optionally) a copy of the license text 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. For specifics of how to configure and use Creative Commons licenses, see the configuration section.

1.2.4 Persistent URLs and Identifiers

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 in the long term. To help solve this problem, a core DSpace feature is the creation of a persistent identifier for every item, collection and community stored in DSpace. To persist identifiers, DSpace requires a storage- and location- independent mechanism for creating and maintaining identifiers. DSpace uses the [CNRI Handle System](#) 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 unique 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](#), 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:

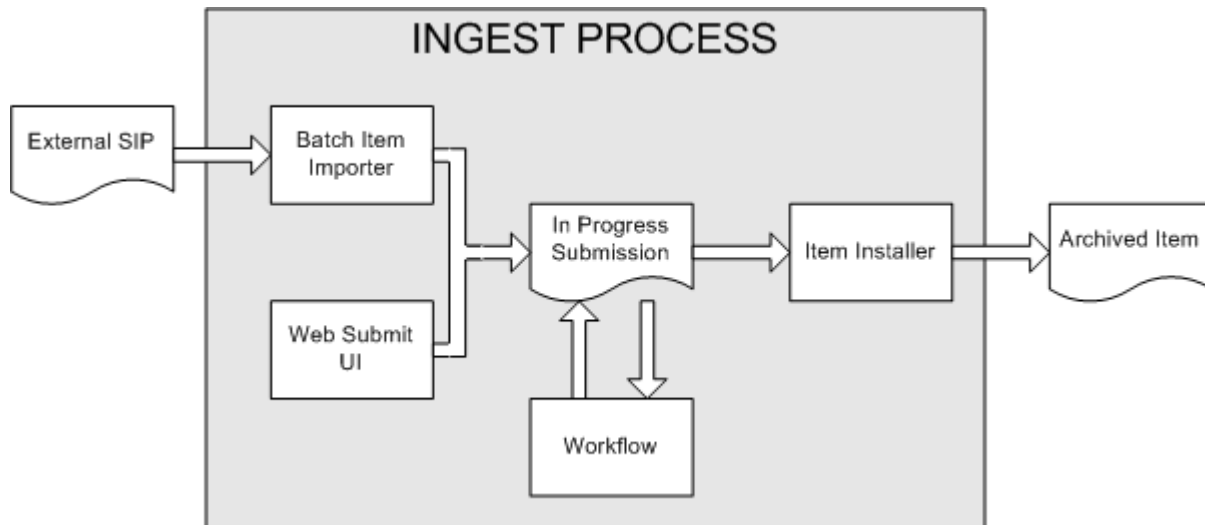
```
https://dspace.myu.edu/bitstream/123.456/789/24/foo.html
```

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.

1.2.5 Getting content into DSpace

The Manual DSpace Submission and Workflow System

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



DSpace Ingest Process

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

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

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

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

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

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.

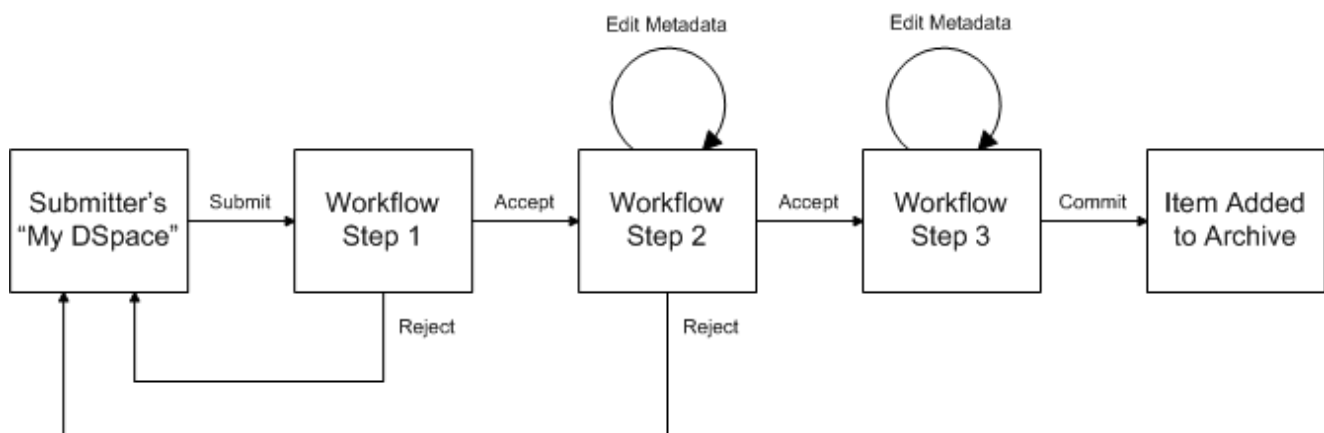
In other words, the sequence is this: The collection receives a submission. If the collection has a group assigned for workflow step 1, that step is invoked, and the group is notified. Otherwise, workflow step 1 is skipped.

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

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

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



Submission Workflow in DSpace

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

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

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

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

Command line import facilities

DSpace includes batch tools to import 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. For more information see [Item Importer and Exporter](#).

DSpace also includes various package importer tools, which support many common content packaging formats like METS. For more information see [Package Importer and Exporter](#).

Registration for externally hosted files

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.

1.2.6 Getting content out of DSpace

OAI Support

The [Open Archives Initiative](#) has developed a [protocol for metadata harvesting](#). 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](#) 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.

SWORD Support

SWORD (Simple Web-service Offering Repository Deposit) is a protocol that allows the remote deposit of items into repositories. SWORD was further developed in SWORD version 2 to add the ability to retrieve, update, or delete deposits. DSpace supports the SWORD protocol via the 'sword' web application and SWord v2 via the swordv2 web application. The specification and further information can be found at <http://swordapp.org>.

Command Line Export Facilities

DSpace includes batch tools to 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. For more information see [Item Importer and Exporter](#).

DSpace also includes various package exporter tools, which support many common content packaging formats like METS. For more information see [Package Importer and Exporter](#).

Packager Plugins

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

A package is typically an archive file such as a Zip or "tar" file, including a *manifest* document which contains metadata and a description of the package contents. The [IMS Content Package](#) 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.

More information about calling Packagers to ingest or disseminate content can be found in the [Package Importer and Exporter](#) section of the System Administration documentation.

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.

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-theses, 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.

1.2.7 User Management

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.

User Accounts (E-Person)

DSpace holds 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, if LDAP authentication is used for this E-Person.

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.

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.

1.2.8 Access Control

Authentication

Authentication is when an application session positively identifies itself as belonging to an E-Person and/or Group. In DSpace 1.4 and later, 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 built-in groups: 'Administrators', who can do anything in a site, and 'Anonymous', which is a list that contains all users. Assigning a policy for an action on an object to anonymous means giving everyone permission to do that action. (For example, most objects in DSpace sites have a policy of 'anonymous' READ.) Permissions must be explicit - lack of an explicit permission results in the default policy of 'deny'. Permissions also do not 'commute'; for example, if an e-person has READ permission on an item, they might not necessarily have READ permission on the bundles and bitstreams in that item. Currently Collections, Communities and Items are discoverable in the browse and search systems regardless of READ authorization.

The following actions are possible:

Collection

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

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.

1.2.9 Usage Metrics

DSpace is equipped with SOLR based infrastructure to log and display pageviews and file downloads.

Item, Collection and Community Usage Statistics

Usage statistics can be retrieved from individual item, collection and community pages. These Usage Statistics pages show:

- Total page visits (all time)
- Total Visits per Month
- File Downloads (all time)*
- Top Country Views (all time)
- Top City Views (all time)

*File Downloads information is only displayed for item-level statistics. Note that downloads from separate bitstreams are also recorded and represented separately. DSpace is able to capture and store File Download information, even when the bitstream was downloaded from a direct link on an external website.

Total Visits	
	Views
DSUG 2009	790

Total Visits Per Month							
	August 2009	September 2009	October 2009	November 2009	December 2009	January 2010	February 2010
DSUG 2009	0	0	491	82	151	59	7

System Statistics

Various statistical reports 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.

The report includes following sections

- A customizable general overview of activities in the archive, by default including:
 - Number of items archived
 - Number of bitstream views
 - Number of item page views
 - Number of collection page views
 - Number of community page views
 - Number of user logins
 - Number of searches performed
 - Number of license rejections
 - Number of OAI Requests
- Customizable summary of archive contents
- Broken-down list of item viewings
- A full break-down of all performed actions
- User logins
- Most popular searches
- Log Level Information
- Processing information!stats_genrl_overview.png!

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.

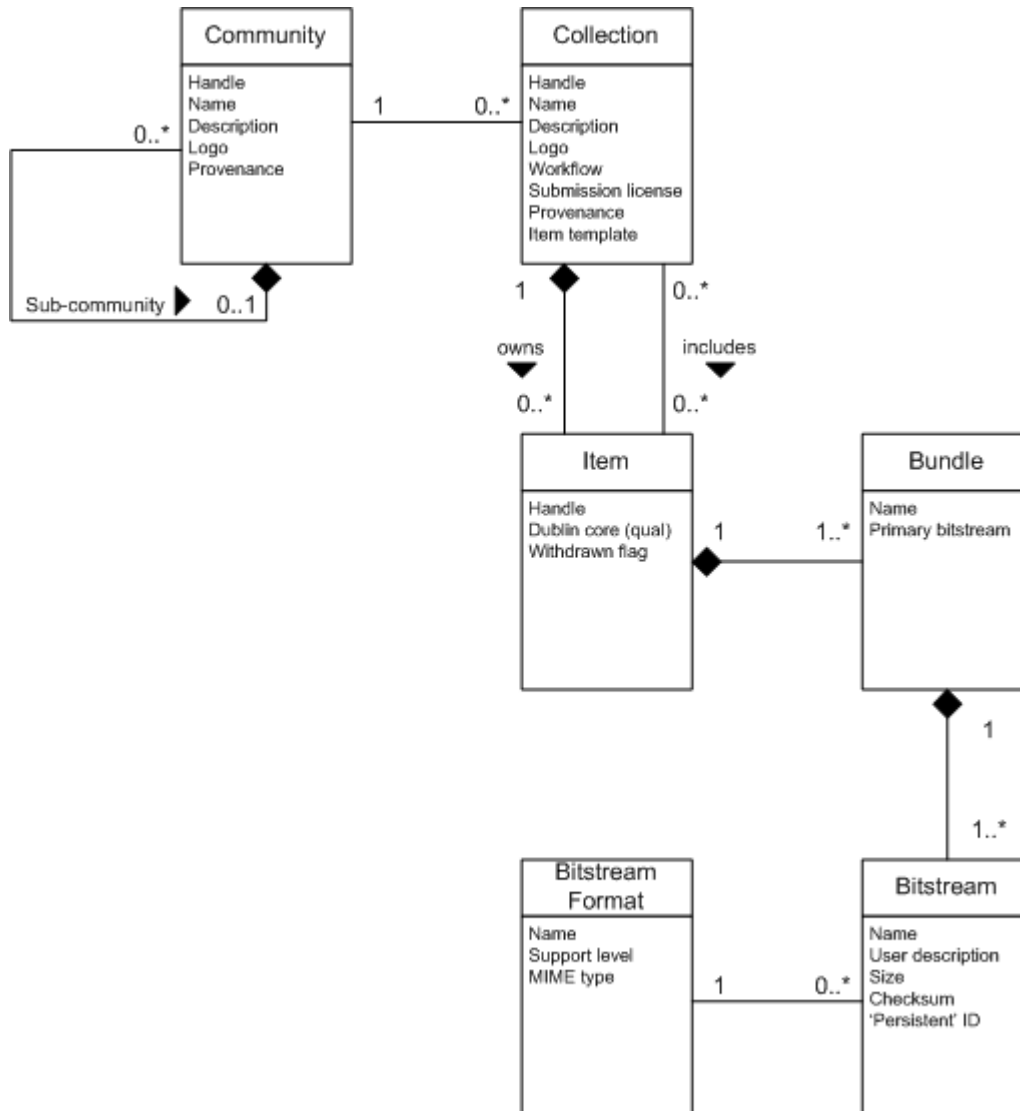
1.2.10 Digital Preservation

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.

1.2.11 System Design

Data Model



Data Model Diagram

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

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

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

Items are further subdivided into named *bundles of bitstreams*. Bitstreams are, as the name suggests, streams of bits, usually ordinary computer files. Bitstreams that are somehow closely related, for example HTML files and images that compose a single HTML document, are 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) associated with the item. This license specifies what end users downloading the content can do with the content

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

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

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

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

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

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

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). Both are achieved using a simple, lightweight API.

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

2 Installing DSpace

- [1 For the Impatient](#)
- [2 Hardware Recommendations](#)
- [3 Prerequisite Software](#)
 - [3.1 UNIX-like OS or Microsoft Windows](#)
 - [3.2 Oracle Java JDK 7 or OpenJDK 7](#)
 - [3.2.1 Elasticsearch requirements for Java](#)
 - [3.3 Apache Maven 3.0.5+ \(Java build tool\)](#)
 - [3.3.1 Configuring a Proxy](#)
 - [3.4 Apache Ant 1.8 or later \(Java build tool\)](#)
 - [3.5 Relational Database: \(PostgreSQL or Oracle\)](#)
 - [3.6 Servlet Engine \(Apache Tomcat 7 or later, Jetty, Caucho Resin or equivalent\)](#)
 - [3.7 Perl \(only required for \[dSPACE\]/bin/dSPACE-info.pl\)](#)
- [4 Installation Instructions](#)
 - [4.1 Overview of Install Options](#)
 - [4.2 Overview of DSpace Directories](#)
 - [4.3 Installation](#)
- [5 Advanced Installation](#)
 - [5.1 'cron' jobs / scheduled tasks](#)
 - [5.2 Multilingual Installation](#)
 - [5.3 DSpace over HTTPS](#)
 - [5.3.1 Enabling the HTTPS support in Tomcat 7.0](#)
 - [5.3.2 Using SSL on Apache HTTPD with mod_jk](#)
 - [5.4 The Handle Server](#)
 - [5.4.1 Updating Existing Handle Prefixes](#)
 - [5.5 Google and HTML sitemaps](#)
 - [5.6 Statistics](#)
- [6 Windows Installation](#)
- [7 Checking Your Installation](#)
- [8 Known Bugs](#)
- [9 Common Problems](#)
 - [9.1 Common Installation Issues](#)
 - [9.2 General DSpace Issues](#)

2.1 For the Impatient

Since some users might want to get their test version up and running as fast as possible, offered below is an *unsupported* outline of getting DSpace to run quickly in a Unix-based environment using the DSpace source release.

⊖ Only experienced unix admins should even attempt the following without going to the detailed [Installation Instructions](#)

```
useradd -m dspace
gzip xzf dspace-5.x-src-release.tar.gz
createuser --username=postgres --no-superuser --pwprompt dspace
createdb --username=postgres --owner=dspace --encoding=UNICODE dspace
cd [dspace-source]
vi build.properties
mkdir [dspace]
chown dspace [dspace]
su - dspace
cd [dspace-source]/dspace
mvn package
cd [dspace-source]/dspace/target/dspace-installer
ant fresh_install
cp -r [dspace]/webapps/* [tomcat]/webapps
/etc/init.d/tomcat start
[dspace]/bin/dspace create-administrator
```

2.2 Hardware Recommendations

You can install and run DSpace on most modern PC, laptop or server hardware. However, if you intend to run DSpace for a large community of potential end users, carefully review the [Hardware Recommendations](#).

2.3 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.

2.3.1 UNIX-like OS or Microsoft Windows


- UNIX-like OS (Linux, HP/UX, Mac OSX, 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 distribution's documentation or local system administrators to determine what is already available.

- Microsoft Windows: After verifying all prerequisites below, see the [Windows Installation](#) section for Windows tailored instructions

2.3.2 Oracle Java JDK 7 or OpenJDK 7

Oracle's Java can be downloaded from the following location: <http://www.oracle.com/technetwork/java/javase/downloads/index.html>. The Java SE JDK version is fine, J2EE is not required for DSpace.

OpenJDK download and installation instructions can be found here <http://openjdk.java.net/>.

 Be aware that Tomcat 7 uses Java 1.6 to compile JSPs by default. See information about Tomcat below on how to configure it to use Java 1.7 for JSPs. Tomcat 8 uses Java 1.7 for JSPs by default. If you use another Servlet Container please refer to its documentation on this matter.

Elasticsearch requirements for Java

Elasticsearch (used in DSpace for [Elasticsearch Usage Statistics](#), an optional feature) has its own recommendations regarding Java flavour and version:

<http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/setup.html>

"Elasticsearch is built using Java, and requires at least [Java 7](#) in order to run. Only Oracle's Java and the OpenJDK are supported.

We recommend installing the **Java 8 update 20 or later**, or **Java 7 update 55 or later**. Previous versions of Java 7 are known to have bugs that can cause index corruption and data loss."

2.3.3 Apache Maven 3.0.5+ (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>

Configuring a Proxy

You can configure a proxy to use for some or all of your HTTP requests in Maven. The username and password are only required if your proxy requires basic authentication (note that later releases may support storing your passwords in a secured keystore, in the mean time, please ensure your `settings.xml` file (usually `/${user.home}/.m2/settings.xml`) is secured with permissions appropriate for your operating system).

Example:

```
<settings>
.
.
<proxies>
  <proxy>
    <active>true</active>
    <protocol>http</protocol>
    <host>proxy.somewhere.com</host>
    <port>8080</port>
    <username>proxyuser</username>
    <password>somepassword</password>
    <nonProxyHosts>www.google.com|*.somewhere.com</nonProxyHosts>
  </proxy>
</proxies>
.
.
</settings>
```

2.3.4 Apache Ant 1.8 or later (Java build tool)

Apache Ant is 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-installer` 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>

2.3.5 Relational Database: (PostgreSQL or Oracle)

- **PostgreSQL 9.0 or later:** PostgreSQL can be downloaded from <http://www.postgresql.org/>. Unicode (specifically UTF-8) support must be enabled (but this is enabled by default). Once installed, you need to enable TCP/IP connections (DSpace uses JDBC):
 - In `postgresql.conf`: uncomment the line starting: `listen_addresses = 'localhost'`. This is the default, in recent PostgreSQL releases, but you should at least check it.
 - Then tighten up security a bit by editing `pg_hba.conf` and adding this line: `host dspace dspace 127.0.0.1 255.255.255.255 md5`. This should appear *before* any lines matching `all` databases, because the first matching rule governs.
 - Then restart PostgreSQL.

- **Oracle 10g or later:** Details on acquiring Oracle can be downloaded from the following location: <http://www.oracle.com/database/>. You will need to create a database for DSpace. Make sure that the character set is one of the Unicode character sets. DSpace uses UTF-8 natively, and it is suggested that the Oracle database use the same character set. You will also need to create a user account for DSpace (e.g. *dspace*) and ensure that it has permissions to add and remove tables in the database. Refer to the Quick Installation for more details.
 - **NOTE:** If the database server is not on the same machine as DSpace, you must install the Oracle client to the DSpace server and point `tnsnames.ora` and `listener.ora` files to the database the Oracle server.
 - **NOTE:** DSpace uses sequences to generate unique object IDs — beware Oracle sequences, which are said to lose their values when doing a database export/import, say restoring from a backup. Be sure to run the script `etc/oracle/update-sequences.sql` after importing.
 - For people interested in switching from Postgres to Oracle, I know of no tools that would do this automatically. You will need to recreate the community, collection, and eperson structure in the Oracle system, and then use the item export and import tools to move your content over.

2.3.6 Servlet Engine (Apache Tomcat 7 or later, Jetty, Caucho Resin or equivalent)

Tomcat 7 Version

If you are using Tomcat 7, we recommend running Tomcat 7.0.30 or above. Tomcat 7.0.29 and lower versions suffer from a memory leak. As a result, those versions of tomcat require an unusual high amount of memory to run DSpace. This has been resolved as of Tomcat 7.0.30. More information can be found in [DS-1553](#)

- **Apache Tomcat 7 or later.** Tomcat can be downloaded from the following location: <http://tomcat.apache.org>.
 - Note that DSpace will need to run as the same user as Tomcat, so you might want to install and run Tomcat as a user called '*dspace*'. Set the environment variable `TOMCAT_USER` appropriately.
 - You need to ensure that Tomcat has a) enough memory to run DSpace and b) uses UTF-8 as its default file encoding for international character support. So ensure in your startup scripts (etc) that the following environment variable is set: `JAVA_OPTS="-Xmx512M -Xms64M -Dfile.encoding=UTF-8"`

- **Modifications in *[tomcat]/conf/server.xml*:** You also need to alter Tomcat's default configuration to support searching and browsing of multi-byte UTF-8 correctly. You need to add a configuration option to the `<Connector>` element in *[tomcat]/config/server.xml*: `URIEncoding="UTF-8"` e.g. if you're using the default Tomcat config, it should read:

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

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

- Tomcat 8 and above is using at least Java 1.7 for JSP compilation. However, by default, Tomcat 7 uses Java 1.6 for JSP compilation. If you want to use Java 1.7 in your .jsp files, you have to change the configuration of Tomcat 7. Edit the file called web.xml in the configuration directory of your Tomcat instance ($\{\text{CATALINA_HOME}\}/\text{conf}$ in Tomcat notation). Look for a servlet definition using the `org.apache.jasper.servlet.JSPServlet` servlet-class and add two init parameters `compilerSourceVM` and `compilerTargetVM` as you see it in the example below. Then restart Tomcat.

```

${CATALINA_BASE}/conf/web.xml

<servlet>
  <servlet-name>jsp</servlet-name>
  <servlet-class>org.apache.jasper.servlet.JspServlet</servlet-class>
  <init-param>
    <param-name>fork</param-name>
    <param-value>>false</param-value>
  </init-param>
  <init-param>
    <param-name>xpoweredBy</param-name>
    <param-value>>false</param-value>
  </init-param>
  <init-param>
    <param-name>compilerSourceVM</param-name>
    <param-value>1.7</param-value>
  </init-param>
  <init-param>
    <param-name>compilerTargetVM</param-name>
    <param-value>1.7</param-value>
  </init-param>
  <load-on-startup>3</load-on-startup>
</servlet>

```

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

2.3.7 Perl (only required for [dSPACE]/bin/dSPACE-info.pl)

2.4 Installation Instructions

2.4.1 Overview of Install Options

Two different distributions are available for DSpace, both of which require you to build the distribution using Apache Maven 3. The steps that are required to execute the build are identical. In a nutshell, the binary release build will download pre-compiled parts of DSpace, while the building the source release will compile most of DSpace's source code on your local machine.

It's important to notice that both releases will require outgoing internet connections on the machine or server where you are executing the build, because maven needs to download 3rd party dependencies that are not even included in the DSpace source release distribution.

- Binary 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 only exposes selected parts of the application for customization. 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

- Source Release (dspace-<version>-src-release.zip)
 - This method is recommended for those who wish to develop DSpace further or alter its underlying capabilities to a greater degree.
 - It contains **all** dspace code for the core dspace-api, supporting servlets, taglibraries, aspects and themes for Manakin (dspace-xmlui), and other webservice/applications.
 - Provides all the same capabilities as the binary 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-Ini*- Lightweight Network Interface source module (*deprecated as of 5.0*)
 - *dspace-oai*- OAI-PMH source module
 - *dspace-rdf*- RDF source module
 - *dspace-rest*- REST API source module
 - *dspace-services*- Common Services module
 - *dspace-sword*- SWORD (Simple Web-serve Offering Repository Deposit) deposit service source module
 - *dspace-swordv2*- SWORDv2 source module
 - *dspace-xmlui*- XML-UI (Manakin) source module
 - *dspace-xmlui-mirage2*- Mirage 2 theme for the XMLUI
 - *pom.xml*- DSpace Parent Project definition

2.4.2 Overview of DSpace Directories

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

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

1. **The installation directory**, referred to as `[dspace]`. This is the location where DSpace is installed and running. It is the location that is 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.
2. **The source directory**, referred to as `[dspace-source]`. This is the location where the DSpace release distribution has been unpacked. It usually has the name of the archive that you expanded such as `dspace-<version>-release` or `dspace-<version>-src-release`. Normally it is the directory where all of your "build" commands will be run.

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

If you ever notice that many files seems to have duplicates under `[dspace-source]/dspace/target` do not worry about it. This "target" directory will be used by Maven for the build process and you should not change any file in it unless you know exactly what you are doing.

2.4.3 Installation

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

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

```
useradd -m dspace
```

- 2. Download the latest DSpace release.** There are two version available with each release of DSpace: (*dspace-n.x-release.* and *dspace-n.x-src-release.zzz*); you only need to choose one. If you want a copy of all underlying Java source code, you should download the *dspace-n.x-src-release.xxx* Within each version, you have a choice of compressed file format. Choose the one that best fits your environment.
 - Alternatively, you may choose to check out the latest release from the [DSpace GitHub Repository](#). In this case, you'd be checking out the full Java source code. You'd also want to be sure to checkout the appropriate tag or branch. For more information on using / developing from the GitHub Repository, see: [Development with Git](#)

3. **Unpack the DSpace software.** After downloading the software, based on the compression file format, choose one of the following methods to unpack your software:

a. **Zip file.** If you downloaded *dspace-5.x-release.zip* do the following:

```
unzip dspace-5.x-release.zip
```

b. **.gz file.** If you downloaded *dspace-5.x-release.tar.gz* do the following:

```
gunzip -c dspace-5.x-release.tar.gz | tar -xf -
```

c. **.bz2 file.** If you downloaded *_dspace-5.x-release.tar.bz* do the following:

```
bunzip2 dspace-5.x-release.tar.bz | tar -xf -
```

For ease of reference, we will refer to the location of this unzipped version of the DSpace release as *[dspace-source]* in the remainder of these instructions. After unpacking the file, the user may wish to change the ownership of the *dspace-5.x-release* to the "dspace" user. (And you may need to change the group).

4. Database Setup

- Also see ["Relational Database" prerequisite notes above](#)
- *PostgreSQL:*
 - A PostgreSQL JDBC driver is configured as part of the default DSpace build. You no longer need to copy any PostgreSQL jars to get PostgreSQL installed.
 - Create a `dspace` database user. This is entirely separate from the `dspace` operating-system user created above (*you are still logged in as "root"*):

```
createuser --username=postgres --no-superuser --pwprompt dspace
```

You will be prompted (twice) for a password for the new `dspace` user. Then you'll be prompted for the password of the PostgreSQL superuser (`postgres`).

- Create a `dspace` database, owned by the `dspace` PostgreSQL user (*you are still logged in as 'root'*):

```
createdb --username=postgres --owner=dspace --encoding=UNICODE dspace
```

You will be prompted for the password of the PostgreSQL superuser (`postgres`).

- *Oracle:*
 - Setting up DSpace to use Oracle is a bit different now. You will still need to get a copy of the Oracle JDBC driver, but instead of copying it into a lib directory you will need to install it into your local Maven repository. (You'll need to download it first from this location: <http://www.oracle.com/technetwork/database/enterprise-edition/jdbc-112010-090769.html>.) Run the following command (all on one line):

```
mvn install:install-file
  -Dfile=ojdbc6.jar
  -DgroupId=com.oracle
  -DartifactId=ojdbc6
  -Dversion=11.2.0.4.0
  -Dpackaging=jar
  -DgeneratePom=true
```

- You need to compile DSpace with an Oracle driver (ojdbc6.jar) corresponding to your Oracle version - update the version in *[dspace-source]/pom.xml* E.g.:

```
<dependency>
  <groupId>com.oracle</groupId>
  <artifactId>ojdbc6</artifactId>
  <version>11.2.0.4.0</version>
</dependency>
```

- Create a database for DSpace. Make sure that the character set is one of the Unicode character sets. DSpace uses UTF-8 natively, and it is required that the Oracle database use the same character set. Create a user account for DSpace (e.g. *dspace*) and ensure that it has permissions to add and remove tables in the database.
- Uncomment and edit the Oracle database settings in *[dspace-source]/build.properties* (see below for more information on the build.properties file):

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

Where SID is the SID of your database defined in tnsnames.ora, default Oracle port is 1521


Alternatively, you can use a full SID definition, e.g.:

```
db.url = jdbc:oracle:thin:@(description=(address_list=(address=(protocol=TCP)(
host=localhost)(port=1521)))(connect_data=(service_name=DSPACE)))
```


- Later, during the Maven build step, don't forget to specify `mvn -Ddb.name=oracle package`

5. **Initial Configuration:** Edit `[dspace-source]/build.properties`. This properties file contains the basic settings necessary to actually build/install DSpace for the first time (see [build.properties Configuration](#) for more detail). In particular you'll need to set these properties -- examples or defaults are provided in the file:
- `dspace.install.dir` - must be set to the *[dspace]* (installation) directory (*On Windows be sure to use forward slashes for the directory path!* For example: "C:/dspace" is a valid path for Windows .)
 - `dspace.hostname` - fully-qualified domain name of web server.
 - `dspace.baseUrl` - complete URL of this server's DSpace home page but without any context eg. /xmlui, /oai, etc.
 - `dspace.name` - "Proper" name of your server, e.g. "My Digital Library".
 - `solr.server` - complete URL of the Solr server. DSpace makes use of [Solr](#) for indexing purposes.
 - `default.language`
 - `db.driver`
 - `db.url`
 - `db.username` - the database username used in the previous step.
 - `db.password` - the database password used 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.
 - `mail.feedback.recipient` - mailbox for feedback mail.
 - `mail.admin` - mailbox for DSpace site administrator.
 - `mail.alert.recipient` - mailbox for server errors/alerts (not essential but very useful!)

- `mail.registration.notify`- mailbox for emails when new users register (optional)

 The "build.properties" file is provided as a convenient method of setting only those configurations necessary to install/upgrade DSpace. Any settings changed in this file, will be automatically copied over to the full "dspace.cfg" file (which is held in [`dspace-source`]/`dspace/config/dspace.cfg`). Refer to the [General Configuration](#) section for a fuller explanation.

It is also worth noting that you may choose to copy/rename the "build.properties" under a different name for different environments (e.g. "development.properties", "test.properties", and "production.properties"). You can choose which properties file you want to build DSpace with by passing a "-Denv" (environment) flag to the "mvn package" command (e.g. "mvn package -Denv=test" would build using "test.properties"). See [General Configuration](#) section for more details.

 **Do not remove or comment out settings in build.properties**

When you edit the "build.properties" file (or a custom *.properties file), take care not to remove or comment out any settings. Doing so, may cause your final "dspace.cfg" file to be misconfigured with regards to that particular setting. Instead, if you wish to remove/disable a particular setting, just clear out its value. For example, if you don't want to be notified of new user registrations, ensure the "mail.registration.notify" setting has no value , e.g.
`mail.registration.notify=`

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

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

(Assuming the *dspace* UNIX username.)

7. **Build the Installation Package:** As the *dspace* UNIX user, generate the DSpace installation package.

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


Building with Oracle Database Support

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
```

Enabling and building the DSpace 5 Mirage 2 theme

Mirage 2 is a responsive theme for the XML User Interface, added as a new feature in DSpace 5. It has not yet replaced the Mirage 1 theme as the XMLUI default theme.

The Mirage 2 build requires **git** to be installed on your server. Install git before attempting the Mirage 2 build.

To enable Mirage 2, add the following to the `<themes>` section of `src/dspace/config/xmlui.xconf`, replacing the currently active theme:

```
<theme  
  
name=  
  
"Mirage 2"  
  
regex=  
  
".*"  
  
path=  
  
"Mirage2/"  
  
>
```

It is important to do this before executing the maven build.

Mirage 2 is not yet activated in the default "mvn package" build. To include it as part of the build, run:

```
mvn package -Dmirage2.on=  
true
```

The speed of this specific step of the build can be increased by installing local copies of the specific dependencies required for building Mirage 2. The [Mirage 2 developer documentation](#) provides detailed instructions for these installations. After the installation of these dependencies, you can choose to run:

```
mvn package -Dmirage2.on=
true
-Dmirage2.deps.included=
false
```

Warning: The Mirage 2 build process should NOT be run as "root". It must be run as a non-root user. For more information see: [Mirage 2 Common Build Issues](#)

Defaults to building installation package with settings from "build.properties"

Without any extra arguments, the DSpace installation package will be initialized using the settings in the `[dspace-source]/build.properties` file. However, if you want it to build using a custom properties file, you may specify the "-Denv" (environment) flag as follows:


```
mvn -Denv=test package (would build the installation package using a custom [
dspace-source]/test.properties file)
```

```
mvn -Denv=local package (would build the installation package using a custom [
dspace-source]/local.properties file)
```

See [General Configuration](#) section for more details.

8. Install DSpace: As the *dspace* UNIX user, install DSpace to `[dspace]`:

```
cd [dspace-source]/dspace/target/dspace-installer
ant fresh_install
```

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

9. Decide which DSpace Web Applications you want to install. DSpace comes with a variety of web applications (in `[dspace]/webapps`), each of which provides a different "interface" to your DSpace.

Which ones you install is up to you, but there are a few that we highly recommend (see below):

- a. "xmlui" = This is the [XML-based User Interface \(XMLUI\)](#), based on Apache Cocoon. It comes with a variety of out-of-the-box themes, including [Mirage 1](#) (the default) and [Mirage 2](#) (based on [Bootstrap](#)). *Between the "xmlui" and "jspui", you likely only need to choose one.*
- b. "jspui" = This is the [JSP-based User Interface \(JSPUI\)](#), which is based on [Bootstrap](#). *Between the "xmlui" and "jspui", you likely only need to choose one.*
- c. "solr" (*required*) = This is Apache Solr web application, which is used by the "xmlui" and "jspui" (for search & browse functionality), as well as the OAI-PMH interface. **It must be installed in support of either UI.**
- d. "oai" = This is the [DSpace OAI interface](#). It allows for Metadata and Bitstream (content-file) harvesting, supporting [OAI-PMH](#) (Protocol for Metadata Harvest) and [OAI-ORE](#) (Object Reuse and Exchange) protocols
- e. "rdf" (*new*) = This is the DSpace RDF interface supporting [Linked \(Open\) Data](#).
- f. "rest" = This is the [DSpace REST API](#)
- g. "sword" = This is the DSpace [SWORDv1 interface](#). More info on [SWORD protocol and its usage](#).
- h. "swordv2" = This is the DSpace [SWORDv2 interface](#). More info on [SWORD protocol and its usage](#).
- .
- i. "lni" (*deprecated*) = This is the DSpace [Lightweight Networking Interface \(LNI\)](#), supporting WebDAV / SOAP / RPC API. It is disabled by default as we recommend using REST or SWORD for most activities. In order to build it you must rebuild DSpace with the following flag: `mvn package -Pdspace-lni`

10. Deploy Web Applications:

Please note that in the first instance you should refer to the appropriate documentation for your Web Server of choice. The following instructions are meant as a handy guide. You have two choices or techniques for having Tomcat/Jetty/Resin serve up your web applications:

- *Technique A.* Tell your Tomcat/Jetty/Resin installation where to find your DSpace web application(s). As an example, in the directory `[tomcat]/conf/Catalina/localhost` you could add files similar to the following (but replace `[dspace]` with your installation location):

```

DEFINE A CONTEXT FOR DSpace XML User Interface: xmlui.xml

<?xml version='1.0'?>
<Context
  docBase="[dspace]/webapps/xmlui"
  reloadable="true"
  cachingAllowed="false"/>
```

DEFINE A CONTEXT PATH FOR DSpace JSP User Interface: jspui.xml

```
<?xml version='1.0'?>
<Context
  docBase="[dspace]/webapps/jspui"
  reloadable="true"
  cachingAllowed="false"/>
```

DEFINE A CONTEXT PATH FOR DSpace Solr index: solr.xml

```
<?xml version='1.0'?>
<Context
  docBase="[dspace]/webapps/solr"
  reloadable="true"
  cachingAllowed="false"/>
```

DEFINE A CONTEXT PATH FOR DSpace OAI User Interface: oai.xml

```
<?xml version='1.0'?>
<Context
  docBase="[dspace]/webapps/oai"
  reloadable="true"
  cachingAllowed="false"/>
```

DEFINE ADDITIONAL CONTEXT PATHS FOR OTHER DSPACE WEB APPLICATIONS (REST, SWORD, RDF, LNI , etc.): \[app].xml

```
<?xml version='1.0'?>
<!-- CHANGE THE VALUE OF "[app]" FOR EACH APPLICATION YOU WISH TO ADD -->
<Context
  docBase="[dspace]/webapps/[app]"
  reloadable="true"
  cachingAllowed="false"/>
```

The name of the file (not including the suffix ".xml") will be the name of the context, so for example `xmlui.xml` defines the context at `http://host:8080/xmlui`. To define the *root context* (`http://host:8080/`), name that context's file `ROOT.xml`.

Tomcat Context Settings in Production

The above Tomcat Context Settings show adding the following to each <Context> element: `reloadable="true" cachingAllowed="false"`

These settings are extremely useful to have when you are first getting started with DSpace, as they let you tweak the DSpace XMLUI (XSLTs or CSS) or JSPUI (JSPs) and see your changes get automatically reloaded by Tomcat (without having to restart Tomcat). However, it is worth noting that the [Apache Tomcat](#) documentation recommends Production sites leave the default values in place (`reloadable="false" cachingAllowed="true"`), as allowing Tomcat to automatically reload all changes may result in "significant runtime overhead".

It is entirely up to you whether to keep these Tomcat settings in place. We just recommend beginning with them, so that you can more easily customize your site without having to require a Tomcat restart. Smaller DSpace sites may not notice any performance issues with keeping these settings in place in Production. Larger DSpace sites may wish to ensure that Tomcat performance is more streamlined.

- *Technique B.* Simple and complete. You copy only (or all) of the DSpace Web application(s) you wish to use from the [dspace]/webapps directory to the appropriate directory in your Tomcat/Jetty/Resin installation. For example:

```
cp -R [dspace]/webapps/* [tomcat]/webapps* (This will copy all the web applications to Tomcat).
```

```
cp -R [dspace]/webapps/jspui [tomcat]/webapps* (This will copy only the jspui web application to Tomcat.)
```

11. **Initialize the DSpace Database (optional):** While this step is optional (as the DSpace database will auto-initialize itself on first startup), it's always good to verify one last time that your database connection is working properly. To initialize the database run: (for more information on "database migrate" see [Database Utilities](#))

```
[dspace]/bin/dspace database migrate
```

12. **Administrator Account:** Create an initial administrator account from the command line:

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

13. **Initial Startup!** Now the moment of truth! Start up (or restart) Tomcat/Jetty/Resin. Visit the base URL(s) of your server, depending on which DSpace web applications you want to use. You should see the DSpace home page. Congratulations! Base URLs of DSpace Web Applications:
- a.
 - *JSP User Interface* - (e.g.) `http://dspace.myu.edu:8080/jspui`
 - *XML User Interface* (aka. Manakin) - (e.g.) `http://dspace.myu.edu:8080/xmlui`
 - *OAI-PMH Interface* - (e.g.) `http://dspace.myu.edu:8080/oai/request?verb=Identify` (Should return an XML-based response)

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

2.5 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.

2.5.1 'cron' jobs / scheduled tasks

A few DSpace features **require** that a script is run regularly (via cron, or similar):

- the [e-mail subscription feature](#) that alerts users of new items being deposited;
- the ['media filter' tool](#), that generates thumbnails of images and extracts the full-text of documents for indexing;
- the ['checksum checker'](#) that tests the bitstreams in your repository for corruption;
- the [sitemap generator](#), which enhances the ability of major search engines to index your content and make it findable;
- the [curation system queueing feature](#), which allows administrators to "queue" tasks (to run at a later time) from the Admin UI;
- and [Discovery](#) (search & browse), [OAI-PMH](#) and [Usage Statistics](#) all receive performance benefits from regular re-optimization

For much more information on recommended scheduled tasks, please see [Scheduled Tasks via Cron](#).

2.5.2 Multilingual Installation

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

- *default.locale*, e.g. `default.locale = en`
- *webui.supported_locales*, e.g. `webui.supported.locales = en, de`

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

According to the languages you wish to support, you have to make sure that all the i18n related files are available. See the [Configuring Multilingual Support](#) section for the JSPUI or the [Multilingual Support](#) for XMLUI in the configuration documentation.

2.5.3 DSpace over HTTPS

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

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

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

Enabling the HTTPS support in Tomcat 7.0

Loosely based on <http://tomcat.apache.org/tomcat-7.0-doc/ssl-howto.html>.

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

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

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

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

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

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

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

```
<Connector port="8443"
  URIEncoding="UTF-8"
  maxThreads="150" minSpareThreads="25"
  enableLookups="false"
  disableUploadTimeout="true"
  acceptCount="100"
  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" truststorePass="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"
  enableLookups="false"
  redirectPort="8443"
  acceptCount="100" />
```


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

```

You should be prompted for a password to protect the private key.

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.

- b. 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

```

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

Using SSL on Apache HTTPD with mod_jk

⊖ When using Apache 2.4.2 (and lower) in front of a DSpace webapp deployed in Tomcat, `mod_proxy_ajp` and possibly `mod_proxy_http` breaks the connection to the back end (Tomcat) prematurely leading to response mixups. This is reported as bug CVE-2012-3502 (<http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2012-3502>) of Apache and fixed in Apache 2.4.3 (see http://www.apache.org/dist/httpd/CHANGES_2.4). The 2.2.x branch hasn't shown this problem only the 2.4.x branch has.

If you choose [Apache HTTPD](#) as your primary HTTP server, you can have it forward requests to the [Tomcat servlet container](#) via [Apache Jakarta Tomcat Connector](#). 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](#) for information about using `mod_ssl`.

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](#) 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.

2.5.4 The Handle Server

First a few facts to clear up some common misconceptions:

- You don't **have** to use CNRI's Handle system. At the moment, you need to change the code a little to use something else (e.g PURLs) but that should change soon.

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

If you want to use the Handle system, you'll need to set up a Handle server. This is included with DSpace. Note that this is not required in order to evaluate DSpace; you only need one if you are running a production service. You'll need to obtain a Handle prefix from [the central CNRI Handle site](#).

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 send and receive TCP on port 2641.

- To configure your DSpace installation to run the handle server, run the following command:

```
[dSPACE]/bin/dSPACE make-handle-config [dSPACE]/handle-server
```

Ensure that *[dSPACE]/handle-server* matches whatever you have in *dSPACE.cfg* for the *handle.dir* property.

- If you are using Windows, the proper command is:

```
[dSPACE]/bin/dSPACE dsrun net.handle.server.SimpleSetup [dSPACE]/handle-server
```

Ensure 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.

- 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.
- When CNRI has sent you your naming authority prefix, you will need to edit the *config.dct* file. The file will be found in *[dSPACE]/handle-server*. Look for "300:0.NA/YOUR_NAMING_AUTHORITY". Replace *YOUR_NAMING_AUTHORITY* with the assigned naming authority prefix sent to you.

5. Now start your handle server (as the dspace user):

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

- a. If you are using Windows, the proper command is (please replace "[dspace]\handle-server" with the full path of the handle-server directory):

```
[dspace]/bin/dspace dsrun net.handle.server.Main [dspace]/handle-server
```

Ensure that *[dspace]/handle-server* matches whatever you have in *dspace.cfg* for the *handle.dir* property.

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

Updating Existing Handle Prefixes

If you need to update the handle prefix on items created before the CNRI registration process you can run the *[dspace]/bin/dspace 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/dspace update-handle-prefix 123456789 1303
```

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

2.5.5 Google and HTML sitemaps

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

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

To generate the sitemaps, you need to run *[dspace]/bin/dspace 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 HTML based sitemap
- etc...

When running `[dspace]/bin/dspace 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 at 6:00 am local time each day
0 6 * * * [dspace]/bin/dspace generate-sitemaps
```

 More information on why we **highly recommend** enabling sitemaps can be found at [Search Engine Optimization \(SEO\)](#).

2.5.6 Statistics

DSpace uses the Apache Solr application underlying the statistics. There is no need to download any separate software. All the necessary software is included. To understand all of the configuration property keys, the user should refer to [DSpace Statistic Configuration](#) for detailed information.

2.6 Windows Installation

Essentially installing on Windows is the same as installing on Unix so please refer back to the main [Installation Instructions](#) section.

- Download the DSpace source from [SourceForge](#) and unzip it ([WinZip](#) will do this)
- If you install PostgreSQL, it's recommended to select to install the pgAdmin III tool. It provides a nice User Interface for interacting with PostgreSQL databases.
- For all path separators use forward slashes (e.g. "/"). For example: "C:/dspace" is a valid Windows path. But, be warned that "C:\dspace" IS INVALID and will cause errors.

2.7 Checking Your Installation

The administrator needs to check the installation to make sure all components are working. Here is list of checks to be performed. In brackets after each item, it the associated component or components that might be the issue needing resolution.

- System is up and running. *User can see the DSpace home page. [Tomcat/Jetty, firewall, IP assignment, DNS]*
- Database is running and working correctly. *Attempt to create a user, community or collection. [PostgreSQL, Oracle]*Run the database connection testing command to see if other issues are being reported: `[dspace]/bin/dspace database test`
- Email subsystem is running. The user can issue this command to test the email system: `[dspace]/bin/dspace test-email`It attempts to send a test email to the email address that is set in `dspace.cfg` (`mail.admin`). If it fails, you will get messages informing you as to why, referring you to the DSpace documentation.

2.8 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](#) 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.

2.9 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.

2.9.1 Common Installation Issues

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

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

it usually means you haven't yet added the relevant configuration parameter to your PostgreSQL configuration (see above), or perhaps you haven't restarted PostgreSQL after making the change. Also, make sure that the `db.username` and `db.password` properties are correctly set in `[dspace]/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]/lib`. See above.

- **GeoLiteCity Database file fails to download or install, when you run `ant fresh_install`:** There are two common errors that may occur:
 - If your error looks like this:

```
[get] Error getting http://geolite.maxmind.com/download/geoip/database/
GeoLiteCity.dat.gz to /usr/local/dspace/config/GeoLiteCity.dat.gz
BUILD FAILED
/dspace-release/dspace/target/dspace-installer/build.xml:931: java.net.ConnectException
: Connection timed out
```

it means that you likely either (a) don't have an internet connection to download the necessary GeoLite Database file (used for DSpace Statistics), or (b) the GeoLite Database file's URL is no longer valid.

- Another common message looks like this:

```
[echo] WARNING : FAILED TO DOWNLOAD GEOLITE DATABASE FILE
[echo]           (Used for DSpace Solr Usage Statistics)
```

Again, this means the GeoLite Database file cannot be downloaded or is unavailable for some reason. You should be able to resolve this issue by following the "[Manually Installing/Updating GeoLite Database File](#)" instructions.

2.9.2 General DSpace Issues

- **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 running: `ps -ef | grep java` and look for Tomcat's Java processes. If they stay around after running Tomcat's `shutdown.sh` script, trying running `kill` on them (or `kill -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 running: `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 running `kill` on the process, and stopping and restarting Tomcat.

3 Upgrading DSpace

 These instructions are valid for any of the following upgrade paths:

- Upgrading ANY prior version (1.x.x, 3.x or 4.x) of DSpace to DSpace 5.x

For more information about specific fixes released in each 5.x version, please refer to the appropriate release notes:


- [DSpace Release 5.1 Status](#)
- [DSpace Release 5.0 Status](#)

For more information about new features or major changes in previous releases of DSpace, please refer to following:

- [Past Releases](#) - Provides links to release notes for all prior releases of DSpace
- [Version History](#) - Provides detailed listing of all changes in all prior releases of DSpace

Test Your Upgrade Process

In order to minimize downtime, it is always recommended to first perform a DSpace upgrade using a Development or Test server. You should note any problems you may have encountered (and also how to resolve them) before attempting to upgrade your Production server. It also gives you a chance to "practice" at the upgrade. Practice makes perfect, and minimizes problems and downtime. Additionally, if you are using a version control system, such as subversion or git, to manage your locally developed features or modifications, then you can do all of your upgrades in your local version control system on your Development server and commit the changes. That way your Production server can just checkout your well tested and upgraded code.

 In the notes below [dspace] refers to the install directory for your existing DSpace installation, and [dspace-source] to the source directory for DSpace 5.x. Whenever you see these path references, be sure to replace them with the actual path names on your local system.

- 1 [Backup your DSpace](#)
- 2 [Update Prerequisite Software \(as necessary\)](#)
- 3 [Upgrade Steps](#)

- 4 [Troubleshooting Upgrade Issues](#)
 - 4.1 [Manually Upgrading Solr Indexes](#)
 - 4.2 ["Property was circularly defined" errors](#)

3.1 Backup your DSpace

Before you start your upgrade, it is strongly recommended that you create a backup of your DSpace instance. Backups are easy to recover from; a botched install/upgrade is very difficult if not impossible to recover from. The DSpace specific things to backup are: configs, source code modifications, database, and assetstore. On your server that runs DSpace, you might additionally consider checking on your cron/scheduled tasks, servlet container, and database.

Make a complete backup of your system, including:

- Database: Make a snapshot/dump of the database. For the PostgreSQL database use Postgres' `pg_dump` command. For example:

```
pg_dump -U [database-user] -f [backup-file-location] [database-name]
```

- Assetstore: Backup the directory (`[dspace]/assetstore` by default, and any other assetstores configured in the `[dspace]/config/dspace.cfg` "assetstore.dir" and "assetstore.dir.#" settings)
- Configuration: Backup the entire directory content of `[dspace]/config`.
- Customizations: If you have custom code, such as themes, modifications, or custom scripts, you will want to back them up to a safe location.

3.2 Update Prerequisite Software (as necessary)

DSpace 5.x requires the following versions of prerequisite software:

- Java 7 (Oracle or OpenJDK)
- Apache Maven 3.0.5 or above
- Apache Ant 1.8 or above
- Database
 - PostgreSQL 9.0 or above, OR
 - Oracle 10g or above
- Tomcat 7 or above

Refer to the [Prerequisite Software section of "Installing DSpace"](#) for more details around configuring and installing these prerequisites.

3.3 Upgrade Steps

- 1. Download DSpace 5.x:** Either download DSpace 5.x from DSpace.org or check it out directly from the [Github repository](#).
 - a. NOTE: If you downloaded DSpace do not unpack it on top of your existing installation. Refer to [Installation Instructions, Step 3](#) for unpacking directives.
- 2. Merge any User Interface customizations or other customizations (if needed or desired).** If you have made any local customizations to your DSpace installation they *may* need to be migrated over to the new DSpace.
 - a. NOTE: If you are upgrading across many versions of DSpace at once (e.g. from 1.x.x to 5.x), you may find it easier to *first* upgrade DSpace, and *then* attempt to migrate over your various customizations. Because each major version of DSpace tends to add new configurations and features to the User Interface, older customizations may require more work to "migrate" to the latest version of DSpace. In some situations, it may even be easier to "start fresh", and just re-customize the brand new User Interface with your local color scheme, header/footer, etc.
 - b. Customizations are typically housed in one of the following places:
 - i. JSPUI modifications: `[dspace-source]/dspace/modules/jspui/src/main/webapp/`
 - ii. XMLUI modifications: `[dspace-source]/dspace/modules/xmlui/src/main/webapp/`
 - iii. Config modifications: `[dspace]/config`
- 3. Edit the build.properties file (if needed)** (`[dspace-source]/build.properties`). Any settings changed in this `build.properties` file are automatically copied over to the final `dspace.cfg` file during the "Build DSpace" process (in the next step). For more information on the `build.properties` file, see "[The build.properties Configuration Properties File](#)" section of the [Configuration Reference](#) documentation.
- 4. Build DSpace.** Run the following commands to compile DSpace :

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

The above command will re-compile the DSpace source code and build its "installer". You will find the result in `[dspace-source]/dspace/target/dspace-installer`

Defaults to PostgreSQL settings

Without any extra arguments, the DSpace installation package is initialized for PostgreSQL. If you use Oracle instead, you should build the DSpace installation package as follows:

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

Enabling and building the DSpace 5 Mirage 2 theme

Mirage 2 is a responsive theme for the XML User Interface, added as a new feature in DSpace 5. It has not yet replaced the Mirage 1 theme as the XMLUI default theme.

To enable Mirage 2, add the following to the `<themes>` section of `src/dspace/config/xmlui.xconf`, replacing the currently active theme:

```
<theme
name=
"Mirage 2"
regex=
".*"
path=
"Mirage2/"
/>
```

It is important to do this before executing the maven build.

Mirage 2 is not yet activated in the default "mvn package" build. To include it as part of the build, run:

```
mvn -U clean package -Dmirage2.on=
true
```

The speed of this specific step of the build can be increased by installing local copies of the specific dependencies required for building Mirage 2. The [Mirage 2 developer documentation](#) provides detailed instructions for these installations. After the installation of these dependencies, you can choose to run:

```
mvn -U clean package -Dmirage2.on=
true
-Dmirage2.deps.included=
false
```

Warning: The Mirage 2 build process should NOT be run as "root". It must be run as a non-root user. For more information see: [Mirage 2 Common Build Issues](#)

5. **Stop Tomcat (or servlet container).** Take down your servlet container.
 - a. For Tomcat, use the `$CATALINA_HOME/shutdown.sh` script. (Many Unix-based installations will have a startup/shutdown script in the `/etc/init.d` or `/etc/rc.d` directories.)
6. **Update DSpace Installation.** Update the DSpace installation directory with the new code and libraries. Issue the following commands:

```
cd [dspace-source]/dspace/target/dspace-installer
ant update
```

The above command will also automatically upgrade all your existing Solr indexes (e.g. for Discovery, Statistics, OAI-PMH) to the latest version. For large instances, this may take some time. But, it is important to ensure that your indexes are usable by the latest version of DSpace.

- a. If the Solr index upgrade fails, you may need to [Manually Upgrade your Solr Indexes](#). See the "Troubleshooting Upgrade Issues" section below.
7. **Update your DSpace Configurations.** You should review your configuration for new and changed configurations in DSpace 5.x.
 - a. In the specific case of `dspace.cfg` it is recommended to start with a fresh copy of the file from the *new* version and copy your site-specific settings from the old file. Read the new file carefully to see if you need (or want) other alterations.
 - b. **Please notice** that as of DSpace 4, the default search and browse support has changed from the old Lucene/DBMS-based method to [Discovery](#).
 - c. It is recommended to review all configuration changes that exist in the config directory, and its subdirectories. It is helpful to compare your current configs against a clean checkout of your current version to see what you have customized. You might then also want to compare your current configs with the configs of the version you are upgrading to. A tool that compares files in directories such as Meld or DiffMerge is useful for this purpose.
 - i. Upgrading from 4.x to 5.x, notice that file `config/crosswalks/google-metadata.properties` uses `google.citation_author` instead of `google.citation_authors`

8. **Decide which DSpace Web Applications you want to install.** DSpace comes with a variety of web applications (in `[dspace]/webapps`), each of which provides a different "interface" to your DSpace. Which ones you install is up to you, but there are a few that we highly recommend (see below):
- "xmlui" = This is the XML-based User Interface, based on Apache Cocoon. It comes with a variety of out-of-the-box themes, including [Mirage 1](#) (the default) and [Mirage 2](#) (based on [Bootstrap](#)). *Between the "xmlui" and "jspui", you likely only need to choose one.*
 - "jspui" = This is the JSPUI-based User Interface, which is based on [Bootstrap](#). *Between the "xmlui" and "jspui", you likely only need to choose one.*
 - "solr" (*required*) = This is Apache Solr web application, which is used by the "xmlui" and "jspui" (for search & browse functionality), as well as the OAI-PMH interface. **It must be installed in support of either UI.**
 - "oai" = This is the [DSpace OAI interface](#). It allows for metadata and bitstream (content-file) harvesting, supporting [OAI-PMH](#) (Protocol for Metadata Harvest) and [OAI-ORE](#) (Object Reuse and Exchange) protocols
 - "rest" = This is the [DSpace REST API](#)
 - "sword" = This is the [DSpace SWORDv1 interface](#). More info on [SWORD protocol and its usage](#).
 - "swordv2" = This is the [DSpace SWORDv2 interface](#). More info on [SWORD protocol and its usage](#).
 - .
 - "rdf" (*new*) = This is the DSpace RDF interface supporting [Linked \(Open\) Data](#).
 - "lni" (*deprecated*) = This is the DSpace [Lightweight Networking Interface](#), supporting WebDAV / SOAP / RPC API. It is disabled by default as we recommend using REST or SWORD for most activities. In order to build it you must rebuild DSpace with the following flag: `mvn package -Pdspace-lni`
9. **Deploy DSpace Web Applications.** If necessary, copy the web applications from your `[dspace]/webapps` directory to the subdirectory of your servlet container (e.g. Tomcat):

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

See [the installation guide](#) for full details.

10. **Upgrade your database** (*optional, but recommended for major upgrades*). As of DSpace 5, the DSpace code will automatically upgrade your database (from any prior version of DSpace). By default, this database upgrade occurs automatically when you restart Tomcat (or your servlet container). However, if you have a large repository or are upgrading across multiple versions of DSpace at once, you may wish to manually perform the upgrade (as it could take some time, anywhere from 5-15 minutes for large sites)

- a. First, you can optionally verify whether DSpace correctly detects the version of your DSpace database. It is **very important** that the DSpace version is detected correctly before you attempt the migration:

```
[dspace]/bin/dspace database info
# Look for a line at the bottom that says something like:
# "Your database looks to be compatible with DSpace version ___"
```

- b. Then, you can upgrade your DSpace database to the latest version of DSpace. (NOTE: check the DSpace log, [dspace]/log/dspace.log.[date], for any output from this command)

```
[dspace]/bin/dspace database migrate
```

- c. More information on this new "database" command can be found in [Database Utilities](#) documentation.

11. **Remove deprecated Database Browse Tables (optional, but recommended)**

- a. By default, DSpace now uses [Discovery](#) (backed by Solr) for its Search and Browse engine. Discovery offers additional features like filtered (or faceted) searching, and "access aware" searching which was not offered by the [Legacy Search and Browse](#) system. We highly recommend using Discovery for a better Search and Browse experience. In the future, the Legacy Search and Browse system likely will be removed.
- b. As long as you plan to use the default settings in DSpace (with Discovery enabled), you can safely remove any old Legacy browse tables (named "bi_*", where "bi" = browse index). To do so, simply run:

```
[dspace]/bin/dspace index-db-browse -f -d
```


12. **Restart Tomcat (servlet container).** Now restart your servlet container (Tomcat/Jetty/Resin) and test out the upgrade.
 - a. **Upgrade of database:** If you didn't manually upgrade your database in the previous step, then your database will be automatically upgraded to the latest version. This may take some time (seconds to minutes), depending on the size of your repository, etc. Check the DSpace log (`[dspace] / log / dspace . log . [date]`) for information on its status.
 - b. **Reindexing of all content for search/browse:** If your database was just upgraded (either manually or automatically), all the content in your DSpace will be automatically re-indexed for searching/browsing. As the process can take some time (minutes to hours, depending on the size of your repository), it is performed in the background; meanwhile, DSpace can be used as the index is gradually filled. *But, keep in mind that not all content will be visible until the indexing process is completed.* Again, check the DSpace log (`[dspace] / log / dspace . log . [date]`) for information on its status.
13. **Check your cron / Task Scheduler jobs.** In recent versions of DSpace, some of the scripts names have changed.
 - a. Check the [Scheduled Tasks via Cron](#) documentation for details. Especially pay attention to the Solr Index optimization commands, which ideally should be run regularly (as noted in the previous step).

3.4 Troubleshooting Upgrade Issues

3.4.1 Manually Upgrading Solr Indexes

If you run into issues with the auto-upgrade of your Solr search/browse indexes (during the final part of the `ant update step`), then you may need to manually upgrade your Solr indexes. Depending on the type of failure, there are a few possible fixes.

1. If the "ant update" process failed to download the `lucene-core-3.5.0.jar`, in order to upgrade a DSpace 1.6.x, 1.7.x or 1.8.x index.
 - a. You can manually download the `lucene-core-3.5.0.jar` from <http://search.maven.org/remotecontent?filepath=org/apache/lucene/lucene-core/3.5.0/lucene-core-3.5.0.jar>
 - b. Place the `lucene-core-3.5.0.jar` in your `[dspace-source] / dspace / target / dspace-installer /` directory (i.e. the directory where you ran "ant update" from)
 - c. Re-Run "ant update". This time, it should find the `lucene-core-3.5.0.jar` locally and re-attempt the upgrade of your Solr indexes.
2. If some other error occurred, you may need to **manually** upgrade your Solr indexes.

- a. **Upgrading from DSpace 1.6.x, 1.7.x or 1.8.x:** In DSpace 1.x versions, we used an older version of Solr which is no longer compatible with the current version of Solr.
- If you are using an older version of DSpace, you will see errors similar to this one until you manually upgrade your index:

```
Caused by: org.apache.lucene.index.IndexFormatTooOldException: Format version is not supported (resource: segment _386q in resource ChecksumIndexInput(MMapIndexInput(path="/space/dspace/solr/statistics/data/index/segments_37m6"))): 2.x. This version of Lucene only supports indexes created with release 3.0 and later.
```

- Manually upgrading your Solr index involves temporarily downloading an older version of Lucene (on which Solr is based), and calling its IndexUpgrader script, e.g.

```
# Download Lucene 3.5.0, which can upgrade older Solr/Lucene indexes
wget "http://search.maven.org/remotecontent?filepath=org/apache/lucene/lucene-core/3.5.0/lucene-core-3.5.0.jar" -O lucene-core-3.5.0.jar
# Then, actually upgrade the indexes by loading the lucene-core-3.5.0.jar and calling IndexUpgrader
# Upgrade the Usage Statistics index. Run this if you have Solr Usage Statistics enabled in your UI.
java -cp lucene-core-3.5.0.jar org.apache.lucene.index.IndexUpgrader [dSPACE]/solr/statistics/data/index/

# Upgrade the OAI-PMH indexes. Run this if you use the "oai" webapp.
java -cp lucene-core-3.5.0.jar org.apache.lucene.index.IndexUpgrader [dSPACE]/solr/oai/data/index/

# NOTE: You do not need to upgrade the Discovery Search and Browse indexes as they will be automatically rebuilt on upgrade (See previous upgrade step)
```

- At this point in time, your older indexes will now be compatible with Solr / Lucene 3.5. At this point they are readable by the latest version of Solr.
- However, as a final step, you should still **optimize** each of these indexes using the commands detailed in the "Upgrading from DSpace 3.x or Above" step below

- b. **Upgrading from DSpace 3.x or above:** DSpace provides optimization commands for all Solr indexes. Which ones you need to run depend on which features you are using in DSpace.

```
# First, ensure your Tomcat is started up. All of the below commands will call Solr
directly, which requires Tomcat to be running.

# Optimize Usage Statistics (based on Solr). Run this if you have Usage Statistics
enabled in your UI.
[dspace]/bin/dspace stat-util -o

# Optimize OAI-PMH indexes (based on Solr). Run this if you use the "oai" webapp.
[dspace]/bin/dspace oai import -o

# NOTE: You should not need to optimize the Discovery Search and Browse indexes, as
they will be automatically rebuilt on upgrade (See previous upgrade step)
# However, you still may wish to schedule optimizing of Discovery Search & Browse (via
cron or similar)
# [dspace]/bin/dspace index-discovery -o
```

3.4.2 "Property was circularly defined" errors

If, after running `ant update`, you see an error like this:

```
[dspace-src]/dspace/target/dspace-installer/build.xml:88: Property ____ was circularly defined.
```

This usually means that you are attempting to build the new version of DSpace using an outdated `build.properties` file. Specifically, the property which is reported as being "circularly defined" is likely missing from your local `[dspace-source]/build.properties` file. To resolve this issue, simply:

1. Check your local `build.properties` file to ensure it is up to date, specifically looking for properties which are missing and adding them in.
2. Rebuild DSpace (e.g. `mvn -U clean package`). See [Upgrade Steps](#) (step #4) above.
3. Continue the rest of the [upgrade process](#) (again from step #4) above

4 Using DSpace

This page offers access to all aspects of the documentation relevant to using DSpace after it has been properly installed or upgraded. These pages assume that DSpace is functioning properly. Please refer to the section on [System Administration](#) if you are looking for information on diagnosing DSpace issues and measures you can take to restore your DSpace to a state in which it functions properly.

4.1 Authentication and Authorization

- [Authentication Plugins](#)
- [Embargo](#)
- [Managing User Accounts](#)
- [Request a Copy](#)

4.1.1 Authentication Plugins

- 1 [Stackable Authentication Method\(s\)](#)
 - 1.1 [Authentication by Password](#)
 - 1.1.1 [Enabling Authentication by Password](#)
 - 1.1.2 [Configuring Authentication by Password](#)
 - 1.2 [Shibboleth Authentication](#)
 - 1.2.1 [Enabling Shibboleth Authentication](#)
 - 1.2.2 [Configuring Shibboleth Authentication](#)
 - 1.2.2.1 [Apache "mod_shib" Configuration \(required\)](#)
 - 1.2.2.2 [DSpace Shibboleth Configuration Options](#)
 - 1.3 [LDAP Authentication](#)
 - 1.3.1 [Enabling LDAP Authentication](#)
 - 1.3.2 [Configuring LDAP Authentication](#)
 - 1.3.3 [Enabling Hierarchical LDAP Authentication](#)
 - 1.3.4 [Configuring Hierarchical LDAP Authentication](#)
 - 1.4 [IP Authentication](#)
 - 1.4.1 [Enabling IP Authentication](#)
 - 1.4.2 [Configuring IP Authentication](#)
 - 1.5 [X.509 Certificate Authentication](#)
 - 1.5.1 [Enabling X.509 Certificate Authentication](#)
 - 1.5.2 [Configuring X.509 Certificate Authentication](#)
 - 1.6 [Example of a Custom Authentication Method](#)

Stackable Authentication Method(s)

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

Configuration File:	<code>[dspace]/config/modules/authentication.cfg</code>
Property:	<code>plugin.sequence.org.dspace.authenticate.AuthenticationMethod</code>
Example Value:	<pre>plugin.sequence.org.dspace.authenticate.AuthenticationMethod = \ org.dspace.authenticate.PasswordAuthentication</pre>

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.

Existing Authentication Methods include

- [Authentication by Password](#) (class: `org.dspace.authenticate.PasswordAuthentication`) (DEFAULT)
- [Shibboleth Authentication](#) (class: `org.dspace.authenticate.ShibAuthentication`)
- [LDAP Authentication](#) (class: `org.dspace.authenticate.LDAPAuthentication`)
- [IP Address based Authentication](#) (class: `org.dspace.authenticate.IPAuthentication`)
- [X.509 Certificate Authentication](#) (class: `org.dspace.authenticate.X509Authentication`)

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

Enabling Authentication by Password

By default, this authentication method is enabled in DSpace.

However, to enable Authentication by Password, you must ensure the `org.dspace.authenticate.PasswordAuthentication` class is listed as one of the `AuthenticationMethods` in the following configuration:

Configuration File:	<code>[dspace]/config/modules/authentication.cfg</code>
Property:	<code>plugin.sequence.org.dspace.authenticate.AuthenticationMethod</code>
Example Value:	<pre>plugin.sequence.org.dspace.authenticate.AuthenticationMethod = \ org.dspace.authenticate.PasswordAuthentication</pre>

Configuring Authentication by Password

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

- Use of inbuilt e-mail address/password-based log-in. This is achieved by forwarding a request that is attempting an action requiring authorization to the password log-in servlet, `/password-login`. The password log-in servlet (`org.dspace.app.webui.servlet.PasswordServlet`) contains code that will resume the original request if authentication is successful, as per step 3. described above.
- Users can register themselves (i.e. add themselves as e-people without needing approval from the administrators), and can set their own passwords when they do this
- Users are not members of any special (dynamic) e-person groups

- You can restrict the domains from which new users are able to register. To enable this feature, uncomment the following line from `dspace.cfg`: `authentication.password.domain.valid = example.com` Example options might be `'@example.com'` to restrict registration to users with addresses ending in `@example.com`, or `'@example.com, .ac.uk'` to restrict registration to users with addresses ending in `@example.com` or with addresses in the `.ac.uk` domain.

A full list of all available Password Authentication Configurations:

Configuration File:	<code>[dspace]/config/modules/authentication-password.cfg</code>
Property:	<code>domain.valid</code>
Example Value:	<code>domain.value = @mit.edu, .ac.uk</code>
Informational Note:	This option allows you to limit self-registration to email addresses ending in a particular domain value. The above example would limit self-registration to individuals with " <code>@mit.edu</code> " email addresses and all <code>".ac.uk"</code> email addresses.
Property:	<code>login.specialgroup</code>
Example Value:	<code>login.specialgroup = My DSpace Group</code>
Informational Note:	This option allows you to automatically add all password authenticated users to a specific DSpace Group (the group must exist in DSpace) for the remainder of their logged in session.
Property:	<code>digestAlgorithm</code>
Example Value:	<code>digestAlgorithm = SHA-512</code>
Informational Note:	This option specifies the hashing algorithm to be used in converting plain-text passwords to more secure password digests. The example value is the default. You may select any digest algorithm available through <code>java.security.MessageDigest</code> on your system. At least MD2, MD5, SHA-1, SHA-256, SHA-384, and SHA-512 should be available, but you may have installed others. Most sites will not need to adjust this.

Shibboleth Authentication

Enabling Shibboleth Authentication

To enable Shibboleth Authentication, you must ensure the `org.dspace.authenticate.ShibAuthentication` class is listed as one of the `AuthenticationMethods` in the following configuration:

Configuration File:	<code>[dspace]/config/modules/authentication.cfg</code>
Property:	<code>plugin.sequence.org.dspace.authenticate.AuthenticationMethod</code>
Example Value:	<pre>plugin.sequence.org.dspace.authenticate.AuthenticationMethod = \ org.dspace.authenticate.ShibAuthentication</pre>

Configuring Shibboleth Authentication

Shibboleth is a distributed authentication system for securely authenticating users and passing attributes about the user from one or more identity providers. In the Shibboleth terminology DSpace is a Service Provider which receives authentication information and then based upon that provides a service to the user. To use Shibboleth, DSpace *requires* that you use Apache installed with the `mod_shib` module acting as a proxy for all HTTP requests for your servlet container (typically Tomcat). DSpace will receive authentication information from the `mod_shib` module through HTTP headers.

Before DSpace will work with Shibboleth, you **must** have the following:

1. An Apache web server with the "mod_shib" module installed. As mentioned, this `mod_shib` module acts as a proxy for all HTTP requests for your servlet container (typically Tomcat). Any requests to DSpace that require authentication via Shibboleth should be redirected to 'shibd' (the shibboleth daemon) by this "mod_shib" module. Details on installing/configuring `mod_shib` in Apache are available at: <https://wiki.shibboleth.net/confluence/display/SHIB2/NativeSPApacheConfig> We also have a sample Apache + `mod_shib` configuration provided below.
2. An external Shibboleth Idp (Identity Provider). Using `mod_shib`, DSpace will only act as a Shibboleth SP (Service Provider). The actual Shibboleth Authentication & Identity information must be provided by an external IdP. If you are using Shibboleth at your institution already, then there already should be a Shibboleth IdP available. More information about Shibboleth IdPs versus SPs is available at: <https://wiki.shibboleth.net/confluence/display/SHIB2/UnderstandingShibboleth>

For more information on installing and configuring a Shibboleth Service Provider see: <https://wiki.shibboleth.net/confluence/display/SHIB2/Installation>

Note about Shibboleth Active vs Lazy Sessions:

When configuring your Shibboleth Service Provider there are two Shibboleth paradigms you may use: Active or Lazy Sessions. Active sessions is where the `mod_shib` module is configured to protect an entire URL space. No one will be able to access that URL without first authenticating with Shibboleth. Using this method you will need to configure shibboleth to protect the URL: `"/shibboleth-login"`. The alternative, Lazy Session does not protect any specific URL. Instead Apache will allow access to any URL, and when the application wants to it may initiate an authenticated session.

The Lazy Session method is preferable for most DSpace installations, as you usually want to provide public access to (most) DSpace content, while restricting access to only particular areas (e.g. administration UI/tools, private Items, etc.). When Active Sessions are enabled your *entire* DSpace site will be access restricted. In other words, when using Active Sessions, Shibboleth will require everyone to first authenticate before they can access any part of your repository (which essentially results in a "dark archive", as anonymous access will not be allowed).

Apache "mod_shib" Configuration (required)

As mentioned above, you must have Apache with the "mod_shib" module installed in order for DSpace to be able to act as a Shibboleth Service Provider (SP). The mod_shib module acts as a proxy for all HTTP requests for your servlet container (typically Tomcat). Any requests to DSpace that require authentication via Shibboleth should be redirected to 'shibd' (the shibboleth daemon) by this "mod_shib" module. Details on installing/configuring mod_shib in Apache are available at: <https://wiki.shibboleth.net/confluence/display/SHIB2/NativeSPApacheConfig> General information about installing/configuring Shibboleth Service Providers (SPs) can be found at: <https://wiki.shibboleth.net/confluence/display/SHIB2/Installation>

A few extra notes/hints when configuring mod_shib & Apache:

- The Shibboleth setting "ShibUseHeaders" is no longer required to be set to "On", as DSpace will correctly utilize attributes instead of headers.
 - When "ShibUseHeaders" is set to "Off" (which is recommended in the [mod_shib documentation](#)), proper configuration of Apache to pass attributes to Tomcat (via either mod_jk or mod_proxy) can be a bit tricky, SWITCH has [some great documentation](#) on exactly what you need to do. We will eventually paraphrase/summarize this documentation here, but for now, the SWITCH page will have to do.
- When initially setting up Apache & mod_shib, <https://www.testshib.org/> provides a great testing ground for your configurations. This site provides a sample/demo Shibboleth IdP (as well as a sample Shibboleth SP) which you can test against. It acts as a "sandbox" to get your configurations working properly, before you point DSpace at your production Shibboleth IdP.

Below, we have provided a sample Apache configuration. However, as every institution has their own specific Apache setup/configuration, it is highly likely that you will need to tweak this configuration in order to get it working properly. Again, see the official mod_shib documentation for much more detail about each of these settings: <https://wiki.shibboleth.net/confluence/display/SHIB2/NativeSPApacheConfig> These configurations are meant to be added to an Apache <VirtualHost> which acts as a proxy to your Tomcat (or other servlet container) running DSpace. More information on Apache VirtualHost settings can be found at: <https://httpd.apache.org/docs/2.2/vhosts/>

```
#### SAMPLE MOD_SHIB CONFIGURATION FOR APACHE2 (it may require local modifications based on your
Apache setup) ####
# While this sample VirtualHost is for HTTPS requests (recommended for Shibboleth, obviously),
# you may also need/want to create one for HTTP (*:80)
<VirtualHost *:443>
    ...
```

```

# PLEASE NOTE: We have omitted many Apache settings (ServerName, LogLevel, SSLCertificateFile,
etc)
# which you may need/want to add to your VirtualHost
# As long as Shibboleth module is installed, enable all Shibboleth/mod_shib related settings
<IfModule mod_shib>
  # Shibboleth recommends turning on UseCanonicalName
  # See "Prepping Apache" in https://wiki.shibboleth.net/confluence/display/SHIB2/
NativeSPApacheConfig
  UseCanonicalName On
  # Most DSpace instances will want to use Shibboleth "Lazy Session", which ensures that users

  # can access DSpace without first authenticating via Shibboleth.
  # This section turns on Shibboleth "Lazy Session". Also ensures that once they have
authenticated
  # (by accessing /Shibboleth.sso/Login path), then their Shib session is kept alive
  <Location />
    AuthType shibboleth
    ShibRequireSession Off
    require shibboleth
    # If your "shibboleth2.xml" file specifies an <ApplicationOverride> setting for your
    # DSpace Service Provider, then you may need to tell Apache which "id" to redirect Shib
requests to.
    # Just uncomment this and change the value "my-dspace-id" to the associated @id attribute
value.
    #ShibRequestSetting applicationId my-dspace-id
  </Location>
  # If a user attempts to access the DSpace shibboleth login page, force them to authenticate
via Shib
  <Location "/shibboleth-login">
    AuthType shibboleth
    ShibRequireSession On
    # Please note that setting ShibUseHeaders to "On" is a potential security risk.
    # You may wish to set it to "Off". See the mod_shib docs for details about this setting:
    # https://wiki.shibboleth.net/confluence/display/SHIB2/NativeSPApacheConfig#
NativeSPApacheConfig-AuthConfigOptions
    # Here's a good guide to configuring Apache + Tomcat when this setting is "Off":
    # https://www.switch.ch/de/aa1/support/serviceproviders/sp-access-rules.html#
javaapplications
    ShibUseHeaders On
    require valid-user
  </Location>
  # Ensure /Shibboleth.sso path (in Apache) can be accessed
  # By default it may be inaccessible if your Apache security is tight.
  <Location "/Shibboleth.sso">
    Order deny,allow
    Allow from all
    # Also ensure Shibboleth/mod_shib responds to this path
    SetHandler shib
  </Location>

  # Finally, you may need to ensure requests to /Shibboleth.sso are NOT redirected
  # to Tomcat (as they need to be handled by mod_shib instead).
  # NOTE: THIS SETTING IS LIKELY ONLY NEEDED IF YOU ARE USING mod_proxy TO REDIRECT

```

```
# ALL REQUESTS TO TOMCAT (e.g. ProxyPass / ajp://localhost:8009/)
# ProxyPass /Shibboleth.sso !
</IfModule>

...

</VirtualHost>
```

DSpace Shibboleth Configuration Options

Authentication Methods:

DSpace supports authentication using NetID, or email address. A user's NetID is a unique identifier from the IdP that identifies a particular user. The NetID can be of almost any form such as a unique integer, string, or with Shibboleth 2.0 you can use "targeted ids". You will need to coordinate with your shibboleth federation or identity provider. There are three ways to supply identity information to DSpace:

1) NetID from Shibboleth Header (*best*)

The NetID-based method is superior because users may change their email address with the identity provider. When this happens DSpace will not be able to associate their new address with their old account.

2) Email address from Shibboleth Header (*okay*)

In the case where a NetID header is not available or not found DSpace will fall back to identifying a user based-upon their email address.

3) Tomcat's Remote User (*worst*)

In the event that neither Shibboleth headers are found then as a last resort DSpace will look at Tomcat's remote user field. This is the least attractive option because Tomcat has no way to supply additional attributes about a user. Because of this the autoregister option is not supported if this method is used.

Identity Scheme Migration Strategies:

If you are currently using Email based authentication (either 1 or 2) and want to upgrade to NetID based authentication then there is an easy path. Simply enable shibboleth to pass the NetID attribute and set the netid-header below to the correct value. When a user attempts to log in to DSpace first DSpace will look for an EPerson with the passed NetID, however when this fails DSpace will fall back to email based authentication. Then DSpace will update the user's EPerson account record to set their netted so all future authentications for this user will be based upon netted. One thing to note is that DSpace will prevent an account from switching NetIDs. If an account already has a NetID set and then they try and authenticate with a different NetID the authentication will fail.

EPerson Metadata:

One of the primary benefits of using Shibboleth based authentication is receiving additional attributes about users such as their names, telephone numbers, and possibly their academic department or graduation semester if desired. DSpace treats the first and last name attributes differently because they (along with email address) are the three pieces of minimal information required to create a new user account. For both first and last name supply direct mappings to the Shibboleth headers. In addition to the first and last name DSpace supports other metadata fields such as phone, or really anything you want to store on an eperson object. Beyond the phone field, which is accessible in the user's profile screen, none of these additional metadata fields will be used by DSpace out-of-the box. However if you develop any local modification you may access these attributes from the EPerson object. The Vireo ETD workflow system utilizes this to aid students when submitting an ETD.

Role-based Groups:

DSpace is able to place users into pre-defined groups based upon values received from Shibboleth. Using this option you can place all faculty members into a DSpace group when the correct affiliation's attribute is provided. When DSpace does this they are considered 'special groups', these are really groups but the user's membership within these groups is not recorded in the database. Each time a user authenticates they are automatically placed within the pre-defined DSpace group, so if the user loses their affiliation then the next time they login they will no longer be in the group.

Depending upon the shibboleth attributed use in the role-header it may be scoped. Scoped is shibboleth terminology for identifying where an attribute originated from. For example a students affiliation may be encoded as "student@tamu.edu". The part after the @ sign is the scope, and the preceding value is the value. You may use the whole value or only the value or scope. Using this you could generate a role for students and one institution different than students at another institution. Or if you turn on ignore-scope you could ignore the institution and place all students into one group.

The values extracted (a user may have multiple roles) will be used to look up which groups to place the user into . The groups are defined as "role.<role-name>" which is a comma separated list of DSpace groups.

Configuration File:	<code>[dspace]/config/modules/authentication-shibboleth.cfg</code>
Property:	<code>lazysession</code>
Example Value:	<code>lazysession = true</code>
Informational Note:	Whether to use lazy sessions or active sessions. For more DSpace instances, you will likely want to use lazy sessions. Active sessions will force every user to authenticate via Shibboleth before they can access your DSpace (essentially resulting in a "dark archive").
Property:	<code>lazysession.loginurl</code>
Example Value:	<code>lazysession.loginurl = /Shibboleth.sso/Login</code>

Configuration File:	<code>[dspace]/config/modules/authentication-shibboleth.cfg</code>
Informational Note:	The url to start a shibboleth session (only for lazy sessions). Generally this setting will be <code>"/Shibboleth.sso/Login"</code>
Property:	<code>lazysession.secure</code>
Example Value:	<code>lazysession.secure = true</code>
Informational Note:	Force HTTPS when authenticating (only for lazy sessions). Generally this is recommended to be <code>"true"</code> .
Property:	<code>netid-header</code>
Example Value:	<code>netid-header = SHIB-NETID</code>
Informational Note:	The HTTP header where shibboleth will supply a user's NetID. This HTTP header should be specified as an Attribute within your Shibboleth <code>"attribute-map.xml"</code> configuration file.
Property:	<code>email-header</code>
Example Value:	<code>email-header = SHIB-MAIL</code>
Informational Note:	The HTTP header where the shibboleth will supply a user's email address. This HTTP header should be specified as an Attribute within your Shibboleth <code>"attribute-map.xml"</code> configuration file.
Property:	<code>email-use-tomcat-remote-user</code>
Example Value:	<code>email-use-tomcat-remote-user = false</code>
Informational Note:	Used when a netid or email headers are not available should Shibboleth authentication fall back to using Tomcat's remote user feature? Generally this is not recommended. See the "Authentication Methods" section above.
Property:	<code>reconvert.attributes</code>
Example Value:	<code>reconvert.attributes = false</code>
Informational Note:	Shibboleth attributes are by default UTF-8 encoded. Some servlet container automatically converts the attributes from ISO-8859-1 (latin-1) to UTF-8. As the attributes already were UTF-8 encoded it may be necessary to reconvert them. If you set this property true, DSpace

Configuration File:	<code>[dspace]/config/modules/authentication-shibboleth.cfg</code>
	converts all shibboleth attributes retrieved from the servlet container from UTF-8 to ISO-8859-1 and uses the result as if it were UTF-8. This procedure restores the shibboleth attributes if the servlet container wrongly converted them from ISO-8859-1 to UTF-8. Set this true, if you notice character encoding problems within shibboleth attributes.
Property:	<code>autoregister</code>
Example Value:	<code>autoregister = true</code>
Informational Note:	Should we allow new users to be registered automatically?
Property:	<code>sword.compatibility</code>
Example Value:	<code>sword.compatibility = true</code>
Informational Note:	Sword compatibility will allow this authentication method to work when using sword. Sort relies on username and password based authentication and is entirely incapable of supporting shibboleth. This option allows you to authenticate username and passwords for sword sessions with out adding another authentication method onto the stack. You will need to ensure that a user has a password. One way to do that is to create the user via the create-administrator command line command and then edit their permissions.
Property:	<code>firstname-header</code>
Example Value:	<code>firstname-header = SHIB_GIVENNAME</code>
Informational Note:	The HTTP header where the shibboleth will supply a user's given name. This HTTP header should be specified as an Attribute within your Shibboleth "attribute-map.xml" configuration file.
Property:	<code>lastname-header</code>
Example Value:	<code>lastname-header = SHIB_SN</code>
Informational Note:	The HTTP header where the shibboleth will supply a user's surname. This HTTP header should be specified as an Attribute within your Shibboleth "attribute-map.xml" configuration file.
Property:	<code>eperson.metadata</code>

Configuration File:	<code>[dspace]/config/modules/authentication-shibboleth.cfg</code>
Example Value:	<pre> eperson.metadata = \ SHIB-telephone => phone, \ SHIB-cn => cn </pre>
Informational Note:	Additional user attributes mapping, multiple attributes may be stored for each user. The left side is the Shibboleth-based metadata Header and the right side is the eperson metadata field to map the attribute to.
Property:	<code>eperson.metadata.autocreate</code>
Example Value:	<code>eperson.metadata.autocreate = true</code>
Informational Note:	If the eperson metadata field is not found, should it be automatically created?
Property:	<code>role-header</code>
Example Value:	<code>role-header = SHIB_SCOPED_AFFILIATION</code>
Informational Note:	The shibboleth header to do role-based mappings (see section on roll based mapping section above)
Property:	<code>role-header.ignore-scope</code>
Example Value:	<code>role-header.ignore-scope = true</code>
Informational Note:	Whether to ignore the attribute's scope (everything after the @ sign for scoped attributes)
Property:	<code>role-header.ignore-value</code>
Example Value:	<code>role-header.ignore-value = false</code>
Informational Note:	Whether to ignore the attribute's value (everything before the @ sign for scoped attributes)
Property:	<code>role.[affiliation-attribute]</code>

Configuration File:	<code>[dspace]/config/modules/authentication-shibboleth.cfg</code>
Example Value:	<pre> role.faculty = Faculty, Member \ role.staff = Staff, Member \ role.student = Students, Member </pre>
Informational Note:	Mapping of affiliation values to DSpace groups. See the "Role-based Groups" section above for more info.

LDAP Authentication

Enabling LDAP Authentication

To enable LDAP Authentication, you must ensure the `org.dspace.authenticate.LDAPAuthentication` class is listed as one of the `AuthenticationMethods` in the following configuration:

Configuration File:	<code>[dspace]/config/modules/authentication.cfg</code>
Property:	<code>plugin.sequence.org.dspace.authenticate.AuthenticationMethod</code>
Example Value:	<pre> plugin.sequence.org.dspace.authenticate.AuthenticationMethod = \ org.dspace.authenticate.LDAPAuthentication </pre>

Configuring LDAP Authentication

If LDAP is enabled, 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](#) below for more information about how to do this.

Here is an explanation of each of the different LDAP configuration parameters:

Configuration File:	<code>[dspace]/config/modules/authentication-ldap.cfg</code>
Property:	<code>enable</code>

Configuration File:	<code>[dspace]/config/modules/authentication-ldap.cfg</code>
Example Value:	<code>enable = false</code>
Informational Note:	This setting will enable or disable LDAP authentication in DSpace. With the setting off, users will be required to register and login with their email address. With this setting on, users will be able to login and register with their LDAP user ids and passwords.
Property:	<code>autoregister</code>
Example Value:	<code>autoregister = true</code>
Informational Note:	This will turn LDAP autoregistration on or off. With this on, a new EPerson object will be created for any user who successfully authenticates against the LDAP server when they first login. With this setting off, the user must first register to get an EPerson object by entering their ldap username and password and filling out the forms.
Property:	<code>provider_url</code>
Example Value:	<code>provider_url = ldap://ldap.myu.edu/o=myu.edu</code>
Informational Note:	This is the url to your institution's LDAP server. You may or may not need the /o=myu.edu part at the end. Your server may also require the ldaps:// protocol. (This field has no default value)
Property:	<code>id_field</code>
Example Value:	<code>id_field = uid</code>
Explanation:	This is the unique identifier field in the LDAP directory where the username is stored. (This field has no default value)
Property:	<code>object_context</code>
Example Value:	<code>object_context = ou=people, o=myu.edu</code>
Informational Note:	This is the LDAP object context to use when authenticating the user. By default, DSpace will use this value to create the user's DN in order to attempt to authenticate them. It is appended to the <i>id_field</i> and username. For example <code>uid=username,ou=people,o=myu.edu</code> . You will need to modify this to match your LDAP configuration. (This field has no default value)


Configuration File:	<code>[dspace]/config/modules/authentication-ldap.cfg</code>
	If your users do NOT all exist under a single "object_context" in LDAP, then you should ignore this setting and INSTEAD use the Hierarchical LDAP Authentication settings below (especially see "search.user" or "search.anonymous")
Property:	<code>search_context</code>
Example Value:	<code>search_context = ou=people</code>
Informational Note:	This is the search context used when looking up a user's LDAP object to retrieve their data for autoregistering. With <code>autoregister=true</code> , when a user authenticates without an EPerson object we search the LDAP directory to get their name (<code>id_field</code>) and email address (<code>email_field</code>) so that we can create one for them. So after we have authenticated against <code>uid=username,ou=people,o=byu.edu</code> we now search in <code>ou=people</code> for filtering on <code>[uid=username]</code> . Often the <code>search_context</code> is the same as the <code>object_context</code> parameter. But again this depends on your LDAP server configuration. (This field has no default value, and it MUST be specified when either <code>search.anonymous=true</code> or <code>search.user</code> is specified)
Property:	<code>email_field</code>
Example Value:	<code>email_field = mail</code>
Informational Note:	This is the LDAP object field where the user's email address is stored. "mail" is the most common for LDAP servers. (This field has no default value) If the "email_field" is unspecified, or the user has no email address in LDAP, his/her username (<code>id_field</code> value) will be saved as the email in DSpace (or appended to <code>netid_email_domain</code> , when specified)
Property:	<code>netid_email_domain</code>
Example Value:	<code>netid_email_domain = @example.com</code>
Informational Note:	If your LDAP server does not hold an email address for a user (i.e. no <code>email_field</code>), you can use the following field to specify your email domain. This value is appended to the <code>netid (id_field)</code> in order to make an email address (which is then stored in the DSpace EPerson). For example, a <code>netid</code> of 'user' and <code>netid_email_domain</code> as <code>@example.com</code> would set the email of the user to be user@example.com

Configuration File:	<code>[dspace]/config/modules/authentication-ldap.cfg</code>
	<i>Please note:</i> this field will only be used if "email_field" is unspecified OR the user in question has no email address stored in LDAP. If both "email_field" and "netid_email_domain" are unspecified, then the "id_field" will be used as the email address.
Property:	<code>surname_field</code>
Example Value:	<code>surname_field = sn</code>
Informational Note:	This is the LDAP object field where the user's last name is stored. "sn" is the most common for LDAP servers. If the field is not found the field will be left blank in the new eperson object. (This field has no default value)
Property:	<code>givenname_field</code>
Example Value:	<code>givenname_field = givenName</code>
Informational Note:	This is the LDAP object field where the user's given names are stored. I'm not sure how common the givenName field is in different LDAP instances. If the field is not found the field will be left blank in the new eperson object. (This field has no default value)
Property:	<code>phone_field</code>
Example Value:	<code>phone_field = telephoneNumber</code>
Informational Note:	This is the field where the user's phone number is stored in the LDAP directory. If the field is not found the field will be left blank in the new eperson object. (This field has no default value)
Property:	<code>login.specialgroup</code>
Example Value:	<code>login.specialgroup = group-name</code>
Informational Note:	If specified, all users who successfully login via LDAP will automatically become members of this DSpace Group (for the remainder of their current, logged in session). This DSpace Group must already exist (it will not be automatically created). This is useful if you want a DSpace Group made up of all internal authenticated users. This DSpace Group can then be used to bestow special permissions on any users who have authenticated via LDAP (e.g. you could allow anyone authenticated via LDAP to view special, on campus only collections or similar)

Configuration File:	<code>[dspace]/config/modules/authentication-ldap.cfg</code>
Property:	<code>login.groupmap.*</code>
Example Value:	<pre>login.groupmap.1 = ou=Students:ALL_STUDENTS login.groupmap.2 = ou=Employees:ALL_EMPLOYEES login.groupmap.3 = ou=Faculty:ALL_FACULTY</pre>
Informational Note:	<p>The left part of the value (before the ":") must correspond to a portion of a user's DN (unless "login.group.attribute" is specified..please see below). The right part of the value corresponds to the name of an existing DSpace group.</p> <p>For example, if the authenticated user's DN in LDAP is in the following form:</p> <pre>cn=jdoe,OU=Students,OU=Users,dc=example,dc=edu</pre> <p>that user would get assigned to the ALL_STUDENTS DSpace group for the remainder of their current session.</p> <p>However, if that same user later graduates and is employed by the university, their DN in LDAP may change to:</p> <pre>cn=jdoe,OU=Employees,OU=Users,dc=example,dc=edu</pre> <p>Upon logging into DSpace after that DN change, the authenticated user would now be assigned to the ALL_EMPLOYEES DSpace group for the remainder of their current session.</p> <p><i>Note:</i> This option can be used independently from the login.specialgroup option, which will put all LDAP users into a single DSpace group. Both options may be used together.</p>
Property:	<code>login.groupmap.attribute</code>
Example Value:	<code>login.groupmap.attribute = group</code>
Informational Note:	<p>The value of the "login.groupmap.attribute" should specify the name of a single LDAP attribute. If this property is uncommented, it changes the meaning of the left part of "login.groupmap.*" (see above) as follows:</p> <ul style="list-style-type: none"> • If the authenticated user has this LDAP attribute, look up the value of this LDAP attribute in the left part (before the ":") of the <code>login.groupmap.*</code> value • If that LDAP value is found in any "login.groupmap.*" field, assign this authenticated user to the DSpace Group specified by the right part (after the ":") of the <code>login.groupmap.*</code> value. <p>For example:</p>

Configuration File:	<code>[dspace]/config/modules/authentication-ldap.cfg</code>
	<ul style="list-style-type: none"> • <code>login.groupmap.attribute = group</code> • <code>login.groupmap.1 = mathematics:Mathematics_Group</code> <p>The above would ensure that any authenticated users where their LDAP "group" attribute equals "mathematics" would be added to the DSpace Group named "Mathematics_Group" for the remainder of their current session. However, if that same user logged in later with a new LDAP "group" value of "computer science", he/she would no longer be a member of the "Mathematics_Group" in DSpace.</p>

Enabling Hierarchical LDAP Authentication

 Please note, that DSpace doesn't contain the `LDAPHierarchicalAuthentication` class anymore. This functionality is now supported by `LDAPAuthentication`, which uses the same configuration options.

If your users are spread out across a hierarchical tree on your LDAP server, you may wish to have DSpace search for the user name in your tree. Here's how it works:

1. DSpace gets the user name from the login form
2. DSpace binds to LDAP as an administrative user with right to search in DNs (LDAP may be configured to allow anonymous users to search)
3. DSpace searches for the user name as within DNs (username is a part of full DN)
4. DSpace binds with the found full DN and password from login form
5. DSpace logs user in if LDAP reports successful authentication; refuses login otherwise

Configuring Hierarchical LDAP Authentication

Hierarchical LDAP Authentication shares all the above standard [LDAP configurations](#), but has some additional settings.

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

Configuration File:	<code>[dspace]/config/modules/authentication-ldap.cfg</code>
Property:	<code>search_scope</code>
Example Value:	<code>search_scope = 2</code>

Configuration File:	<code>[dspace]/config/modules/authentication-ldap.cfg</code>
Informational Note:	<p>This is the search scope value for the LDAP search during autoregistering (<code>autoregister=true</code>). This will depend on your LDAP server setup, and is only really necessary if your users are spread out across a hierarchical tree on your LDAP server. This value must be one of the following integers corresponding to the following values:</p> <pre>object scope : 0 one level scope : 1 subtree scope : 2</pre> <p>Please note that "search_context" in the LDAP configurations must also be specified.</p>
Property:	<code>search.anonymous</code>
Example Value:	<code>search.anonymous = true</code>
Informational Note:	<p>If true, DSpace will anonymously search LDAP (in the "search_context") for the DN of the user trying to login to DSpace. This setting is "false" by default. By default, DSpace will either use "search.user" to authenticate for the LDAP search (if search.user is specified), or will use the "object_context" value to create the user's DN.</p>
Property:	<code>search.user</code> <code>search.password</code>
Example Value:	<code>search.user = cn=admin,ou=people,o=myu.edu</code> <code>search.password = password</code>
Informational Note:	<p>The full DN and password of a user allowed to connect to the LDAP server and search (in the "search_context") for the DN of the user trying to login. By default, if unspecified, DSpace will either search LDAP anonymously for the user's DN (when <code>search.anonymous=true</code>), or will use the "object_context" value to create the user's DN.</p>

IP Authentication

Enabling IP Authentication

To enable IP Authentication, you must ensure the `org.dspace.authenticate.IPAuthentication` class is listed as one of the `AuthenticationMethods` in the following configuration:

Configuration File:	<code>[dspace]/config/modules/authentication.cfg</code>
Property:	<code>plugin.sequence.org.dspace.authenticate.AuthenticationMethod</code>
Example Value:

Configuration File:	<code>[dspace]/config/modules/authentication.cfg</code>
	<pre>plugin.sequence.org.dspace.authenticate.AuthenticationMethod = \ org.dspace.authenticate.IPAuthentication</pre>

Configuring IP Authentication

Configuration File:	<code>[dspace]/config/modules/authentication-ip.cfg</code>
----------------------------	--

Once enabled, you are then able to map DSpace groups to IP addresses in `authentication-ip.cfg` by setting `ip.GROUPNAME = iprange[, iprange ...]`, e.g:

```
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
2001:18e8::32                          # IPv6 too
```

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

Notes:

- If the Groupname contains blanks you must escape the spaces, e.g. "Department\ of \ Statistics"
- If your DSpace installation is hidden behind a web proxy, remember to set the `useProxies` configuration option within the 'Logging' section of `dspace.cfg` to use the IP address of the user rather than the IP address of the proxy server.

X.509 Certificate Authentication

Enabling 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`

Configuration File:	<code>[dspace]/config/modules/authentication.cfg</code>
Property:	<code>plugin.sequence.org.dspace.authenticate.AuthenticationMethod</code>
Example Value:	<pre>plugin.sequence.org.dspace.authenticate.AuthenticationMethod = \ org.dspace.authenticate.X509Authentication, \ org.dspace.authenticate.PasswordAuthentication</pre>

Configuring X.509 Certificate Authentication

Configuration File: `[dspace]/config/modules/authentication-x509.cfg`

1. 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

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

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

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

2. 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 `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.

4.1.2 Embargo

- [1 What is an Embargo?](#)
- [2 DSpace 3.0 New Embargo Functionality](#)
- [3 Configuring and using Embargo in DSpace 3.0+](#)
 - [3.1 Introduction](#)
 - [3.2 dspace.cfg](#)
 - [3.3 Submission Process](#)
 - [3.3.1 item-submission.xml](#)
 - [3.3.2 Simple Embargo Settings](#)
 - [3.3.2.1 AccessStep](#)
 - [3.3.2.2 UploadWithEmbargoStep](#)
 - [3.3.3 Advanced Embargo Settings](#)
 - [3.3.3.1 AccessStep](#)
 - [3.3.3.2 UploadWithEmbargoStep](#)
 - [3.3.3.3 Restrict list of displayed groups to specific \(sub\)groups](#)
 - [3.4 Private/Public Item](#)
 - [3.5 Pre-3.0 Embargo Migration Routine](#)
- [4 Technical Specifications](#)
 - [4.1 Introduction](#)
 - [4.2 ResourcePolicy](#)
 - [4.3 Item](#)
 - [4.4 Item.inheritCollectionDefaultPolicies\(Collection c\)](#)
 - [4.5 AuthorizeManager](#)
 - [4.6 Withdraw Item](#)
 - [4.7 Reinstate Item](#)
 - [4.8 Pre-DSpace 3.0 Embargo Compatibility](#)
- [5 Pre-DSpace 3.0 Embargo](#)
 - [5.1 Embargo model and life-cycle](#)
 - [5.1.1 Terms assignment](#)
 - [5.1.2 Terms interpretation/imposition](#)
 - [5.1.3 Embargo period](#)
 - [5.1.4 Embargo lift](#)
 - [5.1.5 Post embargo](#)
 - [5.2 Configuration](#)
 - [5.3 Operation](#)
 - [5.4 Extending embargo functionality](#)

- [5.4.1 Setter](#)
- [5.4.2 Lifter](#)

What is an Embargo?

An embargo is a temporary access restriction placed on metadata or bitstreams. Its scope or duration may vary, but the fact that it eventually expires is what distinguishes it from other content restrictions. For example, it is not unusual for content destined for DSpace to come with permanent restrictions on use or access based on license-driven or other IP-based requirements that limit access to institutionally affiliated users. Restrictions such as these are imposed and managed using standard administrative tools in DSpace, typically by attaching specific policies to Items, Collections, Bitstreams, etc. Embargo functionality was originally introduced as part of DSpace 1.6, enabling embargoes on the level of items that applied to all bitstreams included in the item. In DSpace 3.0, this functionality has been extended for the XML User Interface, enabling embargoes on the level of individual bitstreams.

DSpace 3.0 New Embargo Functionality

Embargoes can be applied per *item* and per *bitstream*. The *item* level embargo will be the default for every *bitstream*, although it could be customized at *bitstream* level.

As a DSpace administrator, you can choose to integrate either Simple or Advanced dialog screens as part of the item submission process. These are outlined in detail in the sections [Simple Embargo Settings](#) and [Advanced Embargo Settings](#).

This preference is stored in the `dspace.cfg` value `webui.submission.restrictstep.enableAdvancedForm`. If not set, the default is for Simple Embargo.



Configuration name changed

Please note that the configuration parameter name has been changed in DSpace 4.0 from `xmlui.submission.restrictstep.enableAdvancedForm` to `webui.submission.restrictstep.enableAdvancedForm`

On the level of an individual item, a new Private/Public state has been introduced to control the visibility of item metadata in the different indexes serving the DSpace web interface (search, browse, discovery), as well as machine interfaces (REST-API, OAI-PMH, ...)

The following functionality has been added in DSpace 3.0:

- *Browse private items*
- *Submission Process*
 - Simple/Advanced Access Step
 - Upload with embargo step
- *Edit Item*

- Make it Private
- Make it Public

The following functionality has been modified in DSpace 3.0:

- Edit Item
- Edit Bitstream
- Wildcard Policy Admin Tool

Configuring and using Embargo in DSpace 3.0+

Introduction

The following sections describe the steps needed to configure and use the new Embargo functionality in DSpace.

Note: when the embargo will be set at *item* level or *bitstream* level a new *ResourcePolicy* will be added.

dspace.cfg

As already mentioned the user will be given the opportunity to choose between:

- Simple Embargo Settings
- Advanced Embargo Settings

To switch between the two, you need to set following variable in the *dspace.cfg* file. A value of *false* (the default) enables the simple settings while a value of *true* enables the advanced settings.

```
webui.submission.restrictstep.enableAdvancedForm=false
```

Submission Process

item-submission.xml

To enable the new embargo, changes are required to the *item-submission.xml* file, located in your config directory. This file determines which steps are executed in the submission of a new item.

Two new submission steps have been introduced in the file. By default, they are not activated yet:

- *AccessStep*: the step in which the user can set the embargo at item level, effectively restricting access to the item metadata.
- *UploadWithEmbargoStep*: the step in which the user can set the embargo at bitstream level. **If this step is enabled, the old *UploadStep* must be disabled. Leaving both steps enabled will result in a system failure.**

Here is an extract from the new file:

```

<!--Step 3 will be to Manage Item access.
  <step>
    <heading>submit.progressbar.access</heading>
    <processing-class>org.dspace.submit.step.AccessStep</processing-class>
    <jspui-binding>org.dspace.app.webui.submit.step.JSPAccessStep</jspui-binding>
    <xmlui-binding>org.dspace.app.xmlui.aspect.submission.submit.AccessStep</xmlui-binding>
    <workflow-editable>true</workflow-editable>
  </step>
-->
<!-- Step 4 Upload Item with Embargo Features (not supported in JSPUI)
  to enable this step, please make sure to comment-out the previous step "UploadStep"
  <step>
    <heading>submit.progressbar.upload</heading>
    <processing-class>org.dspace.submit.step.UploadWithEmbargoStep</processing-class>
    <jspui-binding>org.dspace.app.webui.submit.step.JSPUploadWithEmbargoStep</jspui-binding>
    <xmlui-binding>org.dspace.app.xmlui.aspect.submission.submit.UploadWithEmbargoStep</
xmlui-binding>
    <workflow-editable>true</workflow-editable>
  </step>
-->

```

To enable the new Embargo, ensure that the new steps are uncommented and the old UploadStep is commented out.

Simple Embargo Settings

Using the simple embargo settings, submitters will be able to define embargoes bound to specific dates, that are applied to all anonymous and default read access. To keep the interface simple, options to apply embargoes for particular groups of DSpace users are not shown. The simple embargo settings interface assumes that embargoes always start immediately upon submission, so only end dates are configurable.

AccessStep

The simple *AccessStep* Embargo form renders three options for the user:

- *Private item*: to hide an item's metadata from all search and browse indexes, as well as external interfaces such as OAI-PMH.
- *Embargo Access until Specific Date*: to indicate a date until which the item will be embargoed.
- *Reason*: to elaborate on the specific reason why an item is under embargo.

When Embargo is set, it applies to Anonymous or to any other Group that is indicated to have *default read access* for that specific collection.

This shows how the Access step is rendered, using the simple embargo settings:

Item submission

Initial Questions → Describe → Describe → **Access** → Upload → Review → License →

Complete

Access Settings

Private Item:
If selected, the item won't be searchable

Embargo Access until Specific Date:
Accepted format: mm/dd/yyyy,mm/yyyy, yyyy

Reason:

< Previous Save & Exit Next >

UploadWithEmbargoStep

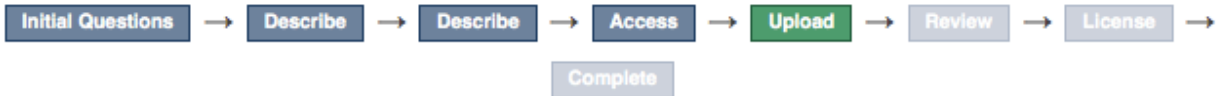
The simple *UploadWithEmbargoStep* form renders two new fields for the user:

- *Embargo Access until Specific Date*: to indicate a date until which the *bitstream* will be embargoed. When left empty, no embargo will be applied.
- *Reason*: to elaborate on the specific reason why the *bitstream* is under embargo.

These fields will be preloaded with the values set in the *AccessStep*.

The following picture shows the form for the Upload step, rendered using the simple embargo settings with preloaded values:

Item submission



Upload File(s)

File:

Please enter the full path of the file on your computer corresponding to your item. If you click "Browse...", a new window will allow you to select the file from your computer.

File Description:

Optionally, provide a brief description of the file, for example "*Main article*", or "*Experiment data readings*".

Embargo Access until Specific Date:

Accepted format: mm/dd/yyyy,mm/yyyy, yyyy

Reason:

Advanced Embargo Settings

The Advanced Embargo settings are really designed with a submitter in mind who is aware of user groups in DSpace and has understanding of how Resource Policies work.

AccessStep

The Advanced *AccessStep* Embargo step allows the users to manage more fine-grained resource policies to attach to the item.

The form will render the following fields:

- *Policies List*: list of the custom policies already added.
- *Private Item*: whether to hide an item's metadata from all search and browse indexes, as well as external interfaces such as OAI-PMH.
- *Name*: to give a name to the policy.
- *Groups*: to indicate the user groups to which the policy will apply.
- *Visible/Embargoed*: whether the Item will be visible or embargoed for that specific group.
- *Embargo Access until Specific Date*: to indicate a date until which the item will be embargoed.
- *Reason*: to elaborate on the specific reason why the policy is applied.

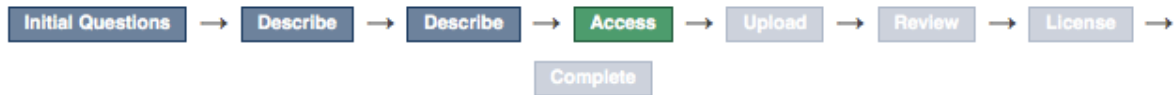
The last two fields will be enabled only when *Embargoed* has been selected.

This step gives the opportunity to the user to manage the policy manually, so that combinations such as the following will be possible:

- Set Embargo for Anonymous
- Set Embargo for anyone, except for the users belonging to a specific group
- Set Embargo for specific groups, but not for other groups ...

Here is a screenshot of the Access step form that will be rendered for the advanced embargo settings:

Item submission



Policies List

Name	Action	Group	Start Date	End Date		
Embargo Policy	READ	Anonymous	2013-01-01		Edit	Remove

Access Settings

Private Item:

If selected, the item won't be searchable

Name:

Groups:

Visible/Embargoed:

Item will be visible once accepted into archive Embargo Access until Specific Date

Accepted format: mm/dd/yyyy,mm/yyyy, yyyy

Reason:

[Confirm Policy & add another](#)

UploadWithEmbargoStep

UploadWithEmbargoStep for Advanced Embargo settings displays an additional *Policies* button next to *Edit* in the list of uploaded files.

Clicking it brings you to the a page where you can edit existing policies on the bitstream and add new ones.

Item submission

Initial Questions →
 Describe →
 Describe →
 Access →
 Upload →
 Review →
 License →
 Complete

Upload File(s)

File:
 Please enter the full path of the file on your computer corresponding to your item. If you click "Browse...", a new window will allow you to select the file from your computer.

File Description:
 Optionally, provide a brief description of the file, for example "Main article", or "Experiment data readings".

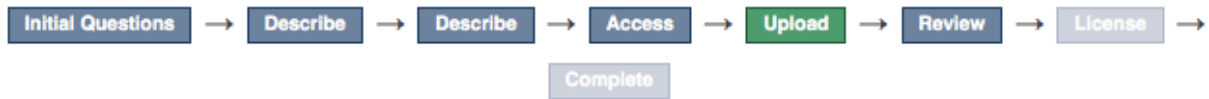
Files Uploaded

Primary	File	Size	Description	Format	
<input type="radio"/>	<input type="checkbox"/> Cameroon.gif	2517 bytes	Unknown	image/gif (Supported)	<input type="button" value="Edit"/> <input type="button" value="Policies"/>
<p>File checksum: MD5:6896170d440933f9e4fd92f5f4a6ee77</p> <div style="border: 1px solid #ccc; padding: 5px; width: fit-content; margin: 10px auto; background-color: #4a7c9c; color: white;"> <input type="button" value="Remove selected files"/> </div>					

< Previous
Save & Exit
Next >

When the button is pushed, a form similar to the one in the *AccessStep* will be rendered, making it possible to manage the policies at *bitstream* level.

Item submission



Policies List

Name	Action	Group	Start Date	End Date		
Embargo Policy	READ	Anonymous	2013-01-01		Edit	Remove

Edit Bitstream Access

Name:

Groups:

Visible/Embargoed:

Item will be visible once accepted into archive Embargo Access until Specific Date

Accepted format: mm/dd/yyyy,mm/yyyy, yyyy

Reason:

[Confirm Policy & add another](#)

[Save](#)

Restrict list of displayed groups to specific (sub)groups

For large instances of DSpace, the list of Groups can be quite long. Groups can be nested. This means that not only EPersons can be members of groups, but groups themselves can belong to other groups.

When advanced embargo settings are enabled, you can limit the list of groups displayed to the submitters to subgroups of a particular group.

To use this feature, assign the super group name to following configuration value in `dspace.cfg`:

```
webui.submission.restrictstep.groups=name_of_the_supergroup
```

Configuration name changed

Please note that the configuration parameter name has been changed in DSpace 4.0 from **xmlui.submission.restrictstep.groups** to **webui.submission.restrictstep.groups**

Once a specific group is configured as supergroup here, only the groups belonging to the indicated group will be loaded in the selection dialogs. By default, all groups are loaded.

Private/Public Item

It is also possible to adjust the Private/Public state of an item after it has been archived in the repository.

Here is a screenshot showing the updated *Edit Item* dialog:

Edit Item

[Item Status](#)
[Item Bitstreams](#)
[Item Metadata](#)
[View Item](#)
[Curate](#)

Welcome to the item management page. From here you can withdraw, reinstate, move or delete the item. You may also update or add new metadata / bitstreams on the other tabs.

Item Internal ID: 81
 Handle: 123456789/67
 Last Modified: 2012-07-19 11:59:57.68
 Item Page: <http://localhost:8208/xmlui/handle/123456789/67>

Edit item's authorization policies: [Authorizations...](#)

Withdraw item from the repository: [Withdraw...](#)

Move item to another collection: [Move...](#)

Make item private: [Make it private...](#)

Completely expunge item: [Permanently delete](#)

[Return](#)

Private items are not retrievable through the DSpace search, browse or Discovery indexes.

Therefore, an admin-only view has been created to browse all private items. Here is a screenshot of this new form:

Browsing by Issue Date

Jump to a point in the index: (Choose month) ▾ (Choose year) ▾

Or type in a year:

Sort by: issue date ▾ Order: ascending ▾ Results: 20 ▾

Now showing items 0-2 of 3 [Previous Page](#) [Next Page](#)

MVP
Ginobili, Manu (2012-05-29)

Don't give up
Pierce, Paul (2012-05-29)

Fatica e sudore
Zeman, Zdenek (2012-07-06)

Now showing items 0-2 of 3 [Previous Page](#) [Next Page](#)

Search DSpace

Browse

All of DSpace
[Communities & Collections](#)
[By Issue Date](#)
[Authors](#)
[Titles](#)
[Subjects](#)

My Account

[Logout](#)
[Profile](#)
[Submissions](#)

Administrative

Access Control
[People](#)
[Groups](#)
[Authorizations](#)

Registries
[Metadata](#)
[Format](#)

[Items](#)
[Withdrawn Items](#)
[Private Items](#)
[Control Panel](#)
[Statistics](#)
[Import Metadata](#)
[Curation Tasks](#)
[Workflow overview](#)

Pre-3.0 Embargo Migration Routine

A migration routine has been developed to migrate the current Embargo to the new one.

To execute it, run the following command:

```
./dspace migrate-embargo -a
```

Technical Specifications

Introduction

The following sections illustrate the technical changes that have been made to the *back-end* to add the new *Advanced Embargo* functionality.

ResourcePolicy

When an embargo is set at *item* level or *bitstream* level, a new *ResourcePolicy* will be added.

Three new attributes have been introduced in the *ResourcePolicy* class:

- *rpname*: resource policy name

- *rptype*: resource policy type
- *rpdescription*: resource policy description

While *rpname* and *rpdescription* are fields manageable by users, the *rptype* is managed by DSpace itself. It represents a type that a resource policy can assume, among the following:

- TYPE_SUBMISSION: all the policies added automatically during the submission process
- TYPE_WORKFLOW: all the policies added automatically during the workflow stage
- TYPE_CUSTOM: all the custom policies added by users
- TYPE_INHERITED: all the policies inherited from the enclosing object (for Item, a Collection; for Bitstream, an Item).

Here is an example of all information contained in a single policy record:

```
policy_id: 4847
resource_type_id: 2
resource_id: 89
action_id: 0
eperson_id:
epersongroup_id: 0
start_date: 2013-01-01
end_date:
rpname: Embargo Policy
rpdescription: Embargoed through 2012
rptype: TYPE_CUSTOM
```

Item

To manage **Private/Public** state a new *boolean* attribute has been added to the Item:

- isDiscoverable

When an Item is private, the attribute will assume the value *false*.

Item.inheritCollectionDefaultPolicies(Collection c)

This method has been adjusted to leave custom policies, added by the users, in place and add the default collection policies only if there are no custom policies.

AuthorizeManager

Some methods have been changed on *AuthorizeManager* to manage the new fields and some convenience methods have been introduced:

```
public static List<ResourcePolicy> findPoliciesByDSOAndType(Context c, DSpaceObject o, String type)
;
public static void removeAllPoliciesByDSOAndTypeNotEqualsTo(Context c, DSpaceObject o, String type)
;
```

```
public static boolean isAnIdenticalPolicyAlreadyInPlace(Context c, DSpaceObject o, ResourcePolicy rp);  
public static ResourcePolicy createOrModifyPolicy(ResourcePolicy policy, Context context, String name, int idGroup, EPerson ePerson, Date embargoDate, int action, String reason, DSpaceObject dso);
```

Withdraw Item

The feature to withdraw an item from the repository has been modified to keep all the custom policies in place.

Reinstate Item

The feature to reinstate an item in the repository has been modified to preserve existing custom policies.

Pre-DSpace 3.0 Embargo Compatibility

The Pre-DSpace 3.0 embargo functionality (see below) has been modified to adjust the policies setter and lifter. These classes now also set the dates within the policy objects themselves in addition to setting the date in the item metadata.

Pre-DSpace 3.0 Embargo

Embargo model and life-cycle

Functionally, the embargo system allows you to attach "terms" to an item before it is placed into the repository, which express how the embargo should be applied. What do we mean by "terms" here? They are really any expression that the system is capable of turning into (1) the time the embargo expires, and (2) a concrete set of access restrictions. Some examples:

"2020-09-12" - an absolute date (i.e. the date embargo will be lifted)

"6 months" - a time relative to when the item is accessioned

"forever" - an indefinite, or open-ended embargo

"local only until 2015" - both a time and an exception (public has no access until 2015, local users OK immediately)

"Nature Publishing Group standard" - look-up to a policy somewhere (typically 6 months)

These terms are interpreted by the embargo system to yield a specific date on which the embargo can be removed (or "lifted"), and a specific set of access policies. Obviously, some terms are easier to interpret than others (the absolute date really requires none at all), and the default embargo logic understands only the most basic terms (the first and third examples above). But as we will see below, the embargo system provides you with the ability to add your own interpreters to cope with any terms expressions you wish to have. This date that is the result of the interpretation is stored with the item. The embargo system detects when that date has passed, and removes the embargo ("lifts it"), so the item bitstreams become available. Here is a more detailed life-cycle for an embargoed item:

Terms assignment

The first step in placing an embargo on an item is to attach (assign) "terms" to it. If these terms are missing, no embargo will be imposed. As we will see below, terms are carried in a configurable DSpace metadata field, so

assigning terms just means assigning a value to a metadata field. This can be done in a web submission user interface form, in a SWORD deposit package, a batch import, etc. - anywhere metadata is passed to DSpace. The terms are not immediately acted upon, and may be revised, corrected, removed, etc, up until the next stage of the life-cycle. Thus a submitter could enter one value, and a collection editor replace it, and only the last value will be used. Since metadata fields are multivalued, theoretically there can be multiple terms values, but in the default implementation only one is recognized.

Terms interpretation/imposition

In DSpace terminology, when an Item has exited the last of any workflow steps (or if none have been defined for it), it is said to be "installed" into the repository. At this precise time, the interpretation of the terms occurs, and a computed "lift date" is assigned, which like the terms is recorded in a configurable metadata field. It is important to understand that this interpretation happens only once, (just like the installation), and cannot be revisited later. Thus, although an administrator can assign a new value to the metadata field holding the terms after the item has been installed, this will have no effect on the embargo, whose "force" now resides entirely in the "lift date" value. For this reason, you cannot embargo content already in your repository (at least using standard tools). The other action taken at installation time is the actual imposition of the embargo. The default behavior here is simply to remove the read policies on all the bundles and bitstreams except for the "LICENSE" or "METADATA" bundles. See the **Extending embargo functionality** section below for how to alter this behavior. Also note that since these policy changes occur before installation, there is no time during which embargoed content is "exposed" (accessible by non-administrators). The terms interpretation and imposition together are called "setting" the embargo, and the component that performs them both is called the embargo "setter".

Embargo period

After an embargoed item has been installed, the policy restrictions remain in effect until removed. This is not an automatic process, however: a "lifter" must be run periodically to look for items whose "lift date" has passed. Note that this means the effective removal of an embargo does **not** occur on the lift date, but on the earliest date after the lift date that the lifter is run. Typically, a nightly cron-scheduled invocation of the lifter is more than adequate, given the granularity of embargo terms. Also note that during the embargo period, all metadata of the item remains visible. This default behavior can be changed. One final point to note is that the "lift date", although it was computed and assigned during the previous stage, is in the end a regular metadata field. That means, if there are extraordinary circumstances that require an administrator (or collection editor - anyone with edit permissions on metadata) to change the lift date, this can be done. Thus, one can "revise" the lift date without reference to the original terms. This date will be checked the next time the "lifter" is run. One could immediately lift the embargo by setting the lift date to the current day, or change it to "forever" to indefinitely postpone lifting.

Embargo lift

When the lifter discovers an item whose lift date is in the past, it removes ("lifts") the embargo. The default behavior of the lifter is to add the resource policies **that would have been added** had the embargo not been imposed. That is, it replicates the standard DSpace behavior, in which an item inherits its policies from its owning collection. As with all other parts of the embargo system, you may replace or extend the default behavior of the lifter (see **Extending embargo functionality** below). You may wish, e.g., to send an email to an administrator or other interested parties when an embargoed item becomes available.

Post embargo

After the embargo has been lifted, the item ceases to respond to any of the embargo life-cycle events. The values of the metadata fields reflect essentially historical or provenance values. With the exception of the additional metadata fields, the item is indistinguishable from items that were never subject to embargo.

Configuration

DSpace embargoes utilize standard metadata fields to hold both the "terms" and the "lift date". Which fields you use are configurable, and no specific metadata element is dedicated or pre-defined for use in embargo. Rather, you must specify exactly what field you want the embargo system to examine when it needs to find the terms or assign the lift date.

The properties that specify these assignments live in `dspace.cfg`:

```
# DC metadata field to hold the user-supplied embargo terms
embargo.field.terms = SCHEMA.ELEMENT.QUALIFIER
# DC metadata field to hold computed "lift date" of embargo
embargo.field.lift = SCHEMA.ELEMENT.QUALIFIER
```

You replace the placeholder values with real metadata field names. If you only need the "default" embargo behavior - which essentially accepts only absolute dates as "terms" - this is the only configuration required, except as noted below.

There is also a property for the special date of "forever":

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

which you may change to suit linguistic or other preference.

You are free to use existing metadata fields, or create new fields. If you choose the latter, you must understand that the embargo system does **not** create or configure these fields: i.e. you must follow all the standard documented procedures for actually creating them (i.e. adding them to the metadata registry, or to display templates, etc) - this does not happen automatically. Likewise, if you want the field for "terms" to appear in submission screens and workflows, you must follow the documented procedure for configurable submission (basically, this means adding the field to `input-forms.xml`). The flexibility of metadata configuration makes it easy for you to restrict embargoes to specific collections, since configurable submission can be defined per collection.

Key recommendations:

1. Use a local metadata schema. Breaking compliance with the standard Dublin Core in the default metadata registry can create a problem for the portability of data to/from of your repository.
2. If using existing metadata fields, avoid any that are automatically managed by DSpace. For example, fields like "date.issued" or "date.accessioned" are normally automatically assigned, and thus must not be recruited for embargo use.

3. Do not place the field for "lift date" in submission screens. This can potentially confuse submitters because they may feel that they can directly assign values to it. As noted in the life-cycle above, this is erroneous: the lift date gets assigned by the embargo system based on the terms. Any pre-existing value will be over-written. But see next recommendation for an exception.
4. As the life-cycle discussion above makes clear, after the terms are applied, that field is no longer actionable in the embargo system. Conversely, the "lift date" field is not actionable **until** the application. Thus you may want to consider configuring both the "terms" and "lift date" to use the same metadata field. In this way, during workflow you would see only the terms, and after item installation, only the lift date. If you wish the metadata to retain the terms for any reason, use 2 distinct fields instead.

Operation

After the fields defined for terms and lift date have been assigned in `dspace.cfg`, and created and configured wherever they will be used, you can begin to embargo items simply by entering data (dates, if using the default setter) in the terms field. They will automatically be embargoed as they exit workflow. For the embargo to be lifted on any item, however, a new administrative procedure must be added: the "embargo lifter" must be invoked on a regular basis. This task examines all embargoed items, and if their "lift date" has passed, it removes the access restrictions on the item. Good practice dictates automating this procedure using cron jobs or the like, rather than manually running it.

The lifter is available as a target of the 1.6 DSpace launcher - see launcher documentation for details.

Extending embargo functionality

The 1.6 embargo system supplies a default "interpreter/imposition" class (the "Setter") as well as a "Lifter", but they are fairly rudimentary in several respects.

Setter

The default setter recognizes only two expressions of terms: either a literal, non-relative date in the fixed format "yyyy-mm-dd" (known as ISO 8601), or a special string used for open-ended embargo (the default configured value for this is "forever", but this can be changed in `dspace.cfg` to "toujours", "unendlich", etc). It will perform a minimal sanity check that the date is not in the past. Similarly, the default setter will only remove all read policies as noted above, rather than applying more nuanced rules (e.g allow access to certain IP groups, deny the rest). Fortunately, the setter class itself is configurable and you can "plug in" any behavior you like, provided it is written in java and conforms to the setter interface. The `dspace.cfg` property:

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

controls which setter to use.

Lifter

The default lifter behavior as described above - essentially applying the collection policy rules to the item - might also not be sufficient for all purposes. It also can be replaced with another class:

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

Pre-3.0 Embargo Lifter Commands

If you have implemented the pre DSpace 3.0 [Embargo](#) feature, you will need to run it periodically to check for Items with expired embargoes and lift them.

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

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

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

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

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

4.1.3 Managing User Accounts

When a user registers an account for the purpose of subscribing to change notices, submitting content, or the like, DSpace creates an EPerson record in the database. Administrators can manipulate these records in several ways.

Please note that when a user has submitted content, his EPerson record cannot be deleted because there are references to it from the submitted item(s). If it is necessary to prevent further use of such an account, it can be marked "cannot log in".

From the browser: XMLUI

TBS

From the browser: JSPUI

TBS

From the command line

The user command

The `dspace user` command adds, lists, modifies, and deletes EPerson records.

To create a new user account:

```
[dspace]/bin/dspace user --add --email jquser@example.com -g John -s User --password hiddensecret
[dspace]/bin/dspace user --add --netid jquser --telephone 555-555-1234 --password hiddensecret
```

One of the options `--email` or `--netid` is required to name the record. The complete options are:

-a	--add	required
-m	--email	email address
-n	--netid	"netid" (a username in an external system such as a directory – see Authentication Methods for details)
-p	--password	a password for the account. Required.
-g	--givenname	First or given name

-s	--surname	Last or surname
-t	--telephone	Telephone number
-l	--language	Preferred language
-c	--requireCertificate	Certificate required? See X.509 Authentication for details.

To list accounts:

```
[dspace]/bin/dspace user --list
```

This simply lists some characteristics of each EPerson.

short	long	meaning
-L	--list	required

To modify an account:

```
[dspace]/bin/dspace user --modify -m george@example.com
```

short	long	meaning
-M	--modify	required
-m	--email	identify the account by email address
-n	--netid	identify the account by netid
-g	--givenname	First or given name
-s	--surname	Last or surname
-t	--telephone	telephone number
-l	--language	preferred language
-c	--requireCertificate	certificate required?
-C	--canLogIn	is the account enabled or disabled?
-i	--newEmail	set or change email address
-l	--newNetid	set or change netid

To delete an account:

```
[dspace]/bin/dspace user --delete -n martha
```

short	long	meaning
-d	--delete	required
-m	--email	identify the account by email address
-n	--netid	identify the account by netid

The Groomer

This tool inspects all user accounts for several conditions.

short	long	meaning
-a	--aging	find accounts not logged in since a given date
-u	--unsalted	find accounts not using salted password hashes
-b	--before	date cutoff for --aging
-d	--delete	delete disused accounts (used with --aging)

Find accounts with unsalted passwords

Earlier versions of DSpace used an "unsalted hash" method to protect user passwords. Recent versions use a salted hash. You can find accounts which have never been converted to salted hashing:

Discovering accounts with unsalted password hashes

```
[DSpace]/bin/dsrun org.dspace.eperson.Groomer -u
```

The output is a list of email addresses for matching accounts.

Find (and perhaps delete) disused accounts

You can list accounts which have not logged on since a given date:

Discovering disused accounts

```
[DSpace]/bin/dsrun org.dspace.eperson.Groomer -a -b 07/20/1969
```

The output is a tab-separated-value table of the EPerson ID, last login date, email address, netid, and full name for each matching account.

You can also have the tool delete matching accounts:

Deleting disused accounts

```
[DSpace]/bin/dsrun org.dspace.eperson.Groomer -a -b 07/20/1969 -d
```

Email Subscriptions

```
/*<![CDATA[* / div.rbtoc1424986246890 {padding: 0px;} div.rbtoc1424986246890 ul {  
list-style: none;margin-left: 0px;} div.rbtoc1424986246890 li {margin-left: 0px;  
padding-left: 0px;} /*]]>*/
```

- [Introduction](#)
- [Adding new subscriptions](#)
- [System configuration for sending out daily emails](#)

Introduction

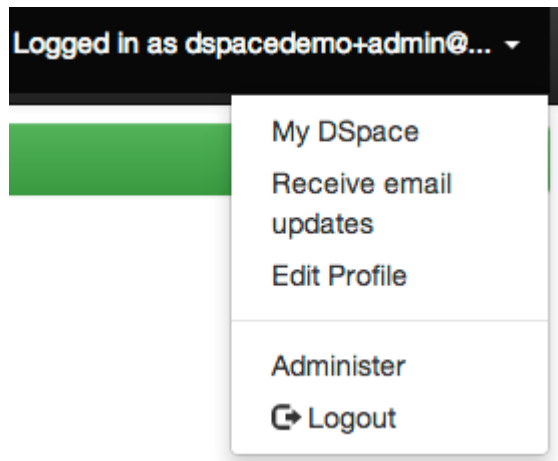
Registered users can subscribe to collections in DSpace. After subscribing, users will receive a daily email containing the new and modified items in the collections they are subscribed to.

Adding new subscriptions

Adding new subscriptions is only available to users who are logged in.

In the XML User interface, new subscriptions are added on the users Profile page.

In the JSP User Interface, a specific dialog "Receive Email Updates" is available from the dropdown in the top right corner.



Receive email updates

System configuration for sending out daily emails

To send out the subscription emails you need to invoke the **sub-daily** script from the DSpace command launcher. It is advised to setup this script as a [scheduled task using cron](#).

This script can be run with a parameter `-t` for testing purposes. When this parameter is passed, the log level is set to DEBUG to ensure that more diagnostic information will be added to the dspace logfile.

4.1.4 Request a Copy

- [Introduction](#)
- [Requesting a copy using the XML User Interface](#)
- [Requesting a copy using the JSP User Interface](#)
- [\(Optional\) Requesting a copy with Help Desk workflow](#)
- [Email templates](#)
- [Configuration parameters](#)
- [Spring Configuration Properties](#)

Introduction

The request a copy functionality was added to DSpace as a measure to facilitate access in those cases when uploaded content can not be openly shared with the entire world immediately after submission into DSpace. It gives users an efficient way to request access to the original submitter of the item, who can approve this access with the click of a button. This practice complies with most applicable policies as the submitter interacts directly with the requester on a case by case basis.

Requesting a copy using the XML User Interface

Users can request a copy by clicking the file thumbnail or the blue lock symbol displayed on files that are restricted to them.


Test item with closed access document
 Doe, John

URI: <http://hdl.handle.net/123456789/10726>
 Date: 2013


Abstract:
 This is a great piece of research. For publisher policy reasons, the associated file can currently not be openly shared with the world. However, the author is able to send you an individual copy as long as you ask nicely.

[Show full item record](#)

Files in this item



Name: Precious-Research ...
Size: 8.849Kb
Format: PDF
Description: Accepted version ...



Restricted Item XMLUI

The request form asks the user for his or her name, email address and message where the reason for requesting access can be entered.

Request a copy of the document

Enter the following information to request a copy of the document from the responsible person

Test item with closed access document

Name:

Your e-mail address:
 This email address is used for sending the document.

Files:
 All files (of this document) in restricted access. Only The requested file.

Message:

After clicking request copy at the bottom of this form, the original submitter of the item will receive an email containing the details of the request. The email also contains a link with a token that brings the original submitter to a page where he or she can either grant or reject access. If the original submitter can not evaluate the request, he or she can forward this email to the right person, who can use the link containing the token without having to log into DSpace.

Document copy request

IF YOU ARE THE AUTHOR (OR AN AUTHOR) OF DOCUMENT "Test item with closed access document" use the buttons to answer the user's request.

This repository will propose an appropriate model reply, which you may edit.

Each of these buttons registers the choice of the submitter, displaying the following form in which an additional reason for granting or rejecting the access can be added.

Document copy request

This is the text to be sent to the applicant (together with the document).

Subject:

Message:

Dear Jane Doe,
 In response to your request I have the pleasure to send you in attachment a copy of the file(s) concerning the document: "Test item with closed access document" (<http://hdl.handle.net/123456789/10726>), of which I am author (or co-author).

Best regards,
 John Doe

After hitting send, the contents of this form will be sent together with the associated files to the email address of the requester. In case the access is rejected, only the reason will be sent to the requester.

After responding positively to a request for copy, the person who approved is presented with an optional form to ask the repository administrator to alter the access rights of the item, allowing unrestricted open access to everyone.

Change permissions request

You may use this occasion to reconsider the access restrictions on the document (to avoid having to respond to these requests), if there is no reason to keep it restricted. To do so, after inserting your name and e-mail (for authentication), click the button "Change to Open Access".

Name:

E-mail:

Requesting a copy using the JSP User Interface

Users can request a copy by clicking the green "Request a Copy" button for files that are restricted to them.

Title:	Test item with closed access document
Authors:	Doe, John
Keywords:	Science Life saving knowledge Funded by taxpayer money
Issue Date:	2013
Publisher:	Not-so-green Publishing Company
Abstract:	This is a great piece of research. For publisher policy reasons, the associated file can currently not be openly shared with the world. However, the author is able to send you an individual copy as long as you ask nicely.
URI:	http://hdl.handle.net/123456789/10726
Appears in Collections:	Request a Copy Documentation

File	Description	Size	Format	
Precious-Research-Article.pdf	Accepted version of my precious scientific manuscript	8.85 kB	Adobe PDF	View/Open Request a copy

Restricted item JSPUI

The request form asks the user for his or her name, email address and message where the reason for requesting access can be entered.

Request a document copy: Test item with closed access document

Requester name:

Requester e-mail:

Files:

all files (of this document) in restricted access

the file(s) you requested

Message:

After clicking request copy at the bottom of this form, the original submitter of the item will receive an email containing the details of the request. The email also contains a link with a token that brings the original submitter to a page where he or she can either grant or reject access. If the original submitter can not evaluate the request, he or she can forward this email to the right person, who can use the link containing the token without having to log into DSpace.

After approving or rejecting the request for a copy, the contents of the form will be sent together with the associated files to the email address of the requester. In case the access is rejected, only the reason will be sent to the requester.

(Optional) Requesting a copy with Help Desk workflow

(Optional) Request Item with HelpDesk intermediary, is steered towards having your Repository Support staff act as a helpdesk that receives all incoming RequestItem requests, and then processes them. This adds the options of "Initial Reply to Requestor" to let the requestor know that their request is being worked on, and an option "Author Permission Request" which allows the helpdesk to email the author of the document, as not all documents are deposited by the author, or the author will need to be tracked down by a support staff, as DSpace might not have their current email address.

Document copy request

IF YOU ARE THE AUTHOR (OR AN AUTHOR) OF DOCUMENT "Test item with closed access document" use the buttons to answer the user's request.

This repository will propose an appropriate model reply, which you may edit.

Initial reply to requester

Author permission request

Send copy

Don't send copy

Initial Reply to Requester

Document copy request

Initial communication with the requester, to let them know their request is being processed.

To:**Subject:****Message:**

Dear Jane Doe,

Thanks for your interest! Since the author owns the copyright for this work, I will contact the author and ask permission to send you a copy. I'll let you know as soon as I hear from the author.

Thanks!
Help Desk <dspace-help@myu.edu>

Author permission request, includes information about the original request (requester name, requester email, requester's reason for requesting). The author/submitter's name and email address will be pre-populated in the form from the submitter, but the email address and author name are editable, as the submitter's of content to DSpace aren't always the author.

Document copy request

This is the text to be sent to the applicant (together with the document).

To:

Subject:

Message:

Dear ,

DSpace has an archived digitized copy of your work, "Test item with closed access document". The digital version is here:

<http://hdl.handle.net/123456789/1999>

It is not accessible to the public. We have received a request for access to work from someone who does not have access to it. Since you own the copyright to your work, we need your permission to grant the requester access. If you grant access, I will email the requester a digital copy.

Please let me know whether you approve or deny this request.

Requested by: Jane Doe <jane@dspace.org>

Message: I would like access to this document as I am doing research in this field at DSpace University.

Thank you!

Help Desk <dSPACE-help@myu.edu>

Email templates

Most of the email templates used by Request a Copy are treated just like other email templates in DSpace. The templates can be found in the `/config/emails` directory and can be altered just by changing the contents and restarting tomcat.

request_item.admin	template for the message that will be sent to the administrator of the repository, after the original submitter requests to have the permissions changed for this item.
---------------------------	---

request_item.author	template for the message that will be sent to the original submitter of an item with the request for copy.
----------------------------	--

The templates for emails that the requester receives, that could have been customized by the approver in the aforementioned dialog are not managed as separate email template files. These defaults are stored in the Messages.properties file under the keys

itemRequest.response.body.approve	Default message for informing the requester of the approval
itemRequest.response.body.reject	Default message for informing the requester of the rejection
itemRequest.response.body.contactAuthor	Default message for the helpdesk to contact the author
itemRequest.response.body.contactRequester	Default message for the helpdesk to contact the requester

Configuration parameters

Request a copy is enabled in DSpace 4 by default. Only two configuration parameters in dspace.cfg relate to Request a Copy:

Property:	<code>request.item.type</code>
Example Value	<code>request.item.type = all</code>
Informational Note	This parameter manages who can file a request for an item. The parameter is optional. When it is empty or commented out, request a copy is disabled across the entire repository. When set to all , any user can file a request for a copy. When set to logged , only registered users can file a request for copy.
Property:	<code>mail.helpdesk</code>
Example Value	<code>mail.helpdesk = foo@bar.com</code>
Informational Note	In JSPUI, the email address assigned to this parameter will receive the emails both for granting or rejecting request a copy requests, as well as requests to change item policies. In XMLUI, the parameter will also receive these requests to change item policies. However, the actual requests for copy in XMLUI will initially be directed at the email address of the original submitter. When this email address can not be retrieved, the address in mail.helpdesk will be used as a fallback. This parameter is optional. If it is empty or commented out, it will default to mail.admin.

Property:	request.item.helpdesk.override
Example Value	request.item.helpdesk.override = true
Informational Note	Should all Request Copy emails go to the helpdesk instead of the item submitter? Default is false, which sends Item Requests to the item submitter.

Spring Configuration Properties

The process that DSpace will determine who the recipient of the Item Request is configurable by Spring.

config/spring/api/requestitem.xml

By default the RequestItemMetadataStrategy is enabled, but falls back to the Item Submitter eperson's name and email. You can configure the RequestItemMetadataStrategy to load the author's name and email address if you set that information into an item metadata field.

```
<bean class="org.dspace.app.requestitem.RequestItemMetadataStrategy"
  id="org.dspace.app.requestitem.RequestItemAuthorExtractor">
  <!--
  Uncomment these properties if you want lookup in metadata the email and the name o

  If you don't configure that or if the requested item doesn't have these metadata t

  <property name="emailMetadata" value="schema.element.qualifier" />
  <property name="fullNameMatadata" value="schema.element.qualifier" />

  -->
</bean>
```

Otherwise, if you wish to use the Helpdesk Strategy, you have to comment out the other RequestItemAuthorExtractor, and uncomment:

```
<bean
class=
"org.dspace.app.requestitem.RequestItemHelpdeskStrategy"
id=
```

```
"org.dspace.app.requestitem.RequestItemAuthorExtractor"
```

```
></bean>
```

4.2 Exporting Content and Metadata

General top level page to group all DSpace facilities for exporting content and metadata.

4.2.1 OAI

OAI Interfaces

- 1 [OAI-PMH Server](#)
 - 1.1 [OAI-PMH Server Activation](#)
- 2 [OAI-PMH / OAI-ORE Harvester \(Client\)](#)
 - 2.1 [Harvesting from another DSpace](#)
 - 2.2 [OAI-PMH / OAI-ORE Harvester Configuration](#)

OAI-PMH Server

In the following sections and subpages, you will learn how to configure OAI-PMH server and activate additional OAI-PMH crosswalks. The user is also referred to [OAI-PMH Data Provider](#) for greater depth details of the program.

The OAI-PMH Interface may be used by other systems to harvest metadata records from your DSpace.

OAI-PMH Server Activation

To enable DSpace's OAI-PMH server, just make sure the `[dspace]/webapps/oai/` web application is available from your Servlet Container (usually Tomcat).

- You can test that it is working by sending a request to: `http://[full-URL-to-OAI-PMH]/request?verb=Identify`
- The response should look similar to the response from the DSpace Demo Server: <http://demo.dspace.org/oai/request?verb=Identify>

If you're using a recent browser, you should see a HTML page describing your repository. What you're getting from the server is in fact an XML file with a link to an XSLT stylesheet that renders this HTML in your browser (client-side). Any browser that cannot interpret XSLT will display pure XML. The default stylesheet is located in [

`dspace/webapps/oai/static/style.xsl` and can be changed by configuring the `stylesheet` attribute of the `Configuration` element in `[dspace]/config/crosswalks/oai/xoai.xml`.

Relevant Links

- [OAI 2.0 Server](#) - basic information needed to configure and use the OAI Server in DSpace
- [OAI-PMH Data Provider 2.0 \(Internals\)](#) - information on how it's implemented
- <http://www.openarchives.org/pmh/> - information on the OAI-PMH protocol and its usage (not DSpace-specific)

OAI-PMH / OAI-ORE Harvester (Client)

This section describes the parameters used in configuring the OAI-ORE / OAI-ORE harvester (for XMLUI only). This harvester can be used to harvest content (bitstreams and metadata) into DSpace from an external OAI-PMH or OAI-ORE server.

Relevant Links

For information on activating & using the OAI-PMH / OAI-ORE Harvester to harvest content into your DSpace, see [Harvesting Items from XMLUI via OAI-ORE or OAI-PMH](#)

Harvesting from another DSpace

If you are harvesting content (bitstreams and metadata) **from** an external DSpace installation via OAI-PMH & OAI-ORE, you first should verify that the external DSpace installation allows for OAI-ORE harvesting.

First, that external DSpace must be running both the OAI-PMH interface and the XMLUI interface to support harvesting content from it via OAI-ORE.

You can verify that OAI-ORE harvesting option is enabled by following these steps:

1. First, check to see if the external DSpace reports that it will support harvesting ORE via the OAI-PMH interface. Send the following request to the DSpace's OAI-PMH interface: `http://[full-URL-to-OAI-PMH]/request?verb=ListRecords&metadataPrefix=ore`
 - The response should be an XML document containing ORE, similar to the response from the DSpace Demo Server: <http://demo.dspace.org/oai/request?verb=ListRecords&metadataPrefix=ore>
2. Next, you can verify that the XMLUI interface supports OAI-ORE (it should, as long as it's a current version of DSpace). First, find a valid Item Handle. Then, send the following request to the DSpace's XMLUI interface: `http://[full-URL-to-XMLUI]/metadata/handle/[item-handle]/ore.xml`

- The response should be an OAI-ORE (XML) document which describes that specific Item. It should look similar to the response from the DSpace Demo Server: <http://demo.dspace.org/xmlui/metadata/handle/10673/3/ore.xml>

OAI-PMH / OAI-ORE Harvester Configuration

There are many possible configuration options for the OAI harvester. Most of them are technical and therefore omitted from the `dspace.cfg` file itself, using hard-coded defaults instead. However, should you wish to modify those values, including them in `oai.cfg` will override the system defaults.

Configuration File:	<code>[dspace]/config/modules/oai.cfg</code>
Property:	<code>harvester.eperson</code>
Example Value:	<code>harvester.eperson = admin@myu.edu</code>
Informational Note:	The EPerson under whose authorization automatic harvesting will be performed. This field does not have a default value and must be specified in order to use the harvest scheduling system. This will most likely be the DSpace admin account created during installation.
Property:	<code>dspace.oai.url</code>
Example Value:	<code>dspace.oai.url = \${dspace.baseUrl}/oai</code>
Informational Note:	The base url of the OAI-PMH disseminator webapp (i.e. do not include the <code>/request</code> on the end). This is necessary in order to mint URIs for ORE Resource Maps. The default value of <code>\${dspace.baseUrl}/oai</code> will work for a typical installation, but should be changed if appropriate. Please note that <code>dspace.baseUrl</code> is defined in your <code>dspace.cfg</code> configuration file.
Property:	<code>ore.authoritative.source</code>
Example Value:	<code>ore.authoritative.source = oai xmlui</code>
Informational Note:	<p>The webapp responsible for minting the URIs for ORE Resource Maps. If using <code>oai</code>, the <code>dspace.oai.url</code> config value must be set.</p> <ul style="list-style-type: none"> • When set to 'oai', all URIs in ORE Resource Maps will be relative to the OAI-PMH URL (configured by <code>dspace.oai.url</code> above) • When set to 'xmlui', all URIs in ORE Resource Maps will be relative to the DSpace Base URL (configured by <code>dspace.url</code> in the <code>dspace.cfg</code> file)

Configuration File:	<code>[dspace]/config/modules/oai.cfg</code>
	The URIs generated for ORE ReMs follow the following convention for either setting: <code>http://[base-URL]/metadata/handle/[item-handle]/ore.xml</code>
Property:	<code>harvester.autoStart</code>
Example Value:	<code>harvester.autoStart = false</code>
Informational Note:	Determines whether the harvest scheduler process starts up automatically when the XMLUI webapp is redeployed.
Property:	<code>harvester.oai.metadataformats.PluginName</code>
Example Value:	<pre>harvester.oai.metadataformats.PluginName = \ http://www.openarchives.org/OAI/2.0/oai_dc/, Simple Dublin Core</pre>
Informational Note:	This field can be repeated and serves as a link between the metadata formats supported by the local repository and those supported by the remote OAI-PMH provider. It follows the form <code>harvester.oai.metadataformats.PluginName = NamespaceURI,Optional Display Name</code> . The <code>pluginName</code> designates the metadata schemas that the harvester "knows" the local DSpace repository can support. Consequently, the <code>PluginName</code> must correspond to a previously declared ingestion crosswalk. The namespace value is used during negotiation with the remote OAI-PMH provider, matching it against a list returned by the <code>ListMetadataFormats</code> request, and resolving it to whatever <code>metadataPrefix</code> the remote provider has assigned to that namespace. Finally, the optional display name is the string that will be displayed to the user when setting up a collection for harvesting. If omitted, the <code>PluginName:NamespaceURI</code> combo will be displayed instead.
Property:	<code>harvester.oai.oreSerializationFormat.OREPrefix</code>
Example Value:	<pre>harvester.oai.oreSerializationFormat.OREPrefix = \ http://www.w3.org/2005/Atom</pre>
Informational Note:	

Configuration File:	<code>[dspace]/config/modules/oai.cfg</code>
	This field works in much the same way as <code>harvester.oai.metadataformats.PluginName</code> . The <code>OREPrefix</code> must correspond to a declared ingestion crosswalk, while the Namespace must be supported by the target OAI-PMH provider when harvesting content.
Property:	<code>harvester.timePadding</code>
Example Value:	<code>harvester.timePadding = 120</code>
Informational Note:	Amount of time subtracted from the from argument of the PMH request to account for the time taken to negotiate a connection. Measured in seconds. Default value is 120.
Property:	<code>harvester.harvestFrequency</code>
Example Value:	<code>harvester.harvestFrequency = 720</code>
Informational Note:	How frequently the harvest scheduler checks the remote provider for updates. Should always be longer than <i>timePadding</i> . Measured in minutes. Default value is 720.
Property:	<code>harvester.minHeartbeat</code>
Example Value:	<code>harvester.minHeartbeat = 30</code>
Informational Note:	The heartbeat is the frequency at which the harvest scheduler queries the local database to determine if any collections are due for a harvest cycle (based on the <i>harvestFrequency</i>) value. The scheduler is optimized to then sleep until the next collection is actually ready to be harvested. The <i>minHeartbeat</i> and <i>maxHeartbeat</i> are the lower and upper bounds on this timeframe. Measured in seconds. Default value is 30.
Property:	<code>harvester.maxHeartbeat</code>
Example Value:	<code>harvester.maxHeartbeat = 3600</code>
Informational Note:	The heartbeat is the frequency at which the harvest scheduler queries the local database to determine if any collections are due for a harvest cycle (based on the <i>harvestFrequency</i>) value. The scheduler is optimized to then sleep until the next collection is actually ready to be harvested. The <i>minHeartbeat</i> and <i>maxHeartbeat</i> are the lower and upper bounds on this timeframe. Measured in seconds. Default value is 3600 (1 hour).
Property:	<code>harvester.maxThreads</code>

Configuration File:	<code>[dspace]/config/modules/oai.cfg</code>
Example Value:	<code>harvester.maxThreads = 3</code>
Informational Note:	How many harvest process threads the scheduler can spool up at once. Default value is 3.
Property:	<code>harvester.threadTimeout</code>
Example Value:	<code>harvester.threadTimeout = 24</code>
Informational Note:	How much time passes before a harvest thread is terminated. The termination process waits for the current item to complete ingest and saves progress made up to that point. Measured in hours. Default value is 24.
Property:	<code>harvester.unknownField</code>
Example Value:	<code>harvester.unknownField = fail add ignore</code>
Informational Note:	You have three (3) choices. When a harvest process completes for a single item and it has been passed through ingestion crosswalks for ORE and its chosen descriptive metadata format, it might end up with DIM values that have not been defined in the local repository. This setting determines what should be done in the case where those DIM values belong to an already declared schema. <i>Fail</i> will terminate the harvesting task and generate an error. Ignore will quietly omit the unknown fields. Add will add the missing field to the local repository's metadata registry. Default value: fail .
Property:	<code>harvester.unknownSchema</code>
Example Value:	<code>harvester.unknownSchema = fail add ignore</code>
Informational Note:	When a harvest process completes for a single item and it has been passed through ingestion crosswalks for ORE and its chosen descriptive metadata format, it might end up with DIM values that have not been defined in the local repository. This setting determines what should be done in the case where those DIM values belong to an unknown schema. Fail will terminate the harvesting task and generate an error. Ignore will quietly omit the unknown fields. Add will add the missing schema to the local repository's metadata registry, using the schema name as the prefix and "unknown" as the namespace. Default value: fail .
Property:	<code>harvester.acceptedHandleServer</code>

Configuration File:	<code>[dspace]/config/modules/oai.cfg</code>
Example Value:	<pre> harvester.acceptedHandleServer = \ hdl.handle.net, handle.test.edu </pre>
Informational Note:	A harvest process will attempt to scan the metadata of the incoming items (identifier.uri field, to be exact) to see if it looks like a handle. If so, it matches the pattern against the values of this parameter. If there is a match the new item is assigned the handle from the metadata value instead of minting a new one. Default value: <i>hdl.handle.net</i> .
Property:	<code>harvester.rejectedHandlePrefix</code>
Example Value:	<code>harvester.rejectedHandlePrefix = 123456789, myeduHandle</code>
Informational Note:	Pattern to reject as an invalid handle prefix (known test string, for example) when attempting to find the handle of harvested items. If there is a match with this config parameter, a new handle will be minted instead. Default value: <i>123456789</i> .

OAI 2.0 Server

- 1 [Introduction](#)
 - 1.1 [What is OAI 2.0?](#)
 - 1.2 [Why OAI 2.0?](#)
 - 1.3 [Concepts \(XOAI Core Library\)](#)
- 2 [OAI 2.0](#)
 - 2.1 [Using Solr](#)
 - 2.1.1 [OAI Manager \(Solr Data Source\)](#)
 - 2.1.2 [Scheduled Tasks](#)
 - 2.2 [Using Database](#)
 - 2.2.1 [OAI Manager \(Database Data Source\)](#)
 - 2.2.2 [Scheduled Tasks](#)
 - 2.3 [Client-side stylesheet](#)
 - 2.4 [Metadata Formats](#)
 - 2.5 [Encoding problems](#)
- 3 [Configuration](#)
 - 3.1 [Basic Configuration](#)
 - 3.2 [Advanced Configuration](#)
 - 3.2.1 [General options](#)
 - 3.2.2 [Add/Remove Metadata Formats](#)

- 3.2.3 [Add/Remove Metadata Fields](#)
- 4 [Driver/OpenAIRE compliance](#)
 - 4.1 [Driver Compliance](#)
 - 4.2 [OpenAIRE compliance](#)

Introduction

[Open Archives Initiative Protocol for Metadata Harvesting](#) is a low-barrier mechanism for repository interoperability. Data Providers are repositories that expose structured metadata via OAI-PMH. Service Providers then make OAI-PMH service requests to harvest that metadata. OAI-PMH is a set of six verbs or services that are invoked within HTTP.

What is OAI 2.0?

OAI 2.0 is a Java implementation of an OAI-PMH data provider interface developed by [Lyncode](#) that uses XOAI, an OAI-PMH Java Library.

Why OAI 2.0?

Projects like [OpenAIRE](#), [Driver](#) have specific metadata requirements (to the published content through the OAI-PMH interface). As the OAI-PMH protocol doesn't establish any frame to these specifics, OAI 2.0 can, in a simple way, have more than one instance of an OAI interface (feature provided by the XOAI core library) so one could define an interface for each project. That is the main purpose, although, OAI 2.0 allows much more than that.

Concepts (XOAI Core Library)

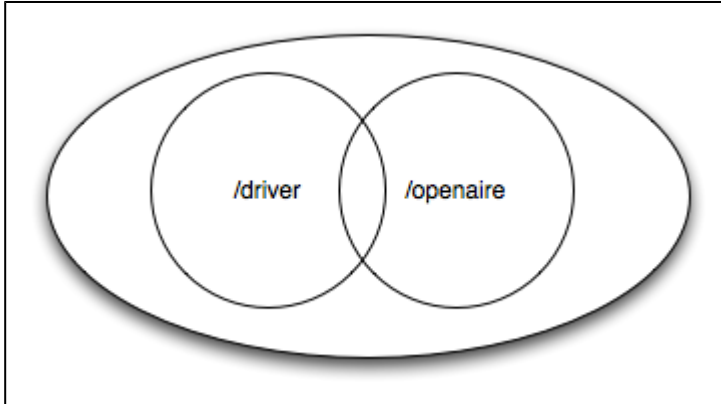
To understand how XOAI works, one must understand the concept of Filter, Transformer and Context. With a Filter it is possible to select information from the data source. A Transformer allows one to make some changes in the metadata before showing it in the OAI interface. XOAI also adds a new concept to the OAI-PMH basic specification, the concept of context. A context is identified in the URL:

```
http://www.example.com/oai/<context>
```

Contexts could be seen as virtual distinct OAI interfaces, so with this one could have things like:

- <http://www.example.com/oai/request>
- <http://www.example.com/oai/driver>
- <http://www.example.com/oai/openaire>

With this ingredients it is possible to build a robust solution that fulfills all requirements of *Driver*, *OpenAIRE* and also other project-specific requirements. As shown in Figure 1, with contexts one could select a subset of all available items in the data source. So when entering the *OpenAIRE* context, all OAI-PMH request will be restricted to that subset of items.



At this stage, contexts could be seen as sets (also defined in the basic OAI-PMH protocol). The magic of XOAI happens when one needs specific metadata format to be shown in each context. Metadata requirements by *Driver* slightly differ from the *OpenAIRE* ones. So for each context one must define its specific transformer. So, contexts could be seen as an extension to the concept of sets.

To implement an OAI interface from the XOAI core library, one just needs to implement the datasource interface.

OAI 2.0

OAI 2.0 is a separate webapp which is a complete substitute for the old "oai" webapp. OAI 2.0 has a configurable data source, by default it will not query the DSpace SQL database at the time of the OAI-PMH request. Instead, it keeps the required metadata in its Solr index (currently in a separate "oai" Solr core) and serves it from there. It's also possible to set OAI 2.0 to only use the database for querying purposes if necessary, but this decreases performance significantly. Furthermore, it caches the requests, so doing the same query repeatedly is very fast. In addition to that it also compiles DSpace items to make uncached responses much faster.

Details about OAI 2.0 internals can be found [here](#).

Using Solr

OAI 2.0 uses the Solr data source by default.

The Solr index can be updated at your convenience, depending on how fresh you need the information to be. Typically, the administrator sets up a nightly cron job to update the Solr index from the SQL database.

OAI Manager (Solr Data Source)

OAI manager is a utility that allows one to do certain administrative operations with OAI. You can call it from the command line using the dspace launcher:

Syntax

```
[dspace]/bin/dspace oai <action> [parameters]
```

Actions

- import *Imports DSpace items into OAI Solr index (also cleans OAI cache)*

- clean-cache *Cleans the OAI cache*

Parameters

- -o *Optimize index after indexing*
- -c *Clears the Solr index before indexing (it will import all items again)*
- -v *Verbose output*
- -h *Shows an help text*

Scheduled Tasks

In order to refresh the OAI Solr index, it is required to run the `[dspace]/bin/dspace oai import` command periodically. You can add the following task to your crontab:

```
0 3 * * * [dspace]/bin/dspace oai import
```

Note that `[dspace]` should be replaced by the correct value, that is, the value defined in `dspace.cfg` parameter `dspace.dir`.

Using Database

OAI 2.0 could also work using the database for querying. To configure that one must change the `[dspace]/config/modules/oaai.cfg` file, specifically the "storage" parameter, setting it to "database". This decreases performance significantly and likely has no other benefits than leaving out Solr as a dependency.

OAI Manager (Database Data Source)

OAI manager is a utility that allows one to do some administrative operations with OAI. You can call it from the command line using the dspace launcher:

Syntax

```
[dspace]/bin/dspace oai <action> [parameters]
```

Actions

- clean-cache *Cleans the OAI cache*
- compile-items *Compiles DSpace items*
- erase-compiled-items *Erases all DSpace compiled items*

Parameters

- -v *Verbose output*
- -h *Shows an help text*

Scheduled Tasks

In order to refresh the OAI cache and compile DSpace items (for fast responses), it is required to run the `[dspace]/bin/dspace xoai compile-items` command periodically. You can add the following task to your crontab:

```
0 3 * * * [dspace]/bin/dspace oai compile-items
```

Note that `[dspace]` should be replaced by the correct value, that is, the value defined in `dspace.cfg` parameter `dspace.dir`.

Client-side stylesheet

The OAI-PMH response is an XML file. While OAI-PMH is primarily used by harvesting tools and usually not directly by humans, sometimes it can be useful to look at the OAI-PMH requests directly - usually when setting it up for the first time or to verify any changes you make. For these cases, XOAI provides an XSLT stylesheet to transform the response XML to a nice looking, human-readable and interactive HTML. The stylesheet is linked from the XML response and the transformation takes place in the user's browser (this requires a recent browser, older browsers will only display the XML directly). Most automated tools are interested only in the XML file itself and will not perform the transformation. If you want, you can change which stylesheet will be used by placing it into the `[dspace]/webapps/oai/static` directory (or into the `[dspace-src]/dspace-xoai/dspace-xoai-webapp/src/main/webapp/static` after which you have to rebuild DSpace), modifying the "stylesheet" attribute of the "Configuration" element in `[dspace]/config/crosswalks/oai/xoai.xml` and restarting your servlet container.

Metadata Formats

By default OAI 2.0 provides 12 metadata formats within the `/request` context:

1. OAI_DC
2. DIDL
3. DIM
4. ETDMS
5. METS
6. MODS
7. OAI-ORE
8. QDC
9. RDF
10. MARC
11. UKETD_DC
12. XOAI

At `/driver` context it provides:

1. OAI_DC
2. DIDL
3. METS

And at /openaire context it provides:

1. OAI_DC
2. METS

Encoding problems

There are two main potential sources of encoding problems:

- a) The servlet connector port has to use the correct encoding. E.g. for Tomcat, this would be `<Connector port="8080" ... URIEncoding="UTF-8" />`, where the port attribute specifies port of the connector that DSpace is configured to access Solr on (this is usually 8080, 80 or in case of AJP 8009).
- b) System locale of the dspace command line script that is used to do the oai import. Make sure the user account launching the script (usually from cron) has the correct locale set (e.g. en_US.UTF-8). Also make sure the locale is actually present on your system.

Configuration

Basic Configuration

Configuration File:	<code>[dspace]/config/modules/oai.cfg</code>
Property:	<code>storage</code>
Example Value:	<code>storage = solr</code>
Information Note:	This allows to choose the OAI data source between solr and database
Property:	<code>solr.url</code>
Example Value:	<code>solr.url = \${default.solr.server}/oai</code>
Informational Note:	Solr Server location
Property:	<code>identifier.prefix</code>
Example Value:	<code>identifier.prefix = \${dspace.hostname}</code>
Informational Note:	OAI persistent identifier prefix. Format - oai:PREFIX:HANDLE

Configuration File:	<code>[dspace]/config/modules/oai.cfg</code>
Property:	<code>config.dir</code>
Example Value:	<code>config.dir = \${dspace.dir}/config/crosswalks/oai</code>
Informational Note:	Configuration directory, used by XOAI (core library). Contains xoi.xml, metadata format XSLTs and transformer XSLTs.
Property:	<code>cache.dir</code>
Example Value:	<code>cache.dir = \${dspace.dir}/var/oai</code>
Informational Note:	Directory to store runtime generated files (for caching purposes).

Advanced Configuration

OAI 2.0 allows you to configure following advanced options:

- Contexts
- Transformers
- Metadata Formats
- Filters
- Sets

It's an XML file commonly located at: `[dspace]/config/crosswalks/oai/xoi.xml`

General options

These options influence the OAI interface globally. "per page" means per request, next page (if there is one) can be requested using `resumptionToken` provided in current page.

- `indentation` [boolean] - whether the output XML should be indented to make it human-readable
- `maxListIdentifiersSize` [integer] - how many identifiers to show per page (`verb=ListIdentifiers`)
- `maxListRecordsSize` [integer] - how many records to show per page (`verb=ListRecords`)
- `maxListSetsSize` [integer] - how many sets to show per page (`verb=ListSets`)
- `stylesheet` [relative file path] - an xsl stylesheet used by client's web browser to transform the output XML into human-readable HTML

Their location and default values are shown in the following fragment:

```
<Configuration xmlns="http://www.lyncode.com/XOAIConfiguration"
  indentation="false"
  maxListIdentifiersSize="100"
```

```
maxListRecordsSize="100"  
maxListSetsSize="100"  
stylesheet="static/style.xsl">
```

Add/Remove Metadata Formats

Each context could have its own metadata formats. So to add/remove metadata formats to/from it, just need add /remove its reference within `xoai.xml`, for example, imagine one need to remove the XOAI schema from:

```
<Context baseurl="request">  
  <Format refid="oaidc" />  
  <Format refid="mets" />  
  <Format refid="xoai" />  
  <Format refid="didl" />  
  <Format refid="dim" />  
  <Format refid="ore" />  
  <Format refid="rdf" />  
  <Format refid="etdms" />  
  <Format refid="mods" />  
  <Format refid="qdc" />  
  <Format refid="marc" />  
  <Format refid="uketd_dc" />  
</Context>
```

Then one would have:

```
<Context baseurl="request">  
  <Format refid="oaidc" />  
  <Format refid="mets" />  
  <Format refid="didl" />  
  <Format refid="dim" />  
  <Format refid="ore" />  
  <Format refid="rdf" />  
  <Format refid="etdms" />  
  <Format refid="mods" />  
  <Format refid="qdc" />  
  <Format refid="marc" />  
  <Format refid="uketd_dc" />  
</Context>
```

It is also possible to create new metadata format by creating a specific XSLT for it. All already defined XSLT for DSpace can be found in the **[dspace]/config/crosswalks/oai/metadataFormats** directory. So after producing a new one, add the following information (location marked using brackets) inside the `<Formats>` element in `[dspace]/config/crosswalks/oai/xoai.xml`:

```
<Format id="[IDENTIFIER]">  
  <Prefix>[PREFIX]</Prefix>  
  <XSLT>metadataFormats/[XSLT]</XSLT>
```

```

<Namespace>[NAMESPACE]</Namespace>
<SchemaLocation>[SCHEMA_LOCATION]</SchemaLocation>
</Format>
  
```

where:

Parameter	Description
IDENTIFIER	The identifier used within context configurations to reference this specific format, must be unique within all Metadata Formats available.
PREFIX	The prefix used in OAI interface (metadataPrefix=PREFIX).
XSLT	The name of the XSLT file within [dspace]/config/crosswalks/oai/metadataFormats directory
NAMESPACE	XML Default Namespace of the created Schema
SCHEMA_LOCATION	URI Location of the XSD of the created Schema

NOTE: Changes in [dspace]/config/crosswalks/oai/xoai.xml requires reloading/restarting the servlet container.

Add/Remove Metadata Fields

The internal DSpace fields (Dublin Core) are exposed in the internal XOAI format (xml). All other metadata formats exposed via OAI are mapped from this XOAI format using XSLT (xoai.xsl itself is just an identity transformation). These XSLT stylesheets are found in the **[dspace]/config/crosswalks/oai/metadataFormats** directory. So e.g. oai_dc.xsl is a transformation from the XOAI format to the oai_dc format (unqualified Dublin Core).

Therefore exposing any DSpace metadata field in any OAI format is just a matter of modifying the corresponding output format stylesheet (This assumes the general knowledge of how XSLT works. For a tutorial, see e.g. <http://www.w3schools.com/xsl/>).

For example, if you have a DC field "local.note.librarian" that you want to expose in oai_dc as <dc:note> (please note that this is not a valid DC field and thus breaks compatibility), then edit oai_dc.xsl and add the following lines just above the closing tag </oai_dc:dc>:

```

<xsl:for-each select="doc:metadata/doc:element[@name='local']/doc:element[@name='note']/doc:element
/doc:element/doc:field[@name='librarian']">
  <dc:note><xsl:value-of select="." /></dc:note>
</xsl:for-each>
  
```

If you need to add/remove metadata fields, you're changing the output format. Therefore it is recommended to [create a new metadata format](#) as a copy of the one you want to modify. This way the old format will remain available along with the new one and any upgrades to the original format during DSpace upgrades will not

overwrite your customizations. If you need the format to have the same name as the original format (e.g. the default `oai_dc` format), you can create a new [context](#) in `xoai.xml` containing your modified format with the original name, which will be available as `/oai/context-name`.

NOTE: Please, keep in mind that the OAI provider caches the transformed output, so you have to run `[dSPACE]/bin/dSPACE oai clean-cache` after any `.xsl` modification and reload the OAI page for the changes to take effect. When adding/removing metadata formats, making changes in `[dSPACE]/config/crosswalks/oai/xoai.xml` requires reloading/restarting the servlet container.

Driver/OpenAIRE compliance

The default OAI 2.0 installation provides two new contexts. They are:

- [Driver](#) context, which only exposes Driver compliant items;
- [OpenAIRE](#) context, which only exposes OpenAIRE compliant items;

However, in order to be exposed DSpace items must be compliant with Driver/OpenAIRE guide-lines.

Driver Compliance

DRIVER Guidelines for Repository Managers and Administrators on how to expose digital scientific resources using OAI-PMH and Dublin Core Metadata, creating interoperability by homogenizing the repository output. The set driver of OAI-PMH is based on [DRIVER Guidelines 2.0](#) (see the [English version](#) of the document)

This set is used to expose items of the repository that are available for open access. It's not necessary for all the items of the repository to be available for open access.

What specific metadata values are expected?

To have items in this set, you must configure your `input-forms.xml` file in order to comply with the DRIVER Guidelines:

- Must have a publication date - [dc.date.issued](#) (already configured in DSpace items)
- [dc.language](#) must use [ISO639-3](#)
- the value of [dc.type](#) must be one of the 16 options of the guidelines (see page 68)

How do you easily add those metadata values?

As DRIVER guidelines use Dublin Core, all the needed items are already registered in DSpace. You just need to configure the deposit process.

OpenAIRE compliance

The OpenAIRE Guidelines 2.0 provide the OpenAIRE compatibility to repositories and aggregators. By implementing these Guidelines, repository managers are facilitating the authors who deposit their publications in the repository in complying with the EC Open Access requirements. For developers of repository platforms, the Guidelines provide guidance to add supportive functionalities for authors of EC-funded research in future versions.

The name of the set in OAI-PMH is "ec_fundedresources" and will expose the items of the repository that comply with these guidelines. These guidelines are based on top of DRIVER guidelines. See [version 2.0 of the Guidelines](#).

See the [Application Profile of OpenAIRE](#).

What specific metadata values are expected?

These are the OpenAIRE metadata values only, to check these and driver metadata values check page 11 of the OpenAIRE guidelines 2.0.

- [dc:relation](#) with the project ID (see p.8)
- [dc:rights](#) with the access rights information from vocabulary (possible values [here](#))

Optionally:

- [dc:date](#) with the embargo end date (recommended for embargoed items)

```
<dc:date>info:eu-repo/date/embargoEnd/2011-05-12<dc:date>
```

How do you easily add those metadata values?

- Have a [dc:relation](#) field in `input-forms.xml` with a list of the projects. You can also use the [OpenAIRE Authority Control Addon](#) to facilitate the process of finding the project.
- Just use a combo-box for [dc:rights](#) to input the 4 options:
 - `info:eu-repo/semantics/closedAccess`
 - `info:eu-repo/semantics/embargoedAccess`
 - `info:eu-repo/semantics/restrictedAccess`
 - `info:eu-repo/semantics/openAccess`
- Use an input-box for [dc:date](#) to insert the embargo end date



Relevant Links

- OAI 2.0 is a standard part of DSpace 3.0
- Download & Install OAI 2.0 for DSpace 1.8.x: <http://www.lyncode.com/dspace/addons/xoai/>

OAI-PMH Data Provider 2.0 (Internals)

- 1 [OAI-PMH Data Provider 2.0 \(Internals\)](#)
 - 1.1 [Sets](#)
 - 1.2 [Unique Identifier](#)

- [1.3 Access control](#)
- [1.4 Modification Date \(OAI Date Stamp\)](#)
- [1.5 "About" Information](#)
- [1.6 Deletions](#)
- [1.7 Flow Control \(Resumption Tokens\)](#)

OAI-PMH Data Provider 2.0 (Internals)

The DSpace platform supports the [Open Archives Initiative Protocol for Metadata Harvesting](#) (OAI-PMH) version 2.0 as a data provider. This is accomplished using the [XOAI](#) OAI-PMH Java Toolkit.

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. This "webapp" is deployed to receive and respond to OAI-PMH requests via HTTP. In a typical configuration, this is deployed at *oai*, containing request, driver and openaire contexts, 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
```

But one could also provide the Driver or OpenAIRE contexts:

```
http://dspace.myu.edu/oai/driver  
http://dspace.myu.edu/oai/openaire
```

It is this URL that should be registered with www.openarchives.org.

DSpace provides implementations of the XOAI data sources interfaces.

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 and communities as sets.

Each community and collection has a corresponding OAI set, discoverable by harvesters via the ListSets verb. The setSpec is based on the community/collection handle, with the "/" converted to underscore to form a legal setSpec. The setSpec is prefixed by "com_" or "col_" for communities and collections, respectively (this is a change in set names in DSpace 3.0 / OAI 2.0). For example:

```
col_1721.1_1234
```

Naturally enough, the community/collection name is also the name of the corresponding set.

Unique Identifier

Every item in OAI-PMH data repository must have a 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:PREFIX: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 `identifier.prefix` at `[dspace]/config/modules/oai.cfg` file.

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. But, exposed data could be changed by changing the XSLT defined at `[dspace]/config/crosswalks/oai/metadataFormats`.

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, but XOAI core library allows its definition. This requires to dive into code and perform some changes.

Deletions

DSpace keeps track of deletions (withdrawals). These are exposed via OAI, which has a specific mechanism for dealing with this. Since DSpace keeps a permanent record of withdrawn items, in the OAI-PMH sense DSpace supports deletions "persistently". This is as opposed to "transient" deletion support, which would mean that deleted records are forgotten after a time.

Once an item has been withdrawn, OAI-PMH harvests of the date range in which the withdrawal occurred will find the "deleted" record header. Harvests of a date range prior to the withdrawal will *not* find the record, despite the fact that the record did exist at that time.

As an example of this, consider an item that was created on 2002-05-02 and withdrawn on 2002-10-06. A request to harvest the month 2002-10 will yield the "record deleted" header. However, a harvest of the month 2002-05 will not yield the original record.

Note that presently, the deletion of "expunged" items is not exposed through OAI.

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", "ListIdentifiers" and "ListSets" OAI-PMH requests.

Each OAI-PMH ListRecords request will return at most 100 records (by default) but it could be configured in the `[dspace]/config/crosswalks/oai/xoai.xml` file.

When a resumption token is issued, the optional *completeListSize* and *cursor* attributes are included. OAI 2.0 resumption tokens are persistent, so *expirationDate* of the resumption token is undefined, they do not expire.

Resumption tokens contain all the state information required to continue a request and it is encoded in Base64.

4.2.2 Exchanging Content Between Repositories

- 1 [Transferring Content via Export and Import](#)
 - 1.1 [Transferring Communities, Collections, or Items using Packages](#)
- 2 [Transferring Items using Simple Archive Format](#)
- 3 [Transferring Items using OAI-ORE/OAI-PMH Harvester](#)
- 4 [Copying Items using the SWORD Client](#)

Transferring Content via Export and Import

To migrate content from one DSpace to another, you can export content from the Source DSpace and import it into the Destination DSpace.

Transferring Communities, Collections, or Items using Packages

Starting with DSpace 1.7, you can transfer any DSpace content (Communities, Collections or Items) from one DSpace to another by utilizing the [AIP Backup and Restore](#) tool. This tool allows you to export content into a series of Archival Information Packages (AIPs). These AIPs can be used to restore content (from a backup) or move/migrate content to another DSpace installation.

For more information see [AIP Backup and Restore](#).

Transferring Items using Simple Archive Format

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.

First, you should export the DSpace Item(s) into the Simple Archive Format, as detailed at: [Importing and Exporting Items via Simple Archive Format](#). Be sure to use the `--migrate` option, which removes fields that would be duplicated on import. Then import the resulting files into the other instance.

Transferring Items using OAI-ORE/OAI-PMH Harvester

If you are using the XMLUI in both DSpace instances, you may also choose to enable the [OAI-ORE Harvester](#). This OAI-ORE Harvester allows one DSpace installation to harvest Items (via OAI-ORE) from another DSpace Installation (or any other system supporting OAI-ORE). Items are harvested from a remote DSpace Collection into a local DSpace Collection. Harvesting can also be scheduled to run automatically (or by demand).

For more information see [Harvesting Items from XMLUI via OAI-ORE or OAI-PMH](#)

Copying Items using the SWORD Client

4.2.3 SWORDv1 Client

The embedded SWORD Client allows a user (currently restricted to an administrator) to copy an item to a SWORD server. This allows your DSpace installation to deposit items into another SWORD-compliant repository (including another DSpace install).

At present this functionality has only been developed for the XMLUI and is disabled by default.

- 1 [Enabling the SWORD Client](#)
- 2 [Configuring the SWORD Client](#)

Enabling the SWORD Client

To enable the SWORD Client uncomment the `SwordClient` Aspect in `[dspace]/config/xmlui.xconf` file

```
<aspect name="SwordClient" path="resource://aspects/SwordClient/" />
```

Configuring the SWORD Client

All the relevant configuration can be found in `sword-client.cfg`

Configuration File:	<code>[dspace]/config/modules/sword-client.cfg</code>
Property:	<code>targets</code>
Example value:	<pre>targets = http://localhost:8080/sword/servicedocument, \ http://client.swordapp.org/client/servicedocument, \ http://dspace.swordapp.org/sword/servicedocument, \ http://sword.eprints.org/sword-app/servicedocument, \ http://sword.intralibrary.com/IntraLibrary-Deposit/service, \ http://fedora.swordapp.org/sword-fedora/servicedocument</pre>
Informational note:	List of remote Sword servers. Used to build the drop-down list of selectable SWORD targets.
Property:	<code>file-types</code>
Example value:	<code>file-types = application/zip</code>
Informational note:	List of file types from which the user can select. If a type is not supported by the remote server it will not appear in the drop-down list.
Property:	<code>package-formats</code>
Example value:	<pre>package-formats = http://purl.org/net/sword-types/METSDSpaceSIP</pre>
Informational note:	

Configuration File:	<code>[dspace]/config/modules/sword-client.cfg</code>
	List of package formats from which the user can select. If a format is not supported by the remote server it will not appear in the drop-down list.

4.2.4 Linked (Open) Data

- [Introduction](#)
 - [Exchanging repository contents](#)
 - [Terminology](#)
- [Linked \(Open\) Data Support within DSpace](#)
 - [Architecture / Concept](#)
 - [Install a Triple Store](#)
 - [Default configuration and what you should change](#)
 - [Configuration Reference](#)
 - [\[dspace-source\]/dspace/config/modules/rdf.cfg](#)
 - [\[dspace-source\]/dspace/config/modules/rdf/constant-data-*.ttl](#)
 - [\[dspace-source\]/dspace/config/modules/rdf/metadata-rdf-mapping.ttl](#)
 - [\[dspace-source\]/dspace/config/modules/rdf/fuseki-assembler.ttl](#)
 - [Maintenance](#)

Introduction

Exchanging repository contents

Most sites on the Internet are oriented towards human consumption. While HTML may be a good format for presenting information to humans, it is not a good format to export data in a way easy for a computer to work with. Like most software for building repositories, DSpace supports [OAI-PMH](#) as an interface to expose the stored metadata. While OAI-PMH is well known in the field of repositories, it is rarely known elsewhere (e.g. [Google retired its support for OAI-PMH in 2008](#)). The Semantic Web is a generic approach to publish data on the Internet together with information about its semantics. Its application is not limited to repositories or libraries and it has a growing user base. [RDF](#) and [SPARQL](#) are W3C-released standards for publishing structured data on the web in a machine-readable way. The data stored in repositories is particularly suited for use in the Semantic Web, as the metadata are already available. It doesn't have to be generated or entered manually for publication as Linked Data. For most repositories, at least for Open Access repositories, it is quite important to share their stored content. Linked Data is a rather big chance for repositories to present their content in a way that can easily be accessed, interlinked and (re)used.

Terminology

We don't want to give a full introduction into the Semantic Web and its technologies here as this can be easily found in many places on the web. Nevertheless, we want to give a short glossary of the terms used most often in this context to make the following documentation more readable.

Semantic Web	The term "Semantic Web" refers to the part of the Internet containing Linked Data. Just like the World Wide Web, the Semantic Web is also woven together by links among the data.
Linked Data	Data in RDF, following the Linked Data Principles are called Linked Data. The Linked Data Principles describe the expected behavior of data publishers who shall ensure that the published data are easy to find, easy to retrieve, can be linked easily and link to other data as well.
Linked Open Data	Linked Open Data is Linked Data published under an open license. There is no technical difference between Linked Data and Linked Open Data (often abbreviated as LOD). It is only a question of the license used to publish it.
RDF RDF/XML Turtle N-Triples N3-Notation	RDF is an acronym for Resource Description Framework, a metadata model. Don't think of RDF as a format, as it is a model. Nevertheless, there are different formats to serialize data following RDF. RDF/XML, Turtle, N-Triples and N3-Notation are probably the most well-known formats to serialize data in RDF. While RDF/XML uses XML, Turtle, N-Triples and N3-Notation don't and they are easier for humans to read and write. When we use RDF in DSpace configuration files, we currently prefer Turtle (but the code should be able to deal with any serialization).
Triple Store	A triple store is a database to natively store data following the RDF model. Just as you have to provide a relational database for DSpace, you have to provide a Triple Store for DSpace if you want to use the LOD support.
SPARQL	The SPARQL Protocol and RDF Query Language is a family of protocols to query triple stores. Since version 1.1, SPARQL can be used to manipulate triple stores as well, to store, delete or update data in triple stores. DSpace uses SPARQL 1.1 Graph Store HTTP Protocol and SPARQL 1.1 Query Language to communicate with the Triple Store. The SPARQL 1.1 Query Language is often referred to simply as SPARQL, so expect the SPARQL 1.1 Query Language if no other specific protocol out of the SPARQL family is explicitly specified.
SPARQL endpoint	A SPARQL endpoint is a SPARQL interface of a triple store. Since SPARQL 1.1, a SPARQL endpoint can be either read-only, allowing only to query the stored data; or readable and writable, allowing to modify the stored data as well. When talking about a SPARQL endpoint without specifying which SPARQL protocol is used, an endpoint supporting SPARQL 1.1 Query Language is meant.

Linked (Open) Data Support within DSpace

Starting with DSpace 5.0, DSpace provides support for publishing stored contents in form of Linked (Open) Data

Architecture / Concept

To publish content stored in DSpace as Linked (Open) Data, the data have to be converted into RDF. The conversion into RDF has to be configurable as different DSpace instances may use different metadata schemata, different persistent identifiers (DOI, Handle, ...) and so on. Depending on the content to convert, configuration and other parameters, conversion may be time-intensive and impact performance. Content of repositories is much more often read then created, deleted or changed because the main goal of repositories is to safely store their contents. For this reason, the content stored within DSpace is converted and stored in a triple store immediately after it is created or updated. The triple store serves as a cache and provides a SPARQL endpoint to make the converted data accessible using SPARQL. The conversion is triggered automatically by the DSpace event system and can be started manually using the command line interface – both cases are documented below. There is no need to backup the triple store, as all data stored in the triple store can be recreated from the contents stored elsewhere in DSpace (in the assetstore(s) and the database). Beside the SPARQL endpoint, the data should be published as RDF serialization as well. With `dspace-rdf` DSpace offers a module that loads converted data from the triple store and provides it as an RDF serialization. It currently supports RDF/XML, Turtle and N-Triples.

Repositories use Persistent Identifiers to make content citable and to address content. Following the Linked Data Principles, DSpace uses a Persistent Identifier in the form of HTTP(S) URIs, converting a Handle to `http://hdl.handle.net/<handle>` and a DOI to `http://dx.doi.org/<doi>`. Altogether, DSpace Linked Data support spans all three Layers: the storage layer with a triple store, the business logic with classes to convert stored contents into RDF, and the application layer with a module to publish RDF serializations. Just like DSpace allows you to choose Oracle or Postgresql as the relational database, you may choose between different triple stores. The only requirements are that the triple store must support SPARQL 1.1 Query Language and SPARQL 1.1 Graph Store HTTP Protocol which DSpace uses to store, update, delete and load converted data in/out of the triple store and uses the triple store to provide the data over a SPARQL endpoint.



Store public data only in the triple store!

The triple store should contain only data that are public, because the DSpace access restrictions won't affect the SPARQL endpoint. For this reason, DSpace converts only archived, discoverable (non-private) Items, Collections and Communities which are readable for anonymous users. Please consider this while configuring and/or extending DSpace Linked Data support.

The [org.dspace.rdf.conversion](#) package contains the classes used to convert the repository content to RDF. The conversion itself is done by plugins. The [org.dspace.rdf.conversion.ConverterPlugin](#) interface is really simple, so take a look at it if you can program in Java and want to extend the conversion. The only thing important is that

plugins must only create RDF that can be made publicly available, as the triple store provides it using a sparql endpoint for which the DSpace access restrictions do not apply. Plugins converting metadata should check whether a specific metadata field needs to be protected or not (see [org.dspace.app.util.MetadataExposure](#) on how to check that). The [MetadataConverterPlugin](#) is heavily configurable (see below) and is used to convert the metadata of Items. The [StaticDSOConverterPlugin](#) can be used to add static RDF Triples (see below). The [SimpleDSORelationsConverterPlugin](#) creates links between items and collections, collections and communities, subcommunities and their parents, and between top-level communities and the information representing the repository itself.

As different repositories use different persistent identifiers to address their content, different algorithms to create URIs used within the converted data can be implemented. Currently HTTP(S) URIs of the repository (called local URIs), Handles and DOIs can be used. See the configuration part of this document for further information. If you want to add another algorithm, take a look at the [org.dspace.rdf.storage.URIGenerator](#) interface.

Install a Triple Store

In addition to a normal DSpace installation you have to install a triple store. You can use any triple store that supports SPARQL 1.1 Query Language and SPARQL 1.1 Graph Store HTTP Protocol. If you do not have one yet, you can use Apache Fuseki. Download Fuseki from its [official download page](#) and unpack the downloaded archive. The archive contains several scripts to start Fuseki. Use the start script appropriate to the OS of your choice with the options '--localhost --config=<dspace-install>/config/modules/rdf/fuseki-assembly.ttl'. Instead of changing to the directory into which you unpacked Fuseki, you may set the variable FUSEKI_HOME. If you're using Linux and bash, you unpacked Fuseki to /usr/local/jena-fuseki-1.0.1 and you installed DSpace to [dspace-install], this would look like this:

```
export FUSEKI_HOME=/usr/local/jena-fuseki-1.0.1 ; $FUSEKI_HOME/fuseki-server --localhost --config [dspace-install]/config/modules/rdf/fuseki-assembly.ttl
```

Fuseki's archive contains a script to start Fuseki automatically at startup as well.



Make Fuseki connect to localhost only, by using the argument --localhost when launching if you use the configuration provided with DSpace! The configuration contains a writeable SPARQL endpoint that allows any connection to change/delete the content of your triple store.



Use Apache mod proxy, mod rewrite or any other appropriate web server/proxy to make localhost:3030 /dspace/sparql readable from the internet. Use the address under which it is accessible as the address of your public sparql endpoint (see the property public.sparql.endpoint in the [configuration reference](#) below.).

The configuration provided within DSpace makes it store the files for the triple store under `[dspace-install]/triplestore`. Using this configuration, Fuseki provides three SPARQL endpoints: two read-only endpoints and one that can be used to change the data of the triple store. **You should not use this configuration if you let Fuseki connect to the internet directly** as it would make it possible for anyone to delete, change or add information to the triple store. The option `--localhost` tells Fuseki to listen only on the loopback device. You can use Apache `mod_proxy` or any other web or proxy server to make the read-only SPARQL endpoint accessible from the internet. With the configuration described, Fuseki listens to the port 3030 using HTTP. Using the address `http://localhost:3030/` you can connect to the Fuseki Web UI. `http://localhost:3030/dspace/data` addresses a writeable SPARQL 1.1 HTTP Graph Store Protocol endpoint, and `http://localhost:3030/dspace/get` a read-only one. Under `http://localhost:3030/dspace/sparql/` a read-only SPARQL 1.1 Query Language endpoint can be found. **The first one of these endpoints must be not accessible by the internet**, while the last one should be accessible publicly.

Default configuration and what you should change

In the file `[dspace-source]/dspace/config/dspace.cfg` you should look for the property `event.dispatcher.default.consumers` and add `rdf` there. Adding `rdf` there makes DSpace update the triple store automatically as the publicly available content of the repository changes.

As the Linked Data support of DSpace is highly configurable this section gives a short list of things you probably want to configure before using it. Below you can find more information on what is possible to configure.

In the file `[dspace-source]/dspace/config/modules/rdf.cfg` you want to configure the address of the public sparql endpoint and the address of the writable endpoint DSpace use to connect to the triple store (the properties `public.sparql.endpoint`, `storage.graphstore.endpoint`). In the same file you want to configure the URL that addresses the `dspace-rdf` module which is depending on where you deployed it (property `contextPath`) and switch content negotiation on (set property `contentNegotiation.enable = true`).

In the file `[dspace-source]/dspace/config/modules/rdf/constant-data-general.ttl` you should change the links to the Web UI of the repository and the public readable SPARQL endpoint. The URL of the public SPARQL endpoint should point to a URL that is proxied by a webserver to the Triple Store. See the section [Install a Triple Store](#) above for further information.

In the file `[dspace-source]/dspace/config/modules/rdf/constant-data-site.ttl` you may add any triples that should be added to the description of the repository itself.

If you want to change the way the metadata fields are converted, take a look into the file `[dspace-source]/dspace/config/modules/rdf/metadata-rdf-mapping.ttl`. This is also the place to add information on how to map metadata fields that you added to DSpace. There is already a quite acceptable default configuration for the metadata fields which DSpace supports out of the box. If you want to use some specific prefixes in RDF serializations that support prefixes, you have to edit `[dspace-source]/dspace/config/modules/rdf/metadata-prefixes.ttl`.

Configuration Reference

There are several configuration files to configure DSpace's LOD support. The main configuration file can be found under [dspace-source]/dspace/config/modules/rdf.cfg, all other files are positioned in the directory [dspace-source]/dspace/config/modules/rdf/. You'll have to configure where to find and how to connect to the triple store. You may configure how to generate URIs to be used within the generated Linked Data and how to convert the contents stored in DSpace into RDF. We will guide you through the configuration file by file.

[dspace-source]/dspace/config/modules/rdf.cfg

The following properties configure the StaticDSOConverterPlugin.	
Property:	public.sparql.endpoint
Example Value:	public.sparql.endpoint = http://\${dspace.baseUrl}/sparql
Informational Note:	Address of the read-only public SPARQL endpoint supporting SPARQL 1.1 Query Language.
Property:	URIGenerator
Example Value:	URIGenerator = org.dspace.rdf.storage.LocalURIGenerator
Informational Note:	The name of the class that generates the URIs to be used within the converted data. The LocalURIGenerator generates URIs using the \${dspace.url} property. The class org.dspace.rdf.storage.HandleURIGenerator uses handles in form of HTTP URLs. It uses the property \${handle.canonical.prefix} to convert handles into HTTPS URLs. The class org.dspace.rdf.storage.DOIURIGenerator uses DOIs in the form of HTTP URLs if possible or local URIs if there are no DOIs. It uses the DOI resolver "http://dx.doi.org" to convert DOIs into HTTP URLs. The class org.dspace.rdf.storage.DOIHandleGenerator does the same but uses Handles as fallback if no DOI exists. The fallbacks are necessary as DOIs are currently used for Items only and not for Communities or Collections.
Property:	converter
Example Value:	converter = org.dspace.rdf.conversion.RDFConverterImpl
Informational Note:	This property sets the class that manages the whole conversion process. Currently there shouldn't be any need to change it.
Property:	converter.plugins
Example Value:	

The following properties configure the <code>StaticDSOConverterPlugin</code>.	
	<code>converter.plugins = org.dspace.rdf.conversion.StaticDSOConverterPlugin, \</code> <code>org.dspace.rdf.conversion.MetadataConverterPlugin, \</code> <code>org.dspace.rdf.conversion.SimpleDSORelationsConverterPlugin</code>
Informational Note:	List all plugins to be used during the conversion of DSpace contents into RDF. If you write a new conversion plugin you want to add its class name to this property.
Property:	<code>converter.DSOtypes</code>
Example Value:	<code>converter.DSOtypes = SITE, COMMUNITY, COLLECTION, ITEM</code>
Informational Note:	Define which kind of <code>DSpaceObjects</code> should be converted. Bundles and Bitstreams will be converted as part of the Item they belong to. Don't add <code>EPersons</code> here unless you really know what you are doing. All converted data is stored in the triple store that provides a publicly readable SPARQL endpoint. So all data converted into RDF is exposed publicly. Every DSO type you add here must have an HTTP URI to be referenced in the generated RDF, which is another reason not to add <code>EPersons</code> here currently.
Property:	<code>storage</code>
Example Value:	<code>storage = org.dspace.rdf.storage.RDFStorageImpl</code>
Informational Note:	Configure which class to use to store the converted data. This class handles the connection to the SPARQL endpoint. Currently there is only one implementation, so there is no need/possibility to change this property.
Property:	<code>storage.graphstore.endpoint</code>
Example Value:	<code>storage.graphstore.endpoint = http://localhost:3030/dspace/data</code>
Informational Note:	Address of a writable SPARQL 1.1 Graph Store HTTP Protocol endpoint. This address is used to create, update and delete converted data in the triple store. If you use Fuseki with the configuration provided as part of DSpace 5, you can leave this as it is. If you use another Triple Store or configure Fuseki on your own, change this property to point to a writeable SPARQL endpoint supporting the SPARQL 1.1 Graph Store HTTP Protocol.
Property:	<code>storage.graphstore.authentication</code>
Example Value:	<code>storage.graphstore.authentication = no</code>
Informational Note:	Defines whether to use HTTP Basic authentication to connect to the writable SPARQL 1.1 Graph Store HTTP Protocol endpoint.

The following properties configure the StaticDSOConverterPlugin.	
Properties:	storage.graphstore.login storage.graphstore.password
Example Values:	storage.graphstore.login = dspace storage.graphstore.password = ecapsd
Informational Note:	Credentials for the HTTP Basic authentication if it is necessary to connect to the writable SPARQL 1.1 Graph Store HTTP Protocol endpoint.
Property:	storage.sparql.endpoint
Example Value:	storage.sparql.endpoint = http://localhost:3030/dspace/sparql
Informational Note:	Besides a writable SPARQL 1.1 Graph Store HTTP Protocol endpoint, DSpace needs a SPARQL 1.1 Query Language endpoint, which can be read-only. This property allows you to set an address to be used to connect to such a SPARQL endpoint. If you leave this property empty the property <code>public.sparql.endpoint</code> will be used instead.
Properties:	storage.sparql.authentication storage.sparql.login storage.sparql.password
Example Values:	storage.sparql.authentication = yes storage.sparql.login = dspace storage.sparql.password = ecapsd
Informational Note:	As for the SPARQL 1.1 Graph Store HTTP Protocol you can configure DSpace to use HTTP Basic authentication to authenticate against the (read-only) SPARQL 1.1 Query Language endpoint.
Property:	contextPath
Example Value:	contextPath = <code>#{dspace.baseUrl}/rdf</code>
Informational Note:	The content negotiation needs to know where to refer if anyone asks for RDF serializations of content stored within DSpace. This property sets the URL where the dspace-rdf module can be reached on the internet (depending on how you deployed it).
Property:	contentNegotiation.enable
Example Value:	contentNegotiation.enable = true

The following properties configure the StaticDSOConverterPlugin.	
Informational Note:	Defines whether content negotiation should be activated. Set this true, if you use Linked Data support.
Properties:	constant.data.GENERAL constant.data.COLLECTION constant.data.COMMUNITY constant.data.ITEM constant.data.SITE
Example Values:	constant.data.GENERAL = \${dspace.dir}/config/modules/rdf/constant-data-general.ttl constant.data.COLLECTION = \${dspace.dir}/config/modules/rdf/constant-data-collection.ttl constant.data.COMMUNITY = \${dspace.dir}/config/modules/rdf/constant-data-community.ttl constant.data.ITEM = \${dspace.dir}/config/modules/rdf/constant-data-item.ttl constant.data.SITE = \${dspace.dir}/config/modules/rdf/constant-data-site.ttl
Informational Note:	<p>These properties define files to read static data from. These data should be in RDF, and by default Turtle is used as serialization. The data in the file referenced by the property \${constant.data.GENERAL} will be included in every Entity that is converted to RDF. E.g. it can be used to point to the address of the public readable SPARQL endpoint or may contain the name of the institution running DSpace.</p> <p>The other properties define files that will be included if a DSpace Object of the specified type (collection, community, item or site) is converted. This makes it possible to add static content to every Item, every Collection, ...</p>
The following properties configure the MetadataConverterPlugin.	
Property:	metadata.mappings
Example Value:	metadata.mappings = \${dspace.dir}/config/modules/rdf/metadata-rdf-mapping.ttl
Informational Note:	Defines the file that contains the mappings for the MetadataConverterPlugin. See below the description of the configuration file [dspace-source]/dspace/config/modules/rdf/metadata-rdf-mapping.ttl.
Property:	metadata.schema
Example Value:	metadata.schema = file://\${dspace.dir}/config/modules/rdf/metadata-rdf-schema.ttl
Informational Note:	Configures the URL used to load the RDF Schema of the DSpace Metadata RDF mapping Vocabulary. Using a file:// URI makes it possible to convert DSpace content without having an internet connection. The version of the schema has to be the right one for the used code. In DSpace 5.0 we use the version 0.2.0. This Schema can be found here as well: http://

The following properties configure the StaticDSOConverterPlugin.

	digital-repositories.org/ontologies/dspace-metadata-mapping/0.2.0 . The newest version of the Schema can be found here: http://digital-repositories.org/ontologies/dspace-metadata-mapping/ .
Property:	metadata.prefixes
Example Value:	metadata.prefixes = \${dspace.dir}/config/modules/rdf/metadata-prefixes.ttl
Informational Note:	If you want to use prefixes in RDF serializations that support prefixes, you can define these prefixes in the file referenced by this property.

The following properties configure the SimpleDSORelationsConverterPlugin

Property:	simplerelations.prefixes
Example Value:	simplerelations.prefixes = \${dspace.dir}/config/modules/rdf/simple-relations-prefixes.ttl
Informational Note:	If you want to use prefixes in RDF serializations that support prefixes, you can define these prefixes in the file referenced by this property.
Property:	simplerelations.site2community
Example Value:	simplerelations.site2community = http://purl.org/dc/terms/hasPart , http://digital-repositories.org/ontologies/dspace/0.1.0#hasCommunity
Informational Note:	Defines the predicates used to link from the data representing the whole repository to the top level communities. Defining multiple predicates separated by commas will result in multiple triples.
Property:	simplerelations.community2site
Example Value:	simplerelations.community2site = http://purl.org/dc/terms/isPartOf , http://digital-repositories.org/ontologies/dspace/0.1.0#isPartOfRepository
Informational Note:	Defines the predicates used to link from the top level communities to the data representing the whole repository. Defining multiple predicates separated by commas will result in multiple triples.
Property:	simplerelations.community2subcommunity
Example Value:	simplerelations.community2subcommunity = http://purl.org/dc/terms/hasPart , http://digital-repositories.org/ontologies/dspace/0.1.0#hasSubcommunity
Informational Note:	Defines the predicates used to link from communities to their subcommunities. Defining multiple predicates separated by commas will result in multiple triples.

The following properties configure the StaticDSOConverterPlugin.

Property:	simplerelations.subcommunity2community
Example Value:	simplerelations.subcommunity2community = http://purl.org/dc/terms/isPartOf , http://digital-repositories.org/ontologies/dspace/0.1.0#isSubcommunityOf
Informational Note:	Defines the predicates used to link from subcommunities to the communities they belong to. Defining multiple predicates separated by commas will result in multiple triples.
Property:	simplerelations.community2collection
Example Value:	simplerelations.community2collection = http://purl.org/dc/terms/hasPart , http://digital-repositories.org/ontologies/dspace/0.1.0#hasCollection
Informational Note:	Defines the predicates used to link from communities to their collections. Defining multiple predicates separated by commas will result in multiple triples.
Property:	simplerelations.collection2community
Example Value:	simplerelations.collection2community = http://purl.org/dc/terms/isPartOf , http://digital-repositories.org/ontologies/dspace/0.1.0#isPartOfCommunity
Informational Note:	Defines the predicates used to link from collections to the communities they belong to. Defining multiple predicates separated by commas will result in multiple triples.
Property:	simplerelations.collection2item
Example Value:	simplerelations.collection2item = http://purl.org/dc/terms/hasPart , http://digital-repositories.org/ontologies/dspace/0.1.0#hasItem
Informational Note:	Defines the predicates used to link from collections to their items. Defining multiple predicates separated by commas will result in multiple triples.
Property:	simplerelations.item2collection
Example Value:	simplerelations.item2collection = http://purl.org/dc/terms/isPartOf , http://digital-repositories.org/ontologies/dspace/0.1.0#isPartOfCollection
Informational Note:	Defines the predicates used to link from items to the collections they belong to. Defining multiple predicates separated by commas will result in multiple triples.
Property:	simplerelations.item2bitstream
Example Value:	simplerelations.item2bitstream = http://purl.org/dc/terms/hasPart , http://digital-repositories.org/ontologies/dspace/0.1.0#hasBitstream
Informational Note:	Defines the predicates used to link from item to their bitstreams. Defining multiple predicates separated by commas will result in multiple triples.

[dspace-source]/dspace/config/modules/rdf/constant-data-*.ttl

As described in the documentation of the configuration file [dspace-source]/dspace/config/modules/rdf.cfg, the constant-data-*.ttl files can be used to add static RDF to the converted data. The data are written in Turtle, but if you change the file suffix (and the path to find the files in rdf.cfg) you can use any other RDF serialization you like to. You can use this, for example, to add a link to the public readable SPARQL endpoint, add a link to the repository homepage, or add a triple to every community or collection defining it as an entity of a specific type like a bibo:collection. The content of the file [dspace-source]/dspace/config/modules/rdf/constant-data-general.ttl will be added to every DSpaceObject that is converted. The content of the file [dspace-source]/dspace/config/modules/rdf/constant-data-community.ttl to every community, the content of the file [dspace-source]/dspace/config/modules/rdf/constant-data-collection.ttl to every collection and the content of the file [dspace-source]/dspace/config/modules/rdf/constant-data-item.ttl to every Item. You can use the file [dspace-source]/dspace/config/modules/rdf/constant-data-site.ttl to specify data representing the whole repository.

[dspace-source]/dspace/config/modules/rdf/metadata-rdf-mapping.ttl

This file should contain several metadata mappings. A metadata mapping defines how to map a specific metadata field within DSpace to a triple that will be added to the converted data. The MetadataConverterPlugin uses these metadata mappings to convert the metadata of a item into RDF. For every metadata field and value it looks if any of the specified mappings matches. If one does, the plugin creates the specified triple and adds it to the converted data. In the file you'll find a lot of examples on how to define such a mapping.

For every mapping a metadata field name has to be specified, e.g. dc.title, dc.identifier.uri. In addition you can specify a condition that is matched against the field's value. The condition is specified as a regular expression (using the syntax of the java class java.util.regex.Pattern). If a condition is defined, the mapping will be used only on fields those values which are matched by the regex defined as condition.

The triple to create by a mapping is specified using reified RDF statements. The [DSpace Metadata RDF Mapping Vocabulary](#) defines some placeholders that can be used. The most important placeholder is dm:DSpaceObjectIRI which is replaced by the URI used to identify the entity being converted to RDF. That means if a specific Item is converted the URI used to address this Item in RDF will be used instead of dm:DSpaceObjectIRI. There are three placeholders that allow reuse of the value of a meta data field. dm:DSpaceValue will be replace by the value as it is. dm:LiteralGenerator allows one to specify a regex and replacement string for it (see the syntax of the java classes java.util.regex.Pattern and java.util.regex.Matcher) and creates a Literal out of the field value using the regex and the replacement string. dm:ResourceGenerator does the same as dm:LiteralGenerator but it generates a HTTP(S) URI that is used in place. So you can use the resource generator to generate URIs containing modified field values (e.g. to link to classifications). If you know regular expressions and turtle, the syntax should be quite self explanatory.

[dspace-source]/dspace/config/modules/rdf/fuseki-assembler.ttl

This is a configuration for the triple store Fuseki of the Apache Jena project. You can find more information on the configuration it provides in the section [Install a Triple Store](#) above.

Maintenance

As described [above](#) you should add `rdf` to the property `event.dispatcher.default.consumers` and in `dspace.cfg`. This configures DSpace to automatically update the triple store every time the publicly available content of the repository is changed. Nevertheless there is a command line tool that gives you the possibility to update the content of the triple store. As the triple store is used as a cache only, you can delete its content and reindex it every time you think it is necessary or helpful. The command line tool can be started by the following command which will show its online help:

```
[dspace-install]/bin/dspace dsrun rdfizer --help
```

The online help should give you all necessary information. There are commands to delete one specific entity; to delete all information stored in the triple store; to convert one item, one collection or community (including all subcommunities, collections and items) or to convert the complete content of your repository. If you start using the Linked Open Data support on a repository that already contains content, you should run `[dspace-install]/bin/dspace dsrun rdfizer --convert-all` once.

Every time content of DSpace is converted or Linked Data is requested, DSpace will try to connect to the triple store. So ensure that it is running (as you do with e.g. your servlet container or relational database).

4.3 Ingesting Content and Metadata

This is a new top level page grouping all documentation concerning all different ways to ingest content and metadata into DSpace

4.3.1 Submission User Interface

This page explains various customization and configuration options that are available within DSpace for the Item Submission user interface.

- [1 Default Submission Process](#)
 - [1.1 Optional Steps](#)
- [2 Understanding the Submission Configuration File](#)
 - [2.1 The Structure of item-submission.xml](#)
 - [2.2 Defining Steps \(<step>\) within the item-submission.xml](#)
 - [2.2.1 Where to place your <step> definitions](#)
 - [2.2.2 The ordering of <step> definitions matters!](#)
 - [2.2.3 Structure of the <step> Definition](#)
- [3 Reordering/Removing/Adding Submission Steps](#)
- [4 Assigning a custom Submission Process to a Collection](#)
 - [4.1 Getting A Collection's Handle](#)

- 5 [Custom Metadata-entry Pages for Submission](#)
 - 5.1 [Introduction](#)
 - 5.2 [Describing Custom Metadata Forms](#)
 - 5.3 [The Structure of input-forms.xml](#)
 - 5.3.1 [Adding a Collection Map](#)
 - 5.3.1.1 [Getting A Collection's Handle](#)
 - 5.3.2 [Adding a Form Set](#)
 - 5.3.2.1 [Forms and Pages](#)
 - 5.3.2.2 [Composition of a Field](#)
 - 5.3.2.3 [Item type Based Metadata Collection](#)
 - 5.3.2.4 [Automatically Omitted Fields](#)
 - 5.3.3 [Configuring Controlled Vocabularies](#)
 - 5.3.4 [Adding Value-Pairs](#)
 - 5.3.4.1 [Example](#)
 - 5.4 [Deploying Your Custom Forms](#)
- 6 [Configuring the File Upload step](#)
- 7 [Creating new Submission Steps](#)
 - 7.1 [Creating a Non-Interactive Step](#)
- 8 [Configuring StartSubmissionLookupStep](#)
 - 8.1 [About the Biblio-Transformation-Engine](#)
 - 8.2 [StartSubmissionLookupStep in action!](#)
 - 8.3 [SubmissionLookup service configuration file](#)

Default Submission Process

The DSpace Submission process consists of a series of "steps", where each "step" corresponds to one or more UI pages. By default, the DSpace Submission process includes the following steps, in this order:

1. "Select Collection" step: If not already selected, the user must select a collection to deposit the Item into.
2. "Describe" step: This is where the user may enter descriptive metadata about the Item. This step may consist of one or more pages of metadata entry. By default, there are two pages of metadata-entry. For information on modifying the metadata entry pages, please see [Custom Metadata-entry Pages for Submission](#) section below.
3. "Upload" step: This is where the user may upload one or more files to associate with the Item. For more information on file upload, also see [Configuring the File Upload step](#) below.
4. "Review" step: This is where the user may review all previous information entered, and correct anything as needed.
5. "License" step: This is where the user **must** agree to the repository distribution license in order to complete the deposit. This repository distribution license is defined in the `[dspace]/config/default.license` file. It can also be customized per-collection from the Collection Admin UI.
6. "Complete" step: The deposit is now completed. The Item will either become immediately available or undergo a workflow approval process (depending on the Collection policies). For more information on the workflow approval process see: [Configurable Workflow](#).

To modify or reorganize these submission steps, just modify the `[dspace]/config/item-submission.xml` file. Please see the section below on [Reordering/Removing/Adding Submission Steps](#).

You can also choose to have different submission processes for different DSpace Collections. For more details, please see the section below on [Assigning a custom Submission Process to a Collection](#).

DSpace 4.0 has removed the "Initial Questions" step by default

Prior to DSpace 4.0, the "Initial Questions" step preceded all "Describe" steps. However, it was removed by default in DSpace 4.0.

You may still choose to re-enable the "Initial Questions" step, as needed. However, please note the warning below about the auto-assigning of Dates in the "Initial Questions" step.

Optional Steps

DSpace also ships with several optional steps which you may choose to enable if you wish. In no particular order:

- "Access" step: This step allows the user to (optionally) modify access rights or set an embargo during the deposit of an Item. For more information on this step, and Embargo options in general, please see the [Embargo](#) documentation.
- "CC License" step: This step allows the user to (optionally) assign a Creative Commons license to a particular Item. Please see the [Configuring Creative Commons License](#) section of the Configuration documentation for more details.
- "Start Submission Lookup" step: This step allows the user to search or load metadata from an external service (arXiv online, bibtex file, etc.) and prefill the submission form. For more information on enabling and using it, please see the section on [Configuring StartSubmissionLookupStep](#) below.
- "Initial Questions" step: This step asks users a simple set of "initial questions" which help to determine which metadata fields are displayed in the "Describe" step (see above). These initial questions include:
 - *Multiple Titles*: The item has more than one title, *e.g.* a translated title (If selected, then users will be asked for an alternative title in the Describe step)
 - *Published Before*: The item has been published or publicly distributed before (If selected, then users will be asked for a publication date and publisher in the Describe step).

Initial Questions will auto-assign a publication date when "Published Before" is unselected

Please note, if you enable Initial Questions, and your users do NOT select "Published Before" option, then DSpace will auto-assign a publication date (dc.date.issued) to that particular Item.

It may be entirely accurate for some types of content (e.g. for gray literature or even theses/dissertations) to auto-assign this publication date. As such, you may wish to still enable "Initial Questions" if your repository is mainly for previously unpublished content. You may also choose to only enable it for specific Collections – see [Assigning a custom Submission Process to a Collection](#) section below.

However, if the Item actually was published in some other location, this will result in an incorrect publication date being reported by DSpace. This tendency for an incorrect publication date has been reported by Google Scholar to DSpace developers (see: [DS-1481](#)), which is why the "Initial Questions" are now disabled by default (see [DS-1655](#)).

To enable any of these optional submission steps, just uncomment the step definition within the `[dspace]/config/item-submission.xml` file. Please see the section below on [Reordering/Removing/Adding Submission Steps](#).

You can also choose to enable certain steps only for specific DSpace Collections. For more details, please see the section below on [Assigning a custom Submission Process to a Collection](#).

Understanding the Submission Configuration File

The `[dspace]/config/item-submission.xml` contains the submission configurations for *both* the DSpace JSP user interface (JSPUI) or the DSpace XML user interface (XMLUI or Manakin). This configuration file contains detailed documentation within the file itself, which should help you better understand how to best utilize it.

The Structure of `item-submission.xml`

```
<item-submission>
  <!-- Where submission processes are mapped to specific Collections -->
  <submission-map>
    <name-map collection-handle="default" submission-name="traditional" /> ...
  </submission-map>
  <!-- Where "steps" which are used across many submission processes can be defined in a
        single place. They can then be referred to by ID later. -->
  <step-definitions>
    <step id="collection">
      <processing-class>org.dspace.submit.step.SelectCollectionStep</process;/processing-class>
      <workflow-editable>false</workflow-editable>
    </step>
    ...
  </step-definitions>
  <!-- Where actual submission processes are defined and given names. Each <submission-process>
has
    many <step> nodes which are in the order that the steps should be in.-->
  <submission-definitions> <submission-process name="traditional">
    ...
  <!-- Step definitions appear here! -->
```

```

</submission-process>
...
</submission-definitions>
</item-submission>
  
```

Because this file is in XML format, you should be familiar with XML before editing this file. By default, this file contains the "traditional" Item Submission Process for DSpace, which consists of the following Steps (in this order):

Select Collection -> Describe -> Upload -> Verify -> License -> Complete

If you would like to customize the steps used or the ordering of the steps, you can do so within the `<submission-definition>` section of the *item-submission.xml*.

In addition, you may also specify different Submission Processes for different DSpace Collections. This can be done in the `<submission-map>` section. The *item-submission.xml* file itself documents the syntax required to perform these configuration changes.

Defining Steps (`<step>`) within the *item-submission.xml*

This section describes how Steps of the Submission Process are defined within the *item-submission.xml*.

Where to place your `<step>` definitions

`<step>` definitions can appear in one of two places within the *item-submission.xml* configuration file.

1. Within the `<step-definitions>` section

- This is for globally defined `<step>` definitions (i.e. steps which are used in multiple `<submission-process>` definitions). Steps defined in this section **must** define a unique *id* which can be used to reference this step.
- For example:

```

<step-definitions>
  <step id="custom-step">
    ...
  </step>
  ...
</step-definitions>
  
```

- The above step definition could then be referenced from within a `<submission-process>` as simply `<step id="custom-step"/>`

2. Within a specific `<submission-process>` definition

- This is for steps which are specific to a single `<submission-process>` definition.
- For example:

```

<submission-process>
  <step>
  
```



```

    ...
  </step>
</submission-process>

```

The ordering of <step> definitions matters!

The ordering of the <step> tags within a <submission-process> definition directly corresponds to the order in which those steps will appear!

For example, the following defines a Submission Process where the *License* step directly precedes the *Initial Questions* step (more information about the structure of the information under each <step> tag can be found in the section on Structure of the <step> Definition below):

```

<submission-process>
  <!--Step 1 will be to Sign off on the License-->
  <step>
    <heading>submit.progressbar.license</heading>
    <processing-class>org.dspace.submit.step.LicenseStep</processing-classing-class>
    <jspui-binding>org.dspace.app.webui.submit.step.JSPLicenseStep</jspui-binding>
    <xmlui-binding>org.dspace.app.xmlui.aspect.submission.submit.LicenseStenseStep</
xmlui-binding>
    <workflow-editable>false</workflow-editable>
  </step>
  <!--Step 2 will be to Ask Initial Questions-->
  <step>
    <heading>submit.progressbar.initial-questions</heading>
    <processing-class>org.dspace.submit.step.InitialQuestionsStep</process;/processing-class>
    <jspui-binding>org.dspace.app.webui.submit.step.JSPInitialQuestionsSteonsStep</
jspui-binding>
    <xmlui-binding>org.dspace.app.xmlui.aspect.submission.submit.InitialQutialQuestionsStep</
xmlui-binding>
    <workflow-editable>>true</workflow-editable>
  </step>
  ...[other steps]...
</submission-process>

```

Structure of the <step> Definition

The same <step> definition is used by both the DSpace JSP user interface (JSPUI) an the DSpace XML user interface (XMLUI or Manakin). Therefore, you will notice each <step> definition contains information specific to each of these two interfaces.

The structure of the <step> Definition is as follows:

```

<step>
  <heading>submit.progressbar.describe</heading>
  <processing-class>org.dspace.submit.step.DescribeStep</processing-classing-class>
  <jspui-binding>org.dspace.app.webui.submit.step.JSPDescribeStep</jspuilt;/jspui-binding>
  <xmlui-binding>org.dspace.app.xmlui.aspect.submission.submit.DescribeScribeStep</xmlui-binding>
  <workflow-editable>true</workflow-editable>

```

```
</step>
```

Each *step* contains the following elements. The required elements are so marked:

- **heading**: Partial I18N key (defined in *Messages.properties* for JSPUI or *messages.xml* for XMLUI) which corresponds to the text that should be displayed in the submission Progress Bar for this step. This partial I18N key is prefixed within either the *Messages.properties* or *messages.xml* file, depending on the interface you are using. Therefore, to find the actual key, you will need to search for the partial key with the following prefix:
 - XMLUI: prefix is *xmlui.Submission*. (e.g. "xmlui.Submission.submit.progressbar.describe" for 'Describe' step)
 - JSPUI: prefix is *jsp*. (e.g. "jsp.submit.progressbar.describe" for 'Describe' step) *The 'heading' need not be defined if the step should not appear in the progress bar (e.g. steps which perform automated processing, i.e. non-interactive, should not appear in the progress bar).*
- **processing-class** (Required): Full Java path to the Processing Class for this Step. This Processing Class **must** perform the primary processing of any information gathered in this step, for both the XMLUI and JSPUI. All valid step processing classes must extend the abstract `org.dspace.submit.AbstractProcessingStep` class (or alternatively, extend one of the pre-existing step processing classes in `org.dspace.submit.step.*`)
- **jspui-binding**: Full Java path of the JSPUI "binding" class for this Step. This "binding" class should initialize and call the appropriate JSPs to display the step's user interface. A valid JSPUI "binding" class *must* extend the abstract `org.dspace.app.webui.submit.JSPStep` class. *This property need not be defined if you are using the XMLUI interface, or for steps which only perform automated processing, i.e. non-interactive steps.*
- **xmlui-binding**: Full Java path of the XMLUI "binding" class for this Step. This "binding" class should generate the Manakin XML (DRI document) necessary to generate the step's user interface. A valid XMLUI "binding" class *must* extend the abstract `org.dspace.app.xmlui.submission.AbstractSubmissionStep` class. *This property need not be defined if you are using the JSPUI interface, or for steps which only perform automated processing, i.e. non-interactive steps.*
- **workflow-editable**: Defines whether or not this step can be edited during the *Edit Metadata* process with the DSpace approval/rejection workflow process. Possible values include *true* and *false*. If undefined, defaults to *true* (which means that workflow reviewers would be allowed to edit information gathered during that step).

Reordering/Removing/Adding Submission Steps

The removal of existing steps and reordering of existing steps is a relatively easy process!

Reordering steps

1. Locate the `<submission-process>` tag which defines the Submission Process that you are using. If you are unsure which Submission Process you are using, it's likely the one with `name="traditional"`, since this is the traditional DSpace submission process.
2. Reorder the `<step>` tags within that `<submission-process>` tag. Be sure to move the *entire* `<step>` tag (i.e. everything between and including the opening `<step>` and closing `</step>` tags).
 - *Hint #1:* The `<step>` defining the *Review/Verify* step only allows the user to review information from steps which appear **before** it. So, it's likely you'd want this to appear as one of your last few steps
 - *Hint #2:* If you are using it, the `<step>` defining the *Initial Questions* step should always appear **before** the *Upload* or *Describe* steps since it asks questions which help to set up those later steps.

Removing one or more steps

1. Locate the `<submission-process>` tag which defines the Submission Process that you are using. If you are unsure which Submission Process you are using, it's likely the one with `name="traditional"`, since this is the traditional DSpace submission process.
2. Comment out (i.e. surround with `<!--` and `-->`) the `<step>` tags which you want to remove from that `<submission-process>` tag. Be sure to comment out the *entire* `<step > tag` (i.e. *everything between and including the opening* `<step>` and closing `</step>` tags).
 - *Hint #1:* You cannot remove the *Select a Collection* step, as an DSpace Item cannot exist without belonging to a Collection.
 - *Hint #2:* If you decide to remove the `<step>` defining the *Initial Questions* step, you should be aware that this may affect your *Describe* and *Upload* steps! The *Initial Questions* step asks questions which help to initialize these later steps. If you decide to remove the *Initial Questions* step you may wish to create a custom, automated step which will provide default answers for the questions asked!

Adding one or more optional steps

1. Locate the `<submission-process>` tag which defines the Submission Process that you are using. If you are unsure which Submission Process you are using, it's likely the one with `name="traditional"`, since this is the traditional DSpace submission process.
2. Uncomment (i.e. remove the `<!--` and `-->`) the `<step>` tag(s) which you want to add to that `<submission-process>` tag. Be sure to uncomment the *entire* `<step> tag` (i.e. *everything between and including the opening* `<step>` and closing `</step>` tags).

Assigning a custom Submission Process to a Collection

Assigning a custom submission process to a Collection in DSpace involves working with the *submission-map* section of the *item-submission.xml*. For a review of the structure of the *item-submission.xml* see the section above on Understanding the Submission Configuration File.

Each *name-map* element within *submission-map* associates a collection with the name of a submission definition. Its *collection-handle* attribute is the Handle of the collection. Its *submission-name* attribute is the submission definition name, which must match the *name* attribute of a *submission-process* element (in the *submission-definitions* section of *item-submission.xml*).

For example, the following fragment shows how the collection with handle "12345.6789/42" is assigned the "custom" submission process:

```
<submission-map>
  <name-map collection-handle=" 12345.6789/42" submission-name="custom" />
  ...
</submission-map>
<submission-definitions>
  <submission-process name="custom">
  ...
</submission-definitions>
```

It's a good idea to keep the definition of the *default* name-map from the example *input-forms.xml* so there is always a default for collections which do not have a custom form set.

Getting A Collection's Handle

You will need the *handle* of a collection in order to assign it a custom form set. To discover the handle, go to the "Communities & Collections" page under "Browse" in the left-hand menu on your DSpace home page. Then, find the link to your collection. It should look something like:

```
http://myhost.my.edu/dspace/handle/12345.6789/42
```

The underlined part of the URL is the handle. It should look familiar to any DSpace administrator. That is what goes in the *collection-handle* attribute of your *name-map* element.

Custom Metadata-entry Pages for Submission

Introduction

This section explains how to customize the Web forms used by submitters and editors to enter and modify the metadata for a new item. These metadata web forms are controlled by the *Describe* step within the Submission Process. However, they are also configurable via their own XML configuration file (*input-forms.xml*).

You can customize the "default" metadata forms used by all collections, and also create alternate sets of metadata forms and assign them to specific collections. In creating custom metadata forms, you can choose:

- The number of metadata-entry pages.
- Which fields appear on each page, and their sequence.
- Labels, prompts, and other text associated with each field.

- List of available choices for each menu-driven field.

NOTE: The cosmetic and ergonomic details of metadata entry fields remain the same as the fixed metadata pages in previous DSpace releases, and can only be altered by modifying the appropriate stylesheet and JSP pages.

All of the custom metadata-entry forms for a DSpace instance are controlled by a single XML file, *input-forms.xml*, in the *config* subdirectory under the DSpace home. DSpace comes with a sample configuration that implements the traditional metadata-entry forms, which also serves as a well-documented example. The rest of this section explains how to create your own sets of custom forms.

Describing Custom Metadata Forms

The description of a set of pages through which submitters enter their metadata is called a *form* (although it is actually a set of forms, in the HTML sense of the term). A form is identified by a unique symbolic *name*. In the XML structure, the *form* is broken down into a series of *pages*: each of these represents a separate Web page for collecting metadata elements.

To set up one of your DSpace collections with customized submission forms, first you make an entry in the *form-map*. This is effectively a table that relates a collection to a form set, by connecting the collection's *Handle* to the form name. Collections are identified by handle because their names are mutable and not necessarily unique, while handles are unique and persistent.

A special map entry, for the collection handle "default", defines the *default* form set. It applies to all collections which are not explicitly mentioned in the map. In the example XML this form set is named *traditional* (for the "traditional" DSpace user interface) but it could be named anything.

The Structure of input-forms.xml

The XML configuration file has a single top-level element, *input-forms*, which contains three elements in a specific order. The outline is as follows:

```
<input-forms>
  <!-- Map of Collections to Form Sets -->
  <form-map>
    <name-map collection-handle="default" form-name="traditional" />
    ...
  </form-map>
  <!-- Form Set Definitions -->
  <form-definitions>
    <form name="traditional">
      ...
    </form>
    ...
  </form-definitions>
  <!-- Name/Value Pairs used within Multiple Choice Widgets -->
  <form-value-pairs>
    <value-pairs value-pairs-name="common_iso_languages" dc-term="language_iso">
      ...
    </value-pairs>
  </form-value-pairs>
</input-forms>
```

```
        </value-pairs>
        ...
    </form-value-pairs>
</input-forms>
```

Adding a Collection Map

Each *name-map* element within *form-map* associates a collection with the name of a form set. Its *collection-handle* attribute is the Handle of the collection, and its *form-name* attribute is the form set name, which must match the *name* attribute of a *form* element.

For example, the following fragment shows how the collection with handle "12345.6789/42" is attached to the "TechRpt" form set:

```
<form-map>
  <name-map collection-handle=" 12345.6789/42" form-name=" TechRpt" />
  ...
</form-map>
<form-definitions>
  <form name="TechRept">
    ...
</form-definitions>
```

It's a good idea to keep the definition of the *default* name-map from the example *input-forms.xml* so there is always a default for collections which do not have a custom form set.

Getting A Collection's Handle

You will need the *handle* of a collection in order to assign it a custom form set. To discover the handle, go to the "Communities & Collections" page under "**Browse**" in the left-hand menu on your DSpace home page. Then, find the link to your collection. It should look something like:

```
http://myhost.my.edu/dspace/handle/12345.6789/42
```

The underlined part of the URL is the handle. It should look familiar to any DSpace administrator. That is what goes in the *collection-handle* attribute of your *name-map* element.

Adding a Form Set

You can add a new form set by creating a new *form* element within the *form-definitions* element. It has one attribute, *name*, which as seen above must match the value of the *name-map* for the collections it is to be used for.

Forms and Pages

The content of the *form* is a sequence of *page* elements. Each of these corresponds to a Web page of forms for entering metadata elements, presented in sequence between the initial "Describe" page and the final "Verify" page (which presents a summary of all the metadata collected).

A *form* must contain at least one and at most six pages. They are presented in the order they appear in the XML. Each *page* element must include a *number* attribute, that should be its sequence number, e.g.

```
<page number="1" >
```

The *page* element, in turn, contains a sequence of *field* elements. Each field defines an interactive dialog where the submitter enters one of the Dublin Core metadata items.

Composition of a Field

Each *field* contains the following elements, in the order indicated. The required sub-elements are so marked:

- **dc-schema** (Required) : Name of metadata schema employed, e.g. *dc* for Dublin Core. This value must match the value of the *schema* element defined in *dublin-core-types.xml*
- **dc-element** (Required) : Name of the Dublin Core element entered in this field, e.g. *contributor*.
- **dc-qualifier**: Qualifier of the Dublin Core element entered in this field, e.g. when the field is *contributor.advisor* the value of this element would be *advisor*. Leaving this out means the input is for an unqualified DC element.
- **repeatable**: Value is *true* when multiple values of this field are allowed, *false* otherwise. When you mark a field repeatable, the UI servlet will add a control to let the user ask for more fields to enter additional values. Intended to be used for arbitrarily-repeating fields such as subject keywords, when it is impossible to know in advance how many input boxes to provide.
- **label** (Required): Text to display as the label of this field, describing what to enter, e.g. " *Your Advisor's Name*".
- **input-type**(Required): Defines the kind of interactive widget to put in the form to collect the Dublin Core value. Content must be one of the following keywords:
 - **onebox** – A single text-entry box.
 - **twobox** – A pair of simple text-entry boxes, used for *repeatable* values such as the DC *subject* item. *Note*: The 'twobox' input type is rendered the same as a 'onebox' in the XML-UI, but both allow for ease of adding multiple values.
 - **textarea** – Large block of text that can be entered on multiple lines, e.g. for an abstract.
 - **name** – Personal name, with separate fields for family name and first name. When saved they are appended in the format 'LastName, FirstName'
 - **date** – Calendar date. When required, demands that at least the year be entered.
 - **series** – Series/Report name and number. Separate fields are provided for series name and series number, but they are appended (with a semicolon between) when saved.
 - **dropdown** – Choose value(s) from a "drop-down" menu list. **Note**: You must also include a value for the *value-pairs-name* attribute to specify a list of menu entries from which to choose. Use this to make a choice from a restricted set of options, such as for the *language* item.
 - **qualdrop_value** – Enter a "qualified value", which includes *both* a qualifier from a drop-down menu and a free-text value. Used to enter items like alternate identifiers and codes for a submitted item, e.g. the DC *identifier* field. **Note**: As for the *dropdown* type, you must include the *value-pairs-name* attribute to specify a menu choice list.

- **list** – Choose value(s) from a checkbox or radio button list. If the *repeatable* attribute is set to *true*, a list of checkboxes is displayed. If the *repeatable* attribute is set to *false*, a list of radio buttons is displayed. **Note:** You must also include a value for the *value-pairs-name* attribute to specify a list of values from which to choose.
- **hint** (Required): Content is the text that will appear as a "hint", or instructions, next to the input fields. Can be left empty, but it must be present.
- **required**: When this element is included with any content, it marks the field as a required input. If the user tries to leave the page without entering a value for this field, that text is displayed as a warning message. For example, `<required>You must enter a title.</required>` Note that leaving the required element empty will *not* mark a field as required, e.g.: `<required></required>`
- **visibility**: When this optional element is included with a value, it restricts the visibility of the field to the scope defined by that value. If the element is missing or empty, the field is visible in all scopes. Currently supported scopes are:
 - **workflow** : the field will only be visible in the workflow stages of submission. This is good for hiding difficult fields for users, such as subject classifications, thereby easing the use of the submission system.
 - **submit** : the field will only be visible in the initial submission, and not in the workflow stages. In addition, you can decide which type of restriction apply: read-only or full hidden the field (default behaviour) using the *otherwise* attribute of the *visibility* XML element. For example: `<visibility otherwise="readonly">workflow</visibility>` Note that it is considered a configuration error to limit a field's scope while also requiring it - an exception will be generated when this combination is detected.
Look at the example *input-forms.xml* and experiment with a a trial custom form to learn this specification language thoroughly. It is a very simple way to express the layout of data-entry forms, but the only way to learn all its subtleties is to use it.

For the use of controlled vocabularies see the Configuring Controlled Vocabularies section.

Item type Based Metadata Collection

This feature is available for use with the XMLUI since DSpace 3.0 and with JSPUI since 3.1. A field can be made visible depending on the value of *dc.type*. A new field element, `<type-bind>`, has been introduced to facilitate this. In this example the field will only be visible if a value of "thesis" or "ebook" has been entered into *dc.type* on an earlier page:

```
<field>
  <dc-schema>dc</dc-schema>
  <dc-element>identifier</dc-element>
  <dc-qualifier>isbn</dc-qualifier>
  <label>ISBN</label>
  <type-bind>thesis,ebook</type-bind>
</field>
```

Automatically Omitted Fields

You may notice that some fields are automatically skipped when a custom form page is displayed, depending on the kind of item being submitted. This is because the DSpace user-interface engine skips Dublin Core fields which are not needed, according to the initial description of the item. For example, if the user indicates there are no alternate titles on the first "Describe" page (the one with a few checkboxes), the input for the *title.alternative* DC element is automatically omitted, *even on custom submission pages*.

When a user initiates a submission, DSpace first displays what we'll call the "initial-questions page". By default, it contains three questions with check-boxes:

1. **The item has more than one title, e.g. a translated title** Controls *title.alternative* field.
2. **The item has been published or publicly distributed before** Controls DC fields:
 - *date.issued*
 - *publisher*
 - *identifier.citation*
3. **The item consists of more than one file** *Does not affect any metadata input fields.*

The answers to the first two questions control whether inputs for certain of the DC metadata fields will displayed, even if they are defined as fields in a custom page. Conversely, if the metadata fields controlled by a checkbox are not mentioned in the custom form, the checkbox is omitted from the initial page to avoid confusing or misleading the user.

The two relevant checkbox entries are "The item has more than one title, e.g. a translated title", and "The item has been published or publicly distributed before". The checkbox for multiple titles trigger the display of the field with dc-element equal to "title" and dc-qualifier equal to "alternative". If the controlling collection's form set does not contain this field, then the multiple titles question will not appear on the initial questions page.

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 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.

The taxonomies are described in XML following this (very simple) structure:

```
<node id="acmccs98" label="ACMCCS98">
  <isComposedBy>
    <node id="A." label="General Literature">
      <isComposedBy>
        <node id="A.0" label="GENERAL"/>
        <node id="A.1" label="INTRODUCTORY AND SURVEY"/>
        ...
      </isComposedBy>
    </node>
    ...
  </isComposedBy>
```

```
</node>
```

You are free to use any application you want to create your controlled vocabularies. A simple text editor should be enough for small projects. Bigger projects will require more complex tools. You may use Protegé to create your taxonomies, save them as OWL and then use a XML Stylesheet (XSLT) to transform your documents to the appropriate format. Future enhancements to this add-on should make it compatible with standard schemas such as OWL or RDF.

New vocabularies should be placed in `[dspace]/config/controlled-vocabularies/` and must be according to the structure described.

Vocabularies need to be associated with the correspondant 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>
  <repeatable>true</repeatable>
  <label>Subject Keywords</label>
  <input-type>onebox</input-type>
  <hint>Enter appropriate subject keywords or phrases below.</hint>
  <required></required>
  <vocabulary>srsc</vocabulary>
</field>
```

The vocabulary element has an optional boolean attribute closed that can be used to force input only with the Javascript of controlled-vocabulary add-on. The default behaviour (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

Adding Value-Pairs

Finally, your custom form description needs to define the "value pairs" for any fields with input types that refer to them. Do this by adding a *value-pairs* element to the contents of *form-value-pairs*. It has the following required attributes:

- **value-pairs-name** – Name by which an *input-type* refers to this list.
- **dc-term** – Dublin Core field for which this choice list is selecting a value.

Each *value-pairs* element contains a sequence of *pair* sub-elements, each of which in turn contains two elements:

- **displayed-value** – Name shown (on the web page) for the menu entry.
- **stored-value** – Value stored in the DC element when this entry is chosen. Unlike the HTML *select* tag, there is no way to indicate one of the entries should be the default, so the first entry is always the default choice.

Example

Here is a menu of types of common identifiers:

```
<value-pairs value-pairs-name="common_identifiers" dc-term="identifier">
  <pair>
    <displayed-value>Gov't Doc #</displayed-value>
    <stored-value>govdoc</stored-value>
  </pair>
  <pair>
    <displayed-value>URI</displayed-value>
    <stored-value>uri</stored-value>
  </pair>
  <pair>
    <displayed-value>ISBN</displayed-value>
    <stored-value>isbn</stored-value>
  </pair>
</value-pairs>
```

It generates the following HTML, which results in the menu widget below. (Note that there is no way to indicate a default choice in the custom input XML, so it cannot generate the HTML *SELECTED* attribute to mark one of the options as a pre-selected default.)

```
<select name="identifier_qualifier_0">
  <option VALUE="govdoc">Gov't Doc #</option>
  <option VALUE="uri">URI</option>
  <option VALUE="isbn">ISBN</option>
</select>
```

Deploying Your Custom Forms

The DSpace web application only reads your custom form definitions when it starts up, so it is important to remember:

- *You must always restart Tomcat* (or whatever servlet container you are using) for changes made to the *input-forms.xml* file take effect.

Any mistake in the syntax or semantics of the form definitions, such as poorly formed XML or a reference to a nonexistent field name, will cause a fatal error in the DSpace UI. The exception message (at the top of the stack trace in the *dspace.log* file) usually has a concise and helpful explanation of what went wrong. Don't forget to stop and restart the servlet container before testing your fix to a bug.

Configuring the File Upload step

The *Upload* step in the DSpace submission process has two configuration options which can be set with your `[dspace]/config/dspace.cfg` configuration file. They are as follows:

- *upload.max*- The maximum size of a file (in bytes) that can be uploaded from the JSPUI (not applicable for the XMLUI). It defaults to 536870912 bytes (512MB). You may set this to -1 to disable any file size limitation.
 - *Note:* Increasing this value or setting to -1 does **not** guarantee that DSpace will be able to successfully upload larger files via the web, as large uploads depend on many other factors including bandwidth, web server settings, internet connection speed, etc.
- *webui.submit.upload.required*- Whether or not all users are *required* to upload a file when they submit an item to DSpace. It defaults to 'true'. When set to 'false' users will see an option to skip the upload step when they submit a new item.

Creating new Submission Steps

First, a brief warning: *Creating a new Submission Step requires some Java knowledge, and is therefore recommended to be undertaken by a Java programmer whenever possible*

That being said, at a higher level, creating a new Submission Step requires the following (in this relative order):

1. **(Required)** Create a new Step Processing class
 - This class **must** extend the abstract `org.dspace.submit.AbstractProcessingStep` class and implement all methods defined by that abstract class.
 - This class should be built in such a way that it can process the input gathered from *either* the XMLUI or JSPUI interface.
2. *(For steps using JSPUI)* Create the JSPs to display the user interface. Create a new JSPUI "binding" class to initialize and call these JSPs.
 - Your JSPUI "binding" class must extend the abstract class `org.dspace.app.webui.submit.JSPStep` and implement all methods defined there. It's recommended to use one of the classes in `org.dspace.app.webui.submit.step.*` as a reference.
 - Any JSPs created should be loaded by calling the `showJSP()` method of the `org.dspace.app.webui.submit.JSPStepManager` class
 - If this step gathers information to be reviewed, you must also create a Review JSP which will display a read-only view of all data gathered during this step. The path to this JSP must be returned by your `getReviewJSP()` method. You will find examples of Review JSPs (named similar to `review-[step].jsp`) in the JSP `submit/` directory.

3. (*For steps using XMLUI*) Create an XMLUI "binding" Step Transformer which will generate the DRI XML which Manakin requires.
 - The Step Transformer must extend and implement all necessary methods within the abstract class `org.dspace.app.xmlui.submission.AbstractSubmissionStep`
 - It is useful to use the existing classes in `org.dspace.app.xmlui.submission.submit.*` as references
4. (**Required**) Add a valid Step Definition to the `item-submission.xmlconfiguration` file.
 - This may also require that you add an I18N (Internationalization) key for this step's *heading*. See the sections on [Configuring Multilingual Support for JSPUI](#) or [Configuring Multilingual Support for XMLUI](#) for more details.
 - For more information on `<step>` definitions within the `item-submission.xml`, see the section above on Defining Steps (`<step>`) within the `item-submission.xml`.

Creating a Non-Interactive Step

Non-interactive steps are ones that have no user interface and only perform backend processing. You may find a need to create non-interactive steps which perform further processing of previously entered information.

To create a non-interactive step, do the following:

1. Create the required Step Processing class, which extends the abstract `org.dspace.submit.AbstractProcessingStep` class. In this class add any processing which this step will perform.
2. Add your non-interactive step to your `item-submission.xml` at the place where you wish this step to be called during the submission process. For example, if you want it to be called *immediately after* the existing 'Upload File' step, then place its configuration immediately after the configuration for that 'Upload File' step. The configuration should look similar to the following:

```
<step>
  <processing-class>org.dspace.submit.step.MyNonInteractiveStep</processing-class>
  <workflow-editable>false</workflow-editable>
</step>
```

Note: Non-interactive steps will not appear in the Progress Bar! Therefore, your submitters will not even know they are there. However, because they are not visible to your users, you should make sure that your non-interactive step does not take a large amount of time to finish its processing and return control to the next step (otherwise there will be a visible time delay in the user interface).

Configuring StartSubmissionLookupStep

StartSubmissionLookupStep is a new submission step, available since DSpace 4.0 contributed by [CINECA](#), that extends the basic SelectCollectionStep allowing the user to search or load metadata from an external service (arxiv online, bibtex file, etc.) and prefill the submission form. Thanks to the [EKT](#) works it is underpinned by the Biblio Transformation Engine (<https://github.com/EKT/Biblio-Transformation-Engine>) framework.

To enable the StartSubmissionLookupStep you only need to change the configuration of the id="collection" step to match the following

item-submission.xml excerpt

```
<step id="collection">
  <heading></heading> <!--can specify heading, if you want it to appear in Progress Bar-->
  <processing-class>org.dspace.submit.step.StartSubmissionLookupStep</processing-class>
  <jspui-binding>org.dspace.app.webui.submit.step.JSPStartSubmissionLookupStep</jspui-binding>
  <xmlui-binding>org.dspace.app.xmlui.aspect.submission.submit.SelectCollectionStep</xmlui-
binding>
  <workflow-editable>>false</workflow-editable>
</step>
```



UI compatibility

The new step is available **only for JSP UI**. Nonetheless, if you run both UIs and want the JSP UI benefit of the new step you can configure it as processing class also for XML as it degrades gracefully to the standard SelectCollectionStep logic

About the Biblio-Transformation-Engine

The BTE is a Java framework developed by the Hellenic National Documentation Centre ([EKT](#)) and consists of programmatic APIs for filtering and modifying records that are retrieved from various types of data sources (eg. databases, files, legacy data sources) as well as for outputting them in appropriate standards formats (eg. database files, txt, xml, Excel). The framework includes independent abstract modules that are executed separately, offering in many cases alternative choices to the user depending of the input data set, the transformation workflow that needs to be executed and the output format that needs to be generated.

The basic idea behind the BTE is a standard workflow that consists of three steps, a data loading step, a processing step (record filtering and modification) and an output generation. A data loader provides the system with a set of Records, the processing step is responsible for filtering or modifying these records and the output generator outputs them in the appropriate format.

The standard BTE version offers several predefined Data Loaders as well as Output Generators for basic bibliographic formats. However, Spring Dependency Injection can be utilized to load custom data loaders, filters, modifiers and output generators.

StartSubmissionLookupStep in action!

When StartSubmissionLookupStep is enabled, the user comes up with the following screen when a new submission is initiated:

New submission: get data from bibliographic external service

No collection selected

The screenshot shows a web interface for searching external bibliographic services. It features two tabs: 'Search Form' and 'Results', with 'Search Form' currently selected. Below the tabs is a section titled 'Search for identifier' with a dropdown arrow. The instructions state: 'Fill in publication identifiers (DOI is preferable) and then press "Search". A list of all matching publications will be shown to you to select in order to proceed with the submission process.' There are three input fields: 'PubMed ID:' (with example 'e.g. 20524090'), 'DOI (Digital Object Identifier):' (with example 'e.g. 10.1392/dironix'), and 'arXiv ID:' (with example 'e.g. arXiv:1302.1497'). To the right of each field are logos for the corresponding services: PubMed, CrossRef, and arXiv.org. At the bottom of the search area, there are two buttons: 'Upload a file' and 'Default mode Submission'. A 'Go to DSpace Home' link is located at the bottom left of the interface.

There are four accordion tabs (default configuration hides the third tab):

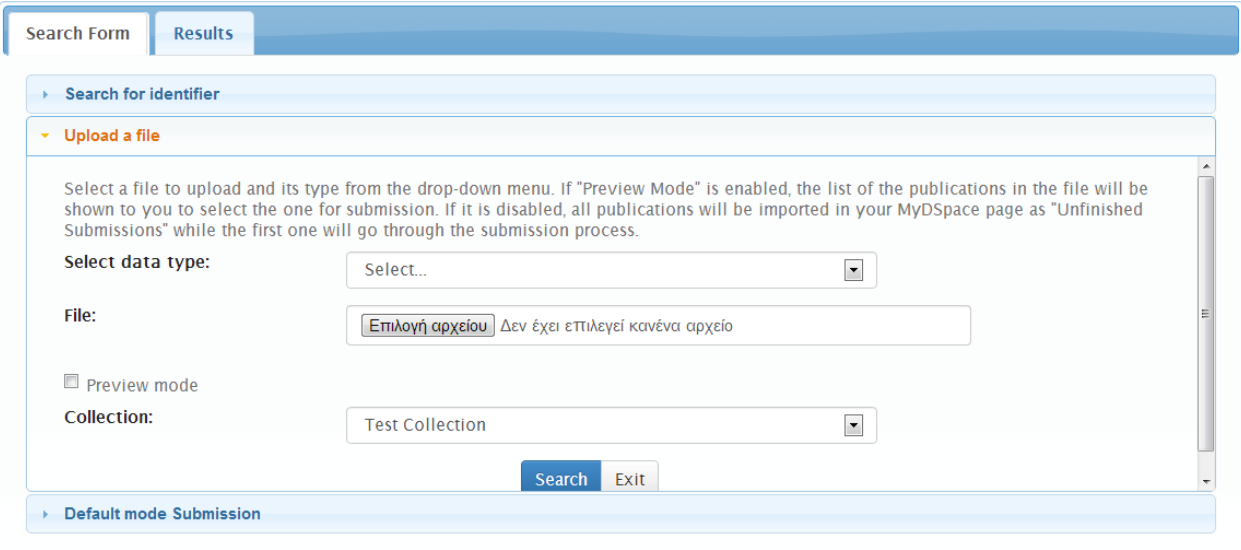
1) Search for identifier: In this tab, the user can search for an identifier in the supported online services (currently, [arXiv](#), [PubMed](#), [CrossRef](#) and [CiNii](#) are supported). The publication results are presented in the tab "Results" in which the user can select the publication to proceed with. This means that a new submission form will be initiated with the form fields prefilled with metadata from the selected publication.

Currently, there are four identifiers that are supported (DOI, PubMed ID, arXiv ID and NAID (CiNii ID)). But these can be extended - refer to the following paragraph regarding the SubmissionLookup service configuration file.

User can fill in any of the four identifiers. DOI is preferable. Keep in mind that the service can integrate results for the same publication from the three different providers so filling any of the four identifiers will pretty much do the work. If identifiers for different publications are provided, the service will return a list of publications which will be shown to user to select. The selected publication will make it to the submission form in which some fields will be pre-filled with the publication metadata. The mapping from the input metadata (from arXiv or Pubmed or CrossRef or CiNii) to the DSpace metadata schema (and thus, the submission form) is configured in the Spring XML file that is discussed later on - you can see a table at the very end of this chapter.

Through the same file, a user can also extend the providers that the SubmissionLookup service can search publication from.

2) Upload a file: In this tab, the user can upload a file, select the type (bibtex, csv, etc.), see the publications in the "Results" tab and then either select one to proceed with the submission or make all of them "Workspace Items" that can be found in the "Unfinished Submissions" section in the "My DSpace" page.



The screenshot shows a web interface for uploading a file to a DSpace repository. The main heading is "New submission: get data from bibliographic external service". Below this, there is a yellow banner indicating "No collection selected". The interface is divided into two tabs: "Search Form" and "Results". The "Results" tab is active, showing a section titled "Upload a file". This section contains a text area with instructions: "Select a file to upload and its type from the drop-down menu. If 'Preview Mode' is enabled, the list of the publications in the file will be shown to you to select the one for submission. If it is disabled, all publications will be imported in your MyDSpace page as 'Unfinished Submissions' while the first one will go through the submission process." Below the text area, there are several form elements: a "Select data type:" dropdown menu with "Select..." as the current selection; a "File:" input field with a button labeled "Επιλογή αρχείου" and a message "Δεν έχει επιλεγεί κανένα αρχείο"; a "Preview mode" checkbox which is currently unchecked; and a "Collection:" dropdown menu with "Test Collection" selected. At the bottom of the form, there are "Search" and "Exit" buttons. Below the form, there is a "Default mode Submission" section and a "Go to" link.

The "preview mode" in the figure above has the following functionality:

"ON": The list of the publications in the uploaded file will be shown to the user to select the one for the submission. The selected publication's metadata will pre-fill the submission form's fields according to configuration in the Spring XML configuration file.

"OFF": All the publications of the uploaded file will be imported in the user's MyDSpace page as "Unfinished Submissions" while the first one will go through the submission process.

(Regarding the pubmed, crossref and arxiv file upload, you can find the attached file named "sample-files.zip" that contains samples of these three file types)

3) Free search: In this tab, the user can freely search for Title, Author and Year in the four supported providers (PubMed, CrossRef, Arxiv and CiNii). By default, the four providers are configured to be disabled for free search but you can enable it via the configuration file. Thus, initially this accordion tab is not shown to the user except for a data loader is declared as a "search provider" - refer to the following paragraphs.

New submission: get data from bibliographic external service

No collection selected

The screenshot shows a web interface for a new submission. At the top, there is a yellow banner indicating 'No collection selected'. Below this, there are two tabs: 'Search Form' and 'Results'. The 'Search Form' tab is active, showing a 'Free search' section. This section includes the PubMed logo and instructions: 'Insert base info about publication: either title or author/year is required. If you know any unique identifier about publication like DOI, Pubmed, or arXiv you can switch on the identifier search mode.' There are three input fields: 'Title:', 'Year:', and 'Authors/Publishers :'. Below these fields are three buttons: 'Search for identifier', 'Upload a file', and 'Default mode Submission'.

The process is the same as in the previous cases. A result of publications is presented to the user to select the one to proceed with the submission.

4) Default mode submission: In this tab, the user can proceed to the default manual submission. The SubmissionLookup service will not run and the submission form will be empty for the user to start filling it.

SubmissionLookup service configuration file

The StartSubmissionLookupStep rely on business logic provided by the SubmissionLookup service that can be heavily extended and customized and is build on top of the BTE.

The basic idea behind BTE is that the system holds the metadata in an internal format using a specific key for each metadata field. DataLoaders load the record using the aforementioned keys, while the output generator needs to map these keys to DSpace metadata fields.

The BTE configuration file is located in path: `[dspace]/config/spring/api/bte.xml` and it's a Spring XML configuration file that consists of Java beans. (If these terms are unknown to you, please refer to Spring Dependency Injection web site for more information.)

The service is broken down into two phases. In the first phase, the imported publications' metadata are converted to an intermediate format while in the second phase, the intermediate format is converted to DSpace metadata schema

Explanation of beans:

```
<bean id="org.dspace.submit.lookup.SubmissionLookupService" />
```

This is the top level bean that describes the service of the SubmissionLookup. It accepts three properties:

- a) **phase1TransformationEngine** : the phase 1 BTE transformation engine.
- b) **phase2TransformationEngine** : the phase 2 BTE transformation engine
- c) **detailFields**: A list of the keys that the user wants to display in the detailed form of a publication. That is, when the results are shown, user can see the details of each one. In the detailed form, some fields appear. These fields are configured by this property. Refer to the table at the very end of this chapter to see the available values. This property is disabled by default while the list that is shown commented out is the default list for the detailed form.

```
<bean id="phase1TransformationEngine" />
```

The transformation engine for the first phase of the service (from external service to intermediate format)

It accepts three properties:

- a) **dataLoader** : The data loader that will be used for the loading of the data
- b) **workflow** : This property refers to the bean that describes the processing steps of the BTE. If no processing steps are listed there all records loaded by the data loader will pass to the output generator, unfiltered and unmodified.
- c) **outputGenerator** : The output generator to be used.

Normally, you do not need to touch any of these three properties. You can edit the reference beans instead.

```
<bean id="multipleDataLoader" />
```

This bean declares the data loader to be used to load publications from. It has one property "dataloadersMap", a map that declares key-value pairs, that is a unique key and the corresponding data loader to be used. Here is the point where a new data loader can be added, in case the ones that are already supported do not meet your needs.

A new data loader class must be created based on the following:

- a) Either extend the abstract class **gr.ekt.bte.core.dataloader.FileDataLoader**

in such a case, your data loader key will appear in the drop down menu of data types in the "Upload a file" accordion tab

b) Or, extend the abstract class `org.dspace.submit.lookup.SubmissionLookupDataLoader`

in such a case, your data loader key will appear as a provider in the " *Search for identifier*" accordion tab

```
<bean id="bibTeXDataLoader" />
<bean id="csvDataLoader" />
<bean id="tsvDataLoader" />
<bean id="risDataLoader" />
<bean id="endnoteDataLoader" />
<bean id="pubmedFileDataLoader" />
<bean id="arXivFileDataLoader" />
<bean id="crossRefFileDataLoader" />
<bean id="ciniiFileDataLoader" />
<bean id="pubmedOnlineDataLoader" />
<bean id="arXivOnlineDataLoader" />
<bean id="crossRefOnlineDataLoader" />
<bean id="ciniiOnlineDataLoader" />
```

These beans are the actual data loaders that are used by the service. They are either "FileDataLoaders" or "SubmissionLookupDataLoaders" as mentioned previously.

The data loaders have the following properties:

a) fieldMap : it is a map that specifies the mapping between the keys that hold the metadata in the input format and the ones that we want to have internal in the BTE. At the end of this article there is a table that summarises the fields that are used from the three online services (pubmed, arXiv and crossRef) - which are the ones that the submission lookup step is capable of reading from the online services - and the keys used internally in the BTE.

Some loaders have more properties:

CSV and **TSV** (which is actually a CSV loader if you look carefully the class value of the bean) loaders have some more properties:

a) skipLines: A number that specifies the first line of the file that loader will start reading data. For example, if you have a csv file that the first row contains the column names, and the second row is empty, the the value of this property must be 2 so as the loader starts reading from row 2 (starting from 0 row). The default value for this property is 0.

b) separator: A value to specify the separator between the values in the same row in order to make the columns. For example, in a TSV data loader this value is "\u0009" which is the "Tab" character. The default value is "," and that is why the CSV data loader doesn't need to specify this property.

c) quoteChar: This property specifies the quote character used in the CSV file. The default value is the double quote character (").

pubmedOnlineDataLoader, **crossRefOnlineDataLoader**, **arXivOnlineDataLoader** and **ciniiOnlineDataLoader** also support another property:

a) searchProvider: if is set to true, the dataloader supports free search by title, author or year. If at least one of these data loaders is declared as a search provider, the accordion tab "Free search" is appeared. Otherwise, it stays hidden.

crossRefOnlineDataLoader and **ciniiOnlineDataLoader** also have two more properties:

a) apiKey/appld respectively: Both these services need to acquire (for free) an API key in order to access their online services. For CrossRef, visit: <http://www.crossref.org/requestaccount/> and for CiNii visit: <https://portaltools.nii.ac.jp/developer/en/>

b) maxResults: the maximum results that these services will reply with to your search. By default, this property is commented out while the default value is 10 for both services.

(Regarding the file dataloaders, you can find the attached file named "sample-files.zip" that contains samples of all the file types that the corresponding data loaders can handle)

```
<bean id="phaseLinearWorkflow" />
```

This bean specifies the processing steps to be applied to the records metadata before they proceed to the output generator of the transformation engine. Currently, three steps are supported, but you can add yours as well.

```
<bean id="mapConverter_arxivSubject" />
<bean id="mapConverter_pubstatusPubmed" />
<bean id="removeLastDot" />
```

These beans are the processing steps that are supported by the 1st phase of transformation engine. The two first map an incoming value to another one specified in a properties file. The last one is responsible to remove the last dot from the incoming value.

All of them have the property "**fieldKeys**" which is a list of keys where the step will be applied.

In the case you need to create your own filters and modifiers follow the instructions below:

To create a new filter, you need to extend the following BTE abstract class:

```
gr.ekt.bte.core.AbstractFilter
```

You will need to implement the following method:

```
public abstract boolean isIncluded ( Record record )
```

Return false if the specified record needs to be filtered, otherwise return true.

To create a new modifier, you need to extend the following BTE abstract class:

```
gr.ekt.bte.core.AbstractModifier
```

You will need to implement the following method:

```
public abstract Record modify ( Record record )
```

within you can make any changes you like in the record. You can use the Record methods to get the values for a specific key and load new ones (For the later, you need to make the Record mutable)

After you create your own filters or modifiers you need to add them in the Spring XML configuration file as in the following example:

```
<bean id="customfilter" class="org.mypackage.MyFilter" />
<bean id="phase1LinearWorkflow" class="gr.ekt.bte.core.LinearWorkflow">
  <property name="process">
    <list>
      ... <old filters and modifiers>...
      <ref bean="customfilter" />
    </list>
  </property>
</bean>
```

```
<bean id="phase2TransformationEngine" />
```

The transformation engine for the second phase of the service (from the intermediate format to DSpace metadata schema)

Normally, you do not need to touch any of these three properties. You can edit the reference beans instead.

```
<bean id="phase2linearWorkflow" />
```

This bean specifies the processing steps to be applied to the records metadata before they proceed to the output generator of the transformation engine. Currently, two steps are supported, but you can add yours as well.

```
<bean id="fieldMergeModifier" />
<bean id="valueConcatenationModifier" />
<bean id="languageCodeModifier" />
```

These beans are the processing steps that are supported by the 2nd phase of transformation engine. The first merges the values of multiple keys to a new key. The second one concatenates the values of a specific key to a unique value. The third one translated the three-letters language code to two-letters one (ie: eng to en)

```
<bean id="org.dspace.submit.lookup.DSpaceWorkspaceItemOutputGenerator" />
```

This bean declares the output generator to be used which is, in this case, a `DSpaceWorkspaceItem` generator. It accepts two properties:

a) outputMap: A map from the intermediate keys to the DSpace metadata schema fields. The table below displays the default output mapping. As you can see, some fields, while they are read from the input source, are not output in DSpace since there are no default metadata schema fields to host them. However, if you create the corresponding metadata field registry, you can come back in this configuration to add a map between the input field key and the DSpace metadata field.

b) extraMetadataToKeep: A list of DSpace metadata schema fields to keep in the output

The following table presents the available keys from the online services, the keys that BTE uses in phase1 and the final output map to DSpace metadata fields.

Arxiv	PubMed	CrossRef	CiNii	BTE Key (phase 1)	Extra Keys created by BTE (phase 2)
title	articleTitle	articleTitle	title	title	
published	pubDate	year	issued	issued	
id				url	
summary	abstractText		description	abstract	
comment				note	
pdfUrl				fulltextUrl	
doi	doi	doi		doi	
journalRef	journalTitle	journalTitle	journal	journal	

Arxiv	PubMed	CrossRef	CiNii	BTE Key (phase 1)	Extra Keys created by BTE (phase 2)
author	author	authors	authors	authors	
authorWithAffiliation				authorsWithAffiliation	
primaryCategory				arxivCategory	
category				arxivCategory	
	pubmedID			pubmedID	
	publicationStatus			publicationStatus	
	pubModel				
	printISSN	printISSN	issn	jissn	
	electronicISSN	electronicISSN		jeissn	
	journalVolume	volume	volume	volume	
	journalIssue	issue	issue	issue	
	language		language	language	
	publicationType	doiType		subtype	
	primaryKeyword		subjects	keywords	allkeywords
	secondaryKeyword			keywords	allkeywords
	primaryMeshHeading			mesh	allkeywords
	secondaryMeshHeading			mesh	allkeywords
	startPage	firstPage	spage	firstpage	
	endPage	lastPage	epage	lastpage	
		printISBN		pisbn	
		electronicISBN		eisbn	
		editionNumber		editionnumber	
		seriesTitle		seriestitle	

Arxiv	PubMed	CrossRef	CiNii	BTE Key (phase 1)	Extra Keys created by BTE (phase 2)
		volumeTitle		volumetitle	
		publicationType			
		editors		editors	
		translators		translators	
		chairs		chairs	
			naid	naid	
			ncid	ncid	
			publisher	publisher	



I can see more beans in the configuration file that are not explained above. Why is this?

The configuration file hosts options for two services. [BatchImport service](#) and [SubmissionLookup service](#). Thus, some beans that are not used in the first service, are not mentioned in this documentation. However, since both services are based on the BTE, some beans are used by both services.

4.3.2 Configurable Workflow

- 1 [Introduction](#)
- 2 [Instructions for Enabling Configurable Reviewer Workflow in XMLUI](#)
 - 2.1 [dspace/config/xmlui.xconf](#)
 - 2.2 [dspace/config/modules/workflow.cfg](#)
- 3 [Data Migration](#)
 - 3.1 [Workflowitem conversion/migration scripts](#)
 - 3.1.1 [Automatic migration](#)
 - 3.1.2 [Java based migration](#)
- 4 [Configuration](#)
 - 4.1 [DSpace.cfg configuration](#)
 - 4.2 [Main workflow configuration](#)

- 4.2.1 [workflow-map](#)
- 4.2.2 [workflow](#)
- 4.2.3 [roles](#)
- 4.2.4 [step](#)
- 4.3 [Workflow actions configuration](#)
 - 4.3.1 [API configuration](#)
 - 4.3.1.1 [User Selection Action](#)
 - 4.3.1.2 [Processing Action](#)
 - 4.3.2 [User Interface configuration](#)
- 5 [Authorizations](#)
- 6 [Database](#)
 - 6.1 [cwf_workflowitem](#)
 - 6.2 [cwf_collectionrole](#)
 - 6.3 [cwf_workflowitemrole](#)
 - 6.4 [cwf_pooltask](#)
 - 6.5 [cwf_claimtask](#)
 - 6.6 [cwf_in_progress_user](#)
- 7 [Additional workflow steps/actions and features](#)
 - 7.1 [Optional workflow steps: Select single reviewer workflow](#)
 - 7.2 [Optional workflow steps: Score review workflow](#)
 - 7.3 [Workflow overview features](#)
- 8 [Known Issues](#)
 - 8.1 [Curation System](#)
 - 8.2 [Existing issues](#)

Introduction

Configurable Workflows are an optional feature that may be enabled for use only within DSpace XMLUI.

The primary focus of the workflow framework is to create a more flexible solution for the administrator to configure, and even to allow an application developer to implement custom steps, which may be configured in the workflow for the collection through a simple configuration file. The concept behind this approach was modeled on the configurable submission system already present in DSpace.

For more information, see the [Configurable Workflow Introductory Video](#)

Instructions for Enabling Configurable Reviewer Workflow in XMLUI



Please note that enabling the Configurable Reviewer Workflow makes changes to the structure of your database that are currently irreversible in any graceful manner, so please **backup your database** in

advance to allow you to restore to that point should you wish to do so. It should also be noted that only the XMLUI has been changed to cope with the database changes. The JSPUI will no longer work if the Configurable Reviewer Workflow is enabled.

dspace/config/xmlui.xconf

The submission aspect has been split up into multiple aspects: one submission aspect for the submission process, one workflow aspect containing the code for the original workflow and one xmlworkflow aspect containing the code for the new XML configurable workflow framework. In order to enable one of the two aspects, either the workflow or xmlworkflow aspect should be enabled in the `[dspace]/config/xmlui.xconf` configuration file. This means that the `xmlui.xconf` configuration for the original workflow is the following:

```
<aspect name="Submission and Workflow" path="resource://aspects/Submission/" />
<aspect name="Original Workflow" path="resource://aspects/Workflow/" />
```

And the `xmlui.xconf` configuration for the new XML configurable workflow is the following:

```
<aspect name="Submission and Workflow" path="resource://aspects/Submission/" />
<aspect name="XMLWorkflow" path="resource://aspects/XMLWorkflow/" />
```

dspace/config/modules/workflow.cfg

Besides that, a workflow configuration file has been created that specifies the workflow that will be used in the back-end of the DSpace code. It is important that the option selected in this configuration file matches the aspect that was enabled. The workflow configuration file is available in `[dspace]/config/modules/workflow.cfg`. This configuration file has been added because it is important that a CLI import process uses the correct workflow and this should not depend on the UI configuration. The `workflow.cfg` configuration file contains the following property:

```
# Original Workflow
#workflow.framework: originalworkflow
#XML configurable workflow
workflow.framework: xmlworkflow
```



Workflow Data Migration

You will also need to follow the [Data Migration Procedure](#) below.

Data Migration

⊖ Please note that enabling the Configurable Reviewer Workflow makes changes to the structure of your database that are currently irreversible in any graceful manner, so please **backup your database** in advance to allow you to restore to that point should you wish to do so. It should also be noted that only the XMLUI has been changed to cope with the database changes. The JSPUI will no longer work if the Configurable Reviewer Workflow is enabled.

Workflowitem conversion/migration scripts

Depending on the workflow that is used by a DSpace installation, different scripts can be used when migrating to the new workflow.

Automatic migration

Automatic migration can be used when the out of the box original workflow framework is used by your DSpace installation. This means that your DSpace installation uses the workflow steps and roles that are available out of the box. The automated migration will migrate the policies, roles, tasks and workflowitems from the original workflow to the new workflow framework.

This process will occur automatically by simply restarting Tomcat (or your servlet container) after enabling "xmlworkflow" in the `workflow.cfg`.

You can also choose to manually kick off this migration by simply running:

```
[dspace]/bin/dspace database migrate
```

For more information on the "database migrate" command, please see [Database Utilities](#).

Java based migration

In case your DSpace installation uses a customized version of the workflow, the migration script might not work properly and a different approach is recommended. Therefore, an additional Java based script has been created that restarts the workflow for all the workflowitems that exist in the original workflow framework. The script will take all the existing workflowitems and place them in the first step of the XML configurable workflow framework thereby taking into account the XML configuration that exists at that time for the collection to which the item has been submitted. This script can also be used to restart the workflow for workflowitems in the original workflow but not to restart the workflow for items in the XML configurable workflow.

To execute the script, run the following CLI command:

```
[dspace]/bin/dspace dsrun org.dspace.xmlworkflow.migration.RestartWorkflow -e admin@myrespository.org
```

The following arguments can be specified when running the script:

- `-e`: specifies the username of an administrator user

- -n: if sending submissions through the workflow, send notification emails
- -p: the provenance description to be added to the item
- -h: help

Configuration

DSpace.cfg configuration

The workflow configuration file is available in `[dspace]/config/modules/workflow.cfg`. This configuration file has been added because it is important that a CLI import process uses the correct workflow and this should not depend on the UI configuration. The `workflow.cfg` configuration file contains the following property:

```
# Original Workflow
#workflow.framework: originalworkflow
#XML configurable workflow
workflow.framework: xmlworkflow
```

Main workflow configuration

The workflow main configuration can be found in the `workflow.xml` file, located in `[dspace]/config/workflow.xml`. An example of this workflow configuration file can be found below.

```
<?xml version="1.0" encoding="UTF-8"?>
<wf-config>
  <workflow-map>
    <!-- collection to workflow mapping -->
    <name-map collection="default" workflow="{workflow.id}"/>
    <name-map collection="123456789/0" workflow="{workflow.id2}"/>
  </workflow-map>
  <workflow start="{start.step.id}" id="{workflow.id}">
    <roles>
      <!-- Roles used in the workflow -->
    </roles>
    <!-- Steps come here-->
    <step id="ExampleStep1" nextStep="ExampleStep2" userSelectionMethod="{
UserSelectionActionId}">
      <!-- Step1 config-->
    </step>
    <step id="ExampleStep2" userSelectionMethod="{UserSelectionActionId}">
    </step>
  </workflow>
  <workflow start="{start.step.id2}" id="{workflow.id}">
    <!-- Another workflow configuration-->
  </workflow>
</wf-config>
```

workflow-map

The workflow map contains a mapping between collections in DSpace and a workflow configuration. Similar to the configuration of the submission process, the mapping can be done based on the handle of the collection. The mapping with "default" as the value for the collection mapping, will be used for the collections not occurring in other mapping tags. Each mapping is defined by a "name-map" tag with two attributes:

- collection: can either be a collection handle or "default"
- workflow: the value of this attribute points to one of the workflow configurations defined by the "workflow" tags

workflow

The workflow element is a repeatable XML element and the configuration between two "workflow" tags represents one workflow process. It requires the following 2 attributes:

- id: a unique identifier used for the identification of the workflow and used in the workflow to collection mapping
- start: the identifier of the first step of the workflow, this will be the entry point of this workflow-process. When a new item has been committed to a collection that uses this workflow, the step configured in the "start" attribute will be the first step the item will go through.

roles

Each workflow process has a number of roles defined between the "roles" tags. A role represents one or more DSpace EPersons or Groups and can be used to assign them to one or more steps in the workflow process. One role is represented by one "role" tag and has the following attributes:

- id: a unique identifier (in one workflow process) for the role
- description: optional attribute to describe the role
- scope: optional attribute that is used to find our group and must have one of the following values:
 - collection: The collection value specifies that the group will be configured at the level of the collection. This type of groups is the same as the type that existed in the original workflow system. In case no value is specified for the scope attribute, the workflow framework assumes the role is a collection role.
 - repository: The repository scope uses groups that are defined at repository level in DSpace. The name attribute should exactly match the name of a group in DSpace.
 - item: The item scope assumes that a different action in the workflow will assign a number of EPersons or Groups to a specific workflow-item in order to perform a step. These assignees can be different for each workflow item.
- name: The name specified in the name attribute of a role will be used to lookup the in DSpace. The lookup will depend on the scope specified in the "scope" attribute:
 - collection: The workflow framework will look for a group containing the name specified in the name attribute and the ID of the collection for which this role is used.
 - repository: The workflow framework will look for a group with the same name as the name specified in the name attribute

- item: in case the item scope is selected, the name of the role attribute is not required
- internal: optional attribute which isn't really used at the moment, false by default

```

<roles>
  <role id="{unique.role.id}" description="{role.description}" scope="{role.scope}" name="{
role.name}" internal="true/false"/>
</roles>

```

step

The step element represents one step in the workflow process. A step represents a number of actions that must be executed by one specified role. In case no role attribute is specified, the workflow framework assumes that the DSpace system is responsible for the execution of the step and that no user interface will be available for each of the actions in this step. The step element has the following attributes in order to further configure it:

- id: The id attribute specifies a unique identifier for the step, this id will be used when configuring other steps in order to point to this step. This identifier can also be used when configuring the start step of the workflow item.
- nextStep: This attribute specifies the step that will follow once this step has been completed under normal circumstances. If this attribute is not set, the workflow framework will assume that this step is an endpoint of the workflow process and will archive the item in DSpace once the step has been completed.
- userSelectionMethod: This attribute defines the UserSelectionAction that will be used to determine how to attach users to this step for a workflow-item. The value of this attribute must refer to the identifier of an action bean in the workflow-actions.xml. Examples of the user attachment to a step are the currently used system of a task pool or as an alternative directly assigning a user to a task.
- role: optional attribute that must point to the id attribute of a role element specified for the workflow. This role will be used to define the epersons and groups used by the userSelectionMethod.
- RequiredUsers

```

<step id="{step.id}" nextStep="{next.step.id}" userSelectionMethod="{user.selection.bean.id}" role=
"{role.id}" >
<!-- optional alternate outcomes, depending on the outcome of the actions you can alter the next
step here -->
<alternativeOutcome>
  <step status="{integer}">{alternate.step.id}</step>
</alternativeOutcome>
<action id="{action.bean.id}"/>
<action id="{action.bean.id.1}"/>
</step>

```

Each step contains a number of actions that the workflow item will go through. In case the action has a user interface, the users responsible for the execution of this step will have to execute these actions before the workflow item can proceed to the next action or the end of the step.

There is also an optional subsection that can be defined for a step part called "alternativeOutcome". This can be used to define outcomes for the step that differ from the one specified in the nextStep attribute. Each action returns an integer depending on the result of the action. The default value is "0" and will make the workflow item proceed to the next action or to the end of the step.

In case an action returns a different outcome than the default "0", the alternative outcomes will be used to lookup the next step. The alternativeOutcome element contains a number of steps, each having a status attribute. This status attribute defines the return value of an action. The value of the element will be used to lookup the next step the workflow item will go through in case an action returns that specified status.

Workflow actions configuration

API configuration

The workflow actions configuration is located in the [dspace]/config/spring/api/ directory and is named "workflow-actions.xml". This configuration file describes the different Action Java classes that are used by the workflow framework. Because the workflow framework uses Spring framework for loading these action classes, this configuration file contains Spring configuration.

This file contains the beans for the actions and user selection methods referred to in the workflow.xml. In order for the workflow framework to work properly, each of the required actions must be part of this configuration.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
  xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:util="http://www.springframework.org/schema/util"
  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/
  schema/beans/spring-beans-2.0.xsd
                        http://www.springframework.org/schema/util http://
  www.springframework.org/schema/util/spring-util-2.0.xsd">
  <!-- At the top are our bean class identifiers --->
  <bean id="{action.api.id}" class="{class.path}" scope="prototype"/>
  <bean id="{action.api.id.2}" class="{class.path}" scope="prototype"/>
  <!-- Below the class identifiers come the declarations for out actions/userSelectionMethods -->
  <!-- Use class workflowActionConfig for an action -->
  <bean id="{action.id}" class="org.dspace.xmlworkflow.state.actions.WorkflowActionConfig" scope="
  prototype">
    <constructor-arg type="java.lang.String" value="{action.id}"/>
    <property name="processingAction" ref="{action.api.id}"/>
    <property name="requiresUI" value="{true/false}"/>
  </bean>
  <!-- Use class UserSelectionActionConfig for a user selection method -->
  <!--User selection actions-->
  <bean id="{action.api.id.2}" class="org.dspace.xmlworkflow.state.actions.UserSelectionActionConfig
  " scope="prototype">
    <constructor-arg type="java.lang.String" value="{action.api.id.2}"/>
    <property name="processingAction" ref="{user.selection.bean.id}"/>
    <property name="requiresUI" value="{true/false}"/>
  </bean>
```

```
</beans>
```

Two types of actions are configured in this Spring configuration file:

- **User selection action:** This type of action is always the first action of a step and is responsible for the user selection process of that step. In case a step has no role attached, no user will be selected and the `NoUserSelectionAction` is used.
- **Processing action:** This type of action is used for the actual processing of a step. Processing actions contain the logic required to execute the required operations in each step. Multiple processing actions can be defined in one step. These user and the workflow item will go through these actions in the order they are specified in the workflow configuration unless an alternative outcome is returned by one of them.

User Selection Action

Each user selection action that is used in the workflow config refers to a bean definition in this `workflow-actions.xml` configuration. In order to create a new user selection action bean, the following XML code is used:

```
<bean id="{action.api.id.2}" class="org.dspace.xmlworkflow.state.actions.UserSelectionActionConfig"
  scope="prototype">
  <constructor-arg type="java.lang.String" value="{action.api.id.2}"/>
  <property name="processingAction" ref="{user.selection.bean.id}"/>
  <property name="requiresUI" value="{true/false}"/>
</bean>
```

This bean defines a new `UserSelectionActionConfig` and the following child tags:

- **constructor-arg:** This is a constructor argument containing the ID the task. This is the same as the `id` attribute of the bean and is used by the workflow config to refer to this action.
- **property processingAction:** This tag refers the the ID of the API bean, responsible for the implementation of the API side of this action. This bean should also be configured in this XML.
- **property requiresUI:** In case this property is true, the workflow framework will expect a user interface for the action. Otherwise the framework will automatically execute the action and proceed to the next one.

Processing Action

Processing actions are configured similar to the user selection actions. The only difference is that these processing action beans are implementations of the `WorkflowActionConfig` class instead of the `UserSelectionActionConfig` class.

User Interface configuration

The configuration file for the workflow user interface actions is located in the `[dspace]/config/spring/xmlui/` and is named "`workflow-actions-xmlui.xml`". Each bean defined here has an `id` which is the action identifier and the class is a classpath which links to the `xmlui` class responsible for generating the User Interface side of the workflow action. Each of the class defined here must extend the `org.dspace.app.xmlui.aspect.submission.workflow.AbstractXMLUIAction` class, this class

contains some basic settings for an action and has a method called `addWorkflowItemInformation()` which will render the given item with a show full link so you don't have to write the same code in each of your actions if you want to display the item. The `id` attribute used for the beans in the configuration must correspond to the `id` used in the workflow configuration. In case an action requires a User Interface class, the workflow framework will look for a UI class in this configuration file.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
  xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:util="http://www.springframework.org/schema/util"
  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/
schema/beans/spring-beans-2.0.xsd
                      http://www.springframework.org/schema/util http://
www.springframework.org/schema/util/spring-util-2.0.xsd">
  <bean id="{action.id}" class="{classpath}" scope="prototype"/>
  <bean id="{action.id.2}" class="{classpath}" scope="prototype"/>
</beans>
```

Authorizations

Currently, the authorizations are always granted and revoked based on the tasks that are available for certain users and groups. The types of authorization policies that is granted for each of these is always the same:

- READ
- WRITE
- ADD
- DELETE

Database

The workflow uses a separate metadata schema named `workflow`. The fields this schema contains can be found in the `[dspace]/config/registries` directory and in the file `workflow-types.xml`. This schema is only used when using the score reviewing system at the moment, but one could always use this schema if metadata is required for custom workflow steps.

The following tables have been added to the DSpace database. All tables are prefixed with 'cwf_' to avoid any confusion with the existing workflow related database tables:

cwf_workflowitem

The `cwf_workflowitem` table contains the different workflowitems in the workflow. This table has the following columns:

- `workflowitem_id`: The identifier of the workflowitem and primary key of this table
- `item_id`: The identifier of the DSpace item to which this workflowitem refers.

- `collection_id`: The collection to which this workflowitem is submitted.
- `multiple_titles`: Specifies whether the submission has multiple titles (important for submission steps)
- `published_before`: Specifies whether the submission has been published before (important for submission steps)
- `multiple_files`: Specifies whether the submission has multiple files attached (important for submission steps)

cwf_collectionrole

The `cwf_collectionrole` table represents a workflow role for one collection. This type of role is the same as the roles that existed in the original workflow meaning that for each collection a separate group is defined to describe the role. The `cwf_collectionrole` table has the following columns:

- `collectionrol_id`: The identifier of the collectionrole and the primary key of this table
- `role_id`: The identifier/name used by the workflow configuration to refer to the collectionrole
- `collection_id`: The collection identifier for which this collectionrole has been defined
- `group_id`: The group identifier of the group that defines the collection role

cwf_workflowitemrole

The `cwf_workflowitemrole` table represents roles that are defined at the level of an item. These roles are temporary roles and only exist during the execution of the workflow for that specific item. Once the item is archived, the `workflowitemrole` is deleted. Multiple rows can exist for one `workflowitem` with e.g. one row containing a group and a few containing `epersons`. All these rows together make up the `workflowitemrole`. The `cwf_workflowitemrole` table has the following columns:

- `workflowitemrole_id`: The identifier of the `workflowitemrole` and the primary key of this table
- `role_id`: The identifier/name used by the workflow configuration to refer to the `workflowitemrole`
- `workflowitem_id`: The `cwf_workflowitem` identifier for which this `workflowitemrole` has been defined
- `group_id`: The group identifier of the group that defines the `workflowitemrole` role
- `eperson_id`: The `eperson` identifier of the `eperson` that defines the `workflowitemrole` role

cwf_pooltask

The `cwf_pooltask` table represents the different task pools that exist for a `workflowitem`. These task pools can be available at the beginning of a step and contain all the users that are allowed to claim a task in this step. Multiple rows can exist for one task pool containing multiple groups and `epersons`. The `cwf_pooltask` table has the following columns:

- `pooltask_id`: The identifier of the `pooltask` and the primary key of this table
- `workflowitem_id`: The identifier of the `workflowitem` for which this task pool exists
- `workflow_id`: The identifier of the workflow configuration used for this `workflowitem`
- `step_id`: The identifier of the step for which this task pool was created
- `action_id`: The identifier of the action that needs to be displayed/executed when the user selects the task from the task pool
- `eperson_id`: The identifier of an `eperson` that is part of the task pool

- `group_id`: The identifier of a group that is part of the task pool

cwf_claimtask

The `cwf_claimtask` table represents a task that has been claimed by a user. Claimed tasks can be assigned to users or can be the result of a claim from the task pool. Because a step can contain multiple actions, the claimed task defines the action at which the user has arrived in a particular step. This makes it possible to stop working halfway the step and continue later. The `cwf_claimtask` table contains the following columns:

- `claimtask_id`: The identifier of the claimtask and the primary key of this table
- `workflowitem_id`: The identifier of the workflowitem for which this task exists
- `workflow_id`: The id of the workflow configuration that was used for this workflowitem
- `step_id`: The step that is currently processing the workflowitem
- `action_id`: The action that should be executed by the owner of this claimtask
- `owner_id`: References the eperson that is responsible for the execution of this task

cwf_in_progress_user

The `cwf_in_progress_user` table keeps track of the different users that are performing a certain step. This table is used because some steps might require multiple users to perform the step before the workflowitem can proceed . The `cwf_in_progress_user` table contains the following columns:

- `in_progress_user_id`: The identifier of the in progress user and the primary key of this table
- `workflowitem_id`: The identifier of the workflowitem for which the user is performing or has performed the step.
- `user_id`: The identifier of the eperson that is performing or has performed the task
- `finished`: Keeps track of the fact that the user has finished the step or is still in progress of the execution

Additional workflow steps/actions and features

Optional workflow steps: Select single reviewer workflow

This workflow makes it possible to assign a single user to review an item. This workflow configuration skips the task pool option meaning that the assigned reviewer no longer needs to claim the task. The configuration consists of the following 2 steps.

- **AssignStep**: During the assignstep, a user has the ability to select a responsible user to review the workflowitem. This means that for each workflowitem, a different user can be selected. Because a user is assigned, the task pool is no longer required.
- **ReviewStep**: The start of the reviewstep is different than the typical task pool. Instead of having a task pool, the user will be automatically assigned to the task. However, the user still has the option to reject the task (in case he or she is not responsible for the assigned task) or review the item. In case the user rejects the task, the workflowitem will be sent to the another step in the workflow as an alternative to the default outcome.

Optional workflow steps: Score review workflow

The score review system allows reviewers to give the reviewed item a rating. Depending on the results of the rating, the item will be approved to go to the next workflow step or will be sent to an alternative step. The score review workflow consists of the following 2 steps.

- **ScoreReviewStep:** The group of responsible users for the score reviewing will be able to claim the task from the taskpool. Depending on the configuration, a different number of users can be required to execute the task. This means that the task will be available in the task pool until the required number of users has at least claimed the task. Once everyone of them has finished the task, the next (automatic) processing step is activated.
- **EvaluationStep:** During the evaluationstep, no user interface is required. The workflow system will automatically execute the step that evaluates the different scores. In case the average score is more than a configurable percentage, the item is approved, otherwise it is rejected.

Workflow overview features

A new features has been added to the XML based workflow that resembles the features available in the JSPUI of DSpace that allows administrators to abort workflowitems. The feature added to the XMLUI allows administrators to look at the status of the different workflowitems and look for workflowitems based on the collection to which they have been submitted. Besides that, the administrator has the ability to permanently delete the workflowitem or send the item back to the submitter.

Known Issues

Curation System

The DSpace 1.7 version of the curation system integration into the original DSpace workflow only exists in the `WorkflowManager.advance()` method. Before advancing to the next workflow step or archiving the Item, a check is performed to see whether any curation tasks need to be executed/scheduled. The problem is that this check is based on the hardcoded workflow steps that exist in the original workflow. These hardcoded checks are done in the `WorkflowCurator` and will need to be changed.

Existing issues

- What happens with collection roles after config changes
- What with workflowitems after config changes
- What with undefined outcomes
- Config checker
- Configurable authorizations?

4.3.3 Importing and Exporting Content via Packages

- 1 [Package Importer and Exporter](#)
 - 1.1 [Supported Package Formats](#)
 - 1.2 [Ingesting](#)
 - 1.2.1 [Ingestion Modes & Options](#)
 - 1.2.1.1 [Ingesting a Single Package](#)
 - 1.2.1.2 [Ingesting Multiple Packages at Once](#)
 - 1.2.2 [Restoring/Replacing using Packages](#)
 - 1.2.2.1 [Default Restore Mode](#)
 - 1.2.2.2 [Restore, Keep Existing Mode](#)
 - 1.2.2.3 [Force Replace Mode](#)
 - 1.3 [Disseminating](#)
 - 1.3.1 [Disseminating a Single Object](#)
 - 1.3.2 [Disseminating Multiple Objects at Once](#)
 - 1.4 [Archival Information Packages \(AIPs\)](#)
 - 1.5 [METS packages](#)

Package Importer and Exporter

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

To see all the options, invoke it as:

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

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

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

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

Supported Package Formats

DSpace comes with several pre-configured package ingestion and dissemination plugins, which allow you to import/export content in a variety of formats.

Pre-Configured Submission Package (SIP) Types

- AIP - Ingests content which is in the [DSpace Archival Information Package \(AIP\) format](#). This is used as part of the DSpace [AIP Backup and Restore](#) process
- DSPACE-ROLES - Ingests DSpace users/groups in the [DSPACE-ROLES XML Schema](#). This is primarily used by the DSpace [AIP Backup and Restore](#) process to ingest/replace DSpace Users & Groups.
- METS - Ingests content which is in the [DSpace METS SIP format](#)
- PDF - Ingests a single PDF file (where basic metadata is extracted from the file properties in the PDF Document).

Pre-Configured Dissemination Package (DIP) Types

- AIP - Exports content which is in the [DSpace Archival Information Package \(AIP\) format](#). This is used as part of the DSpace [AIP Backup and Restore](#) process
- DSPACE-ROLES - Exports DSpace users/groups in the [DSPACE-ROLES XML Schema](#). This is primarily used by the DSpace [AIP Backup and Restore](#) process to export DSpace Users & Groups.
- METS - Exports content in the [DSpace METS SIP format](#)

For a list of all package ingestion and dissemination plugins that are currently installed in your DSpace, you can execute:

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

Some packages ingestion and dissemination plugins also have custom options/parameters. For example, to see a listing of the custom options for the "METS" plugin, you can execute:

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

Ingesting

Ingestion Modes & Options

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

1. Submit/Ingest Mode (`-s` option, default) – submit package to DSpace in order to create a new object(s)
2. Restore Mode (`-r` option) – restore pre-existing object(s) in DSpace based on package(s). This also attempts to restore all handles and relationships (parent/child objects). This is a specialized type of "submit", where the object is created with a known Handle and known relationships.
3. Replace Mode (`-r -f` option) – replace existing object(s) in DSpace based on package(s). This also attempts to restore all handles and relationships (parent/child objects). This is a specialized type of "restore" where the contents of existing object(s) is replaced by the contents in the AIP(s). By default, if a normal "restore" finds the object already exists, it will back out (i.e. rollback all changes) and report which object already exists.

Ingesting a Single Package

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

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

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

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

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

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

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

Ingesting Multiple Packages at Once

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

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

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

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

for example:

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

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



Not All Packers Support Bulk Ingest

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

- METS Packager Plugin
- [AIP Packager Plugin](#)

Restoring/Replacing using Packages

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

There are currently three restore modes:

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

Default Restore Mode

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

Use this 'packager' command template:

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

For example:


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

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

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

Restore, Keep Existing Mode

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

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

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

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

For example:

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

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

Force Replace Mode

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

Potential for Data Loss

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

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

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

For example:

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

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

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

Disseminating

Disseminating a Single Object

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

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

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

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

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

Disseminating Multiple Objects at Once

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

For example, use this 'packager' command template:

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

for example:

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

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

Archival Information Packages (AIPs)

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

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

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

For more information, see the section on [AIP backup & Restore for DSpace](#).

METS packages

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

The DSpace METS SIP profile is available at: [DSpaceMETSSIPProfile](#)

4.3.4 Importing and Exporting Items via Simple Archive Format

- 1 [Item Importer and Exporter](#)
 - 1.1 [DSpace Simple Archive Format](#)
 - 1.2 [Configuring metadata_\[prefix\].xml for Different Schema](#)
 - 1.3 [Importing Items](#)
 - 1.3.1 [Adding Items to a Collection from a directory](#)
 - 1.3.2 [Adding Items to a Collection from a zipfile](#)
 - 1.3.3 [Replacing Items in Collection](#)
 - 1.3.4 [Deleting or Unimporting Items in a Collection](#)
 - 1.3.5 [Other Options](#)
 - 1.3.6 [UI Batch Import \(JSPUI\)](#)
 - 1.3.7 [UI Batch Import \(XMLUI\)](#)
 - 1.4 [Exporting Items](#)

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. Apart from the offered functionality, these tools serve as an example for users who aim to implement their own item importer.

DSpace Simple Archive Format

The basic concept behind the DSpace's Simple Archive Format is to create an archive, which is a directory containing one subdirectory per item. Each item directory contains a file for the item's descriptive metadata, and the files that make up the item.

```
archive_directory/
  item_000/
    dublin_core.xml      -- qualified Dublin Core metadata for metadata fields belonging to
the dc schema
    metadata_[prefix].xml -- metadata in another schema, the prefix is the name of the schema
as registered with the metadata registry
    contents             -- text file containing one line per filename
    collections          -- text file that contains the handles of the collections the
item will belong two. Optional. Each handle in a row.
                        -- Collection in first line will be the owning collection
    file_1.doc           -- files to be added as bitstreams to the item
    file_2.pdf
  item_001/
    dublin_core.xml
    contents
    file_1.png
    ...
```

The `dublin_core.xml` or `metadata_[prefix].xml` file has the following format, where each metadata element has its own entry within a `<dcvalue>` tagset. There are currently three tag attributes available in the `<dcvalue>` tagset:

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

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

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

Every metadata field used, must be registered via the metadata registry of the DSpace instance first, see [Metadata and Bitstream Format Registries](#).



Recommended Metadata

It is recommended to minimally provide "dc.title" and, where applicable, "dc.date.issued". Obviously you can (and should) provide much more detailed metadata about the Item. For more information see: [Metadata Recommendations](#).

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

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

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

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

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

Where `\t` is the tab character.

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

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

'DESCRIPTION' is text of the files description.

Primary is used to specify the primary bitstream.

Configuring metadata_[prefix].xml for Different Schema

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

1. Create a separate file for the other schema named `metadata_[prefix].xml`, where the `[prefix]` is replaced with the schema's prefix.

2. Inside the xml file use the same Dublin Core *syntax*, but on the `<dublin_core>` element include the attribute `schema=[prefix]`.
3. Here is an example for ETD metadata, which would be in the file `metadata_etd.xml`:

```

<?xml version="1.0" encoding="UTF-8"?>
<dublin_core schema="etd">
  <dcvalue element="degree" qualifier="department">Computer Science</dcvalue>
  <dcvalue element="degree" qualifier="level">Masters</dcvalue>
  <dcvalue element="degree" qualifier="grantor">Michigan Institute of Technology</dcvalue>
</dublin_core>

```

Importing Items

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

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

‡ These are mutually exclusive.

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

Adding Items to a Collection from a directory

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

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

At the command line:

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

or by using the short form:

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

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

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

Adding Items to a Collection from a zipfile

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

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

At the command line:

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

or by using the short form:

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

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

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

Replacing Items in Collection

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

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

Long form:

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

Deleting or Unimporting Items in a Collection

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

```
[dspace]/bin/dspace import -e joe@user.com -d -m mapfile
```

In long form:

```
[dspace]/bin/dspace import --eperson=joe@user.com --delete --mapfile mapfile
```

Other Options

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

- **Specifying the owning collection on a per-item basis from the command line administration tool**

If you omit the `-c` flag, which is otherwise mandatory, the `ItemImporter` searches for a file named "collections" in each item directory. This file should contain a list of collections, one per line, specified either by their handle, or by their internal db id. The `ItemImporter` then will put the item in each of the specified collections. The owning collection is the collection specified in the first line of the collections file. If both the `-c` flag is specified and the collections file exists in the item directory, the `ItemImporter` will ignore the collections file and will put the item in the collection specified on the command line. Since the collections file can differ between item directories, this gives you more fine-grained control of the process of batch adding items to collections.

- **Importing with BTE**

The `DSpaceOutputGenerator`, which writes the metadata into the DSpace Simple Archive Format, has been updated to produce the collections file, if a metadata field named "collections" (reserved word) exists in the original metadata. This is mainly applicable to the CSV input format which is more flexible, but could also be implemented with a Modifier that adds the "collections" field to each Record in the BTE pipeline.

Important note: an entry with the "collections" key should be in the output map that is used by the `DSpaceOutputGenerator`.

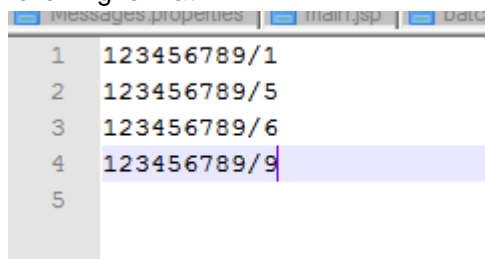
More info in [Importing Items via basic bibliographic formats \(Endnote, BibTex, RIS, TSV, CSV\) and online services \(OAI, arXiv, PubMed, CrossRef, CiNii\)](#).

UI Batch Import (JSPUI)

Batch import can also take place via the Administrator's UI. The steps to follow are:

A. Prepare the data

1. Items, i.e. the metadata and their bitstreams, must be in the Simple Archive Format describer earlier in this chapter. Thus, for each item there must be a separate directory that contains the corresponding files of the specific item.
2. Moreover, in each item directory, there can be another file that describes the collection or the collections that this item will be added to. The name of this file must be "collections" and it is optional. It has the following format:



```
Messages.properties | main.jsp | data
1 123456789/1
2 123456789/5
3 123456789/6
4 123456789/9
5
```

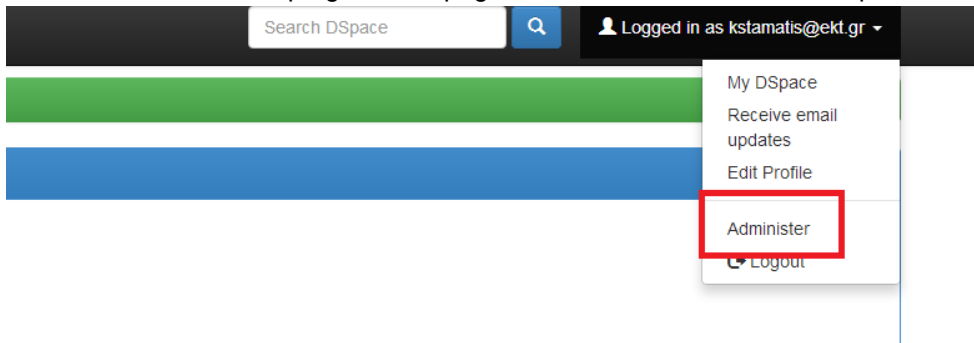
Each line contains the handle of the collection. The collection in the first line is the owning collection while the rest are the other collection the item should belong to.

3. Compress the item directories into a zip file. Please note that you need to zip the actual item directories and not just the directory that contains the item directories. Thus, the final zip file must directly contain the item directories.

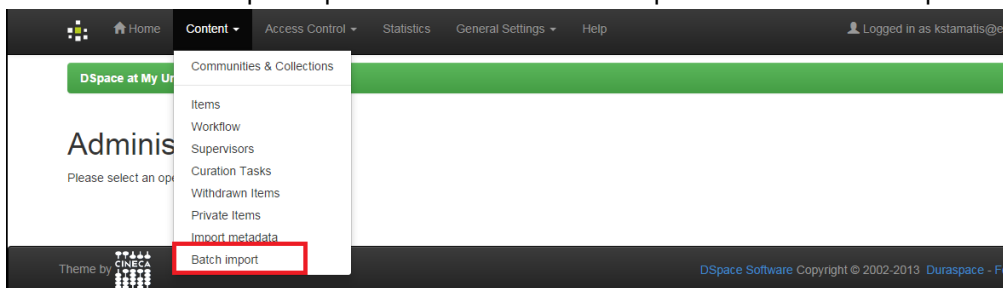
- Place the zip file in a public domain URL, like Dropbox or Google Drive or wherever you have access to do so. Since such a zip file can be very big in size, the batch import UI needs the URL to download it for a public location rather than just upload it and get a timeout exception

B. Import the items via the UI

- Login as an administrator
- Find the menu on the top right of the page, and select the "Administer" option

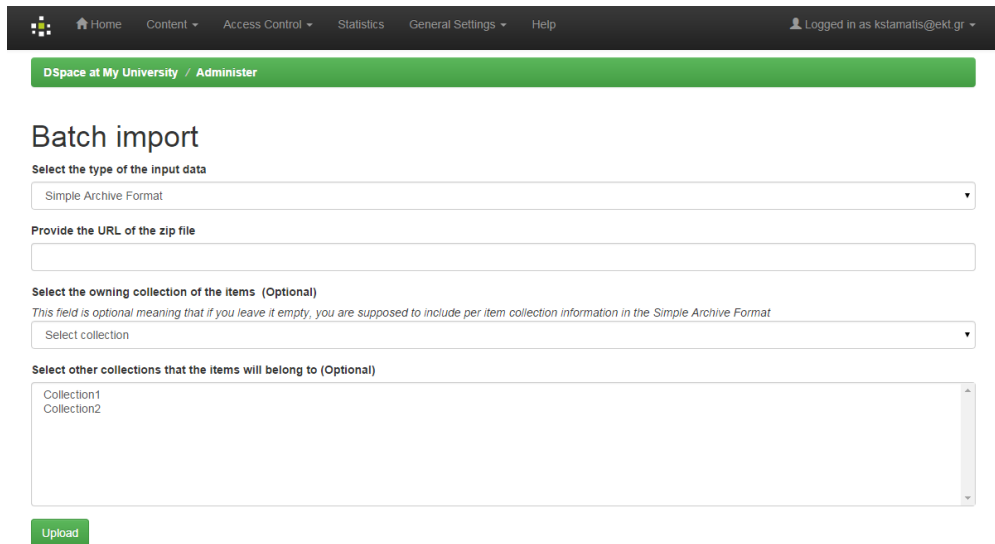


- Select the "Batch Import" option from the "Content" drop down menu on the top of the page



- Fill in the form that appears as follows:

- Field #1: select the type of the input data that you want to batch import. Be sure to select "Simple Archive Format" in this drop down menu
- Field #2: Copy/Paste the public URL where the zip file mentioned earlier is located
- Field #3: Select the owning collection of the items you are importing. This field is optional meaning that if you leave it empty, you are supposed to include per item collection information (via the "collections" file mentioned before) in the Simple Archive Format
- Field #4: Select the other collections the item will belong to. You can select more than one collection by just holding down the Ctrl key on your keyboard. If you select the owning collection in this multiselect input control, it will be ignored at the very end.



DSpace at My University / Administer

Batch import

Select the type of the input data

Simple Archive Format

Provide the URL of the zip file

Select the owning collection of the items (Optional)
This field is optional meaning that if you leave it empty, you are supposed to include per item collection information in the Simple Archive Format

Select collection

Select other collections that the items will belong to (Optional)

Collection1
Collection2

Upload

Comments:

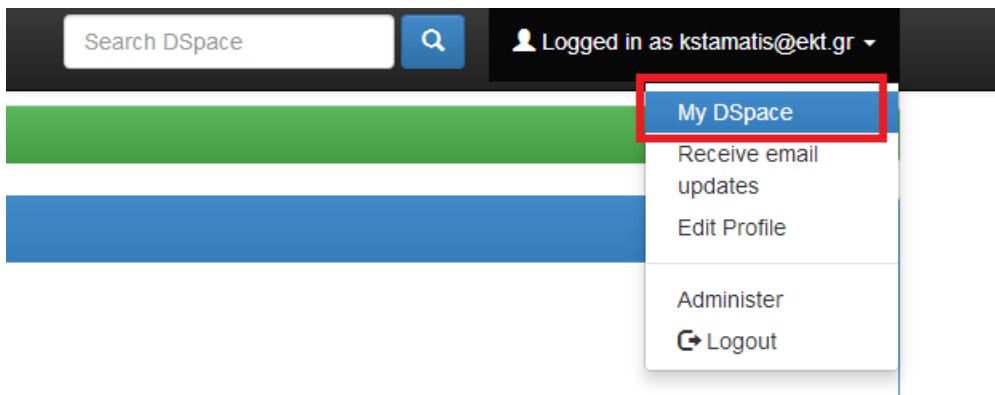
1) If you select an owning collection from this form, then the "collections" file that may be included in the item will be ignored.

2) If you do not specify an owning collection, and for some items no "collections" file exists in the item directory, then the item will not be imported in DSpace

Finally, when you submit the form you will receive a message informing you that the import process is being executed in the background (since it may take long). At the end, you will receive a success or failure email (to the email address of your DSpace account) informing you of the status of the import.

C. View past batch imports (that have be done via the UI)

1. Login
2. Visit "My DSpace" page

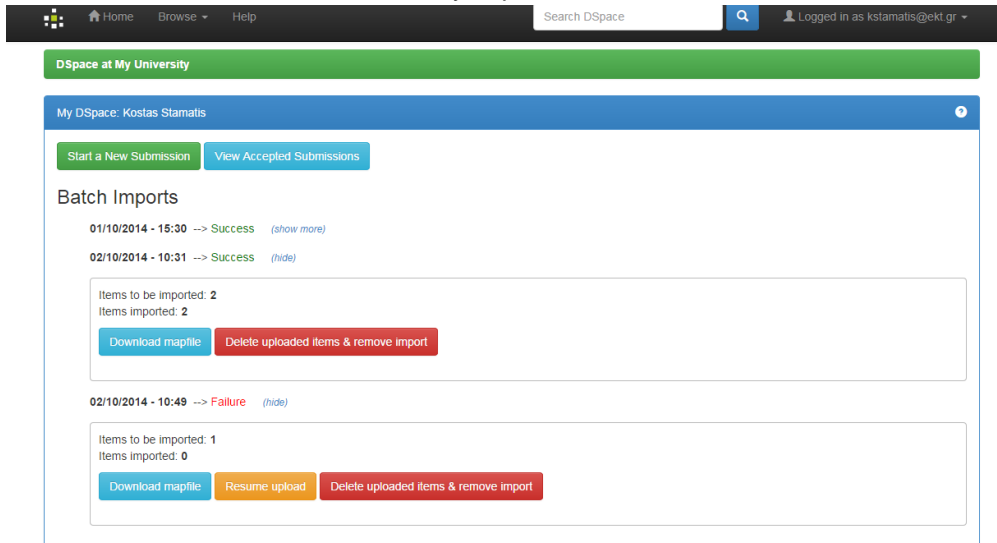


3. On the next page, you can see the history of batch imports. For each import, the following information is available:

The status of the batch import (success or failure)

The number of items that the user tried to import

The number of items that were actually imported



The screenshot shows the 'Batch Imports' section of a DSpace user interface. At the top, there are navigation links for 'Start a New Submission' and 'View Accepted Submissions'. Below this, the 'Batch Imports' section lists two entries:

- 01/10/2014 - 15:30** -> Success (show more)
- 02/10/2014 - 10:31** -> Success (hide)

The first entry (successful) shows 'Items to be imported: 2' and 'Items imported: 2'. It has two buttons: 'Download mapfile' and 'Delete uploaded items & remove import'.

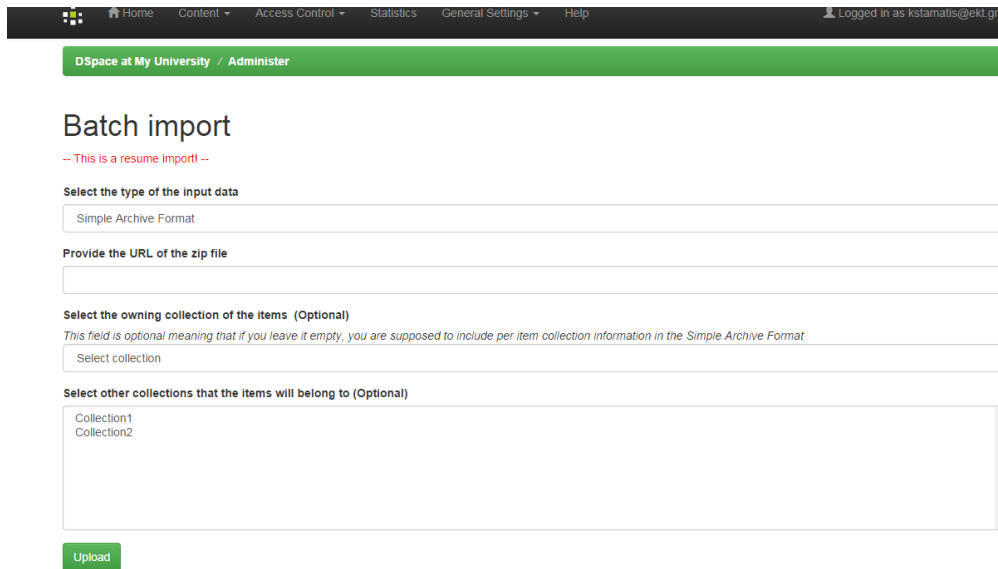
The second entry (failed) shows 'Items to be imported: 1' and 'Items imported: 0'. It has three buttons: 'Download mapfile', 'Resume upload', and 'Delete uploaded items & remove import'.

Moreover, the user can take the following actions:

Download the map file that was produced during the import. This file contains a list of items that were imported with the corresponding handle assigned to them by DSpace.

Delete the imported items. Everything that was imported will be deleted (including the history directory in the "[dspace]/import" directory)

In case of failure, the user can "Resume" the import. The user is taken to the upload form again, but the system recognizes the initial import (and the map file) in order to resume the old import. There is a red label in the form that informs the user about the "Resume" form.



The screenshot shows the 'Batch import' page in the DSpace administrator interface. At the top, there is a navigation bar with links for Home, Content, Access Control, Statistics, General Settings, and Help. The user is logged in as 'kstamatis@ekt.gr'. Below the navigation bar, the page title is 'Batch import' with a red message: '-- This is a resume import! --'. The form contains several sections: 'Select the type of the input data' with a dropdown menu set to 'Simple Archive Format'; 'Provide the URL of the zip file' with an empty text input field; 'Select the owning collection of the Items (Optional)' with a dropdown menu and a note: 'This field is optional meaning that if you leave it empty, you are supposed to include per item collection information in the Simple Archive Format'; and 'Select other collections that the items will belong to (Optional)' with a list box containing 'Collection1' and 'Collection2'. At the bottom of the form is a green 'Upload' button.

UI Batch Import (XMLUI)

A SimpleArchiveFormat package can be imported by an administrator in XMLUI. The SimpleArchiveFormat package needs to be compressed into a ZIP file, and it will be uploaded to XMLUI through the browser. DSpace will then process that ZIP, and ingest items into DSpace. A stable network connection is recommended, as your browser will need to upload a potentially large ZIP file, and then wait while DSpace processes that ZIP file.

While logged in as an administrator, click on Batch Import (ZIP):




Then, choose the owning collection from the collection dropdown, and browse to the ZIP file on your computer that has the SimpleArchiveFormat ZIP file.

Import Batch Load (ZIP)

Select a collection

Collection:

Select the collection you wish to submit an item to.

Special Collections > Postcards > Cape Cod Postcards 

ingest_jdame_20150106-153342.zip

Upload SimpleArchiveFormat ZIP

If successful, you will get a green message with a list of handles that were imported. It is what is considered the "map file".

Notice
Upload successful

```
folder_00000 123456789/13473 folder_00001 123456789/13474 folder_00003 123456789/13475 folder_00004
123456789/13476 folder_00005 123456789/13477 folder_00006 123456789/13478 folder_00007
123456789/13479 folder_00008 123456789/13480 folder_00009 123456789/13481 folder_00010
123456789/13482 folder_00011 123456789/13483 folder_00012 123456789/13484 folder_00013
123456789/13485 folder_00014 123456789/13486 folder_00015 123456789/13487 folder_00016
123456789/13488
```

If an error occurred, you will get a red error message with the issue:

Notice
Import failed

Invalid metadata field: [sd.specifications]

Exporting Items

The item exporter can export a single item or a collection of items, and creates a DSpace simple archive in [the aforementioned format](#) for each exported item. The items are exported in a sequential order in which they are retrieved from the database. As a consequence, the sequence numbers of the item subdirectories (item_000, item_001) are not related to DSpace handle or item ids.

Command used:	<code>[dspace]/bin/dspace export</code>
---------------	---

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

Exporting a Collection

The CLI command to export the items of a collection:

```
[dspace]/bin/dspace export --type=COLLECTION --id=collectionID_or_handle --dest=/path/to/destination --number=seq_num
```

Short form:

```
[dspace]/bin/dspace export -t COLLECTION -i collectionID_or_handle -d /path/to/destination -n seq_num
```

Exporting a Single Item

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

```
[dspace]/bin/dspace export --type=ITEM --id=itemID_or_handle --dest=/path/to/destination --number=seq_num
```

Short form:

```
[dspace]/bin/dspace export -t ITEM -i itemID_or_handle -d /path/to/destination -n 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.

The `-m` Argument

Using the `-m` argument will export the item/collection and also perform the migration step. It will perform the same process that the next section [Exchanging Content Between Repositories](#) performs. We recommend that section to be read in conjunction with this flag being used.

4.3.5 Registering Bitstreams via Simple Archive Format

- 1 [Overview](#)
 - 1.1 [Accessible Storage](#)
 - 1.2 [Registering Items Using the Item Importer](#)
 - 1.3 [Internal Identification and Retrieval of Registered Items](#)
 - 1.4 [Exporting Registered Items](#)
 - 1.5 [Deleting Registered Items](#)

Registering is not Importing

The procedures below will **not import** the actual bitstreams into DSpace. They will merely inform DSpace of an existing location where these Bitstreams can be found. Please refer to [Importing and Exporting Items via Simple Archive Format](#) for information on importing metadata and bitstreams.

Overview

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

Accessible Storage

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

Registering Items Using the Item Importer

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

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

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

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

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

where

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

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

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

(or by using the long form)

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

The `--workflow` and `--test` flags will function as described in [Importing Items](#).

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

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

Internal Identification and Retrieval of Registered Items

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

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

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

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

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

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

Exporting Registered Items


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

Deleting Registered Items

If a registered item is deleted from DSpace, (either interactively or by using the `--delete` or `--replace` flags described in [Importing and Exporting Items via Simple Archive Format](#)) the item will disappear from DSpace but its registered content files will remain in place just as they were prior to registration. Bitstreams not registered but added by DSpace as part of registration, such as `license.txt` files, will be deleted.

4.3.6 Importing Items via basic bibliographic formats (Endnote, BibTex, RIS, TSV, CSV) and online services (OAI, arXiv, PubMed, CrossRef, CiNii)

- 1 [About the Biblio-Transformation-Engine \(BTE\)](#)
 - 1.1 [BTE in DSpace](#)
 - 1.2 [BTE Configuration](#)
 - 1.3 [UI for administrators](#)
 - 1.4 [Case Studies](#)

 This functionality is an extension of that provided by [Importing and Exporting Items via Simple Archive Format](#) so please read that section before continuing. It is underpinned by the Biblio Transformation Engine (<https://github.com/EKT/Biblio-Transformation-Engine>)

About the Biblio-Transformation-Engine (BTE)

The BTE is a Java framework developed by the Hellenic National Documentation Centre (EKT, www.ekt.gr) and consists of programmatic APIs for filtering and modifying records that are retrieved from various types of data sources (eg. databases, files, legacy data sources) as well as for outputting them in appropriate standards formats (eg. database files, txt, xml, Excel). The framework includes independent abstract modules that are executed separately, offering in many cases alternative choices to the user depending on the input data set, the transformation workflow that needs to be executed and the output format that needs to be generated.

The basic idea behind the BTE is a standard workflow that consists of three steps, a data loading step, a processing step (record filtering and modification) and an output generation. A data loader provides the system

with a set of Records, the processing step is responsible for filtering or modifying these records and the output generator outputs them in the appropriate format.

The standard BTE version offers several predefined Data Loaders as well as Output Generators for basic bibliographic formats. However, Spring Dependency Injection can be utilized to load custom data loaders, filters, modifiers and output generators.

BTE in DSpace

The functionality of batch importing items in DSpace using the BTE has been incorporated in the "import" script already used in DSpace for years.

In the import script, there is a new option (option "-b") to import using the BTE and an option -i to declare the type of the input format. All the other options are the same apart from option "-s" that in this case points to a file (and not a directory as it used to) that is the file of the input data. However, in the case of batch BTE import, the option "-s" is not obligatory since you can configure the input from the Spring XML configuration file discussed later on. Keep in mind, that if option "-s" is defined, import will take that option into consideration instead of the one defined in the Spring XML configuration.

Thus, to import metadata from the various input formats use the following commands:

Input	Command
BibTeX (*.bib)	<code>[dspace]/bin/dspace import -b -m mapFile -e example@email.com -c 123456789/1 -s path-to-my-bibtex-file -i bibtex</code>
CSV (*.csv)	<code>[dspace]/bin/dspace import -b -m mapFile -e example@email.com -c 123456789/1 -s path-to-my-csv-file -i csv</code>
TSV (*.tsv)	<code>[dspace]/bin/dspace import -b -m mapFile -e example@email.com -c 123456789/1 -s path-to-my-tsv-file -i tsv</code>
RIS (*.ris)	<code>[dspace]/bin/dspace import -b -m mapFile -e example@email.com -c 123456789/1 -s path-to-my-ris-file -i ris</code>
EndNote	<code>[dspace]/bin/dspace import -b -m mapFile -e example@email.com -c 123456789/1 -s path-to-my-endnote-file -i endnote</code>
OAI-PMH	<code>[dspace]/bin/dspace import -b -m mapFile -e example@email.com -c 123456789/1 -s path-to-my-oai-file -i oai</code>
arXiv	<code>[dspace]/bin/dspace import -b -m mapFile -e example@email.com -c 123456789/1 -s path-to-my-arxiv-file -i arxivXML</code>
PubMed	<code>[dspace]/bin/dspace import -b -m mapFile -e example@email.com -c 123456789/1 -s path-to-my-pubmed-file -i pubmedXML</code>

Input	Command
CrossRef	<code>[dspace]/bin/dspace import -b -m mapFile -e example@email.com -c 123456789/1 -s path-to-my-crossref-file -i crossrefXML</code>
CiNii	<code>[dspace]/bin/dspace import -b -m mapFile -e example@email.com -c 123456789/1 -s path-to-my-crossref-file -i ciniifXML</code>

Keep in mind that the value of the "-e" option must be a valid email of a DSpace user and value of the "-c" option must be the target collection handle. Attached, you can find a .zip file ([sample-files.zip](#)) that includes examples of the file formats that are mentioned above.

BTE Configuration

The basic idea behind BTE is that the system holds the metadata in an internal format using a specific key for each metadata field. DataLoaders load the record using the aforementioned keys, while the output generator needs to map these keys to DSpace metadata fields.

The BTE configuration file is located in path: `[dspace]/config/spring/api/bte.xml` and it's a Spring XML configuration file that consists of Java beans. (If these terms are unknown to you, please refer to Spring Dependency Injection web site for more information.)

Explanation of beans:

```
<bean id="org.dspace.app.itemimport.BTEBatchImportService" />
```

This is the top level bean that describes the service of the batch import from the various external metadata formats. It accepts three properties:

- a) **dataLoaders**: a list of all the possible data loaders that are supported. Keep in mind that for each data loader we specify a key that can be used as the value of option "-i" in the import script that we mentioned earlier. Here is the point where you would add a new custom DataLoader in case the default ones doesn't match your needs.
- b) **outputMap**: a Map between the internal keys that BTE service uses to hold metadata and the DSpace metadata fields. (See later on, how data loaders specify the keys that BTE uses to hold the metadata)
- c) **transformationEngine**: the BTE transformation engine that actually consists of the processing steps that will be applied to metadata during their import to DSpace

```
<bean id="batchImportTransformationEngine" />
```

This bean is instantiated when the batch import takes place. It deploys a new BTE transformation engine that will do the transformation from one format to the other. It needs one input argument, the workflow (the

processing step mentioned before) that will run when transformation takes place. Normally, you don't need to modify this bean.

```
<bean id="batchImportLinearWorkflow" />
```

This bean describes the processing steps. Currently, there are no processing steps meaning that all records loaded by the data loader will pass to the output generator, unfiltered and unmodified. (See next section "Case studies" for info about how to add a filter or a modifier)

```
<bean id="bibTeXDataLoader" />
<bean id="csvDataLoader" />
<bean id="tsvDataLoader" />
<bean id="risDataLoader" />
<bean id="endnoteDataLoader" />
<bean id="pubmedFileDataLoader" />
<bean id="arXivFileDataLoader" />
<bean id="crossRefFileDataLoader" />
<bean id="oaipmhDataLoader" />
```

These data loaders are of two types: "file" data loaders and "online" data loaders. The first 8 of them belong to file data loaders while the last one (OAI data loader) is an online one.

The file data loaders have the following properties:

- a) **filename**: it is a String that specifies the filepath to the file that the loader will read data from. If you specify this property, you do not need to give the option "-s" to the import script in the command prompt. If you, however, specify this property and you also provide a "-s" option in the command line, the option "-s" will be taken into consideration by the data loader.
- b) **fieldMap**: it is a map that specifies the mapping between the keys that hold the metadata in the input file and the ones that we want to have internal in the BTE. This mapping is very important because the internal keys need to be declared in the "outputMap" of the "DataLoaderService" bean. Be aware that each data loader has each own input file keys. For example, RIS loader uses the keys "T1, AU, SO ..." while the TSV or CSV use the index number of the column that the value resides.

Some loaders have more properties:

CSV and TSV (which is actually a CSV loader if you look carefully the class value of the bean) loaders have some more properties:

- a) **skipLines**: A number that specifies the first line of the file that loader will start reading data. For example, if you have a csv file that the first row contains the column names, and the second row is empty, the the value of this property must be 2 so as the loader starts reading from row 2 (starting from 0 row). The default value for this property is 0.

- b) **separator**: A value to specify the separator between the values in the same row in order to make the columns . For example, in a TSV data loader this value is "\u0009" which is the "Tab" character. The default value is "," and that is why the CSV data loader doesn't need to specify this property.
- c) **quoteChar**: This property specifies the quote character used in the CSV file. The default value is the double quote character (").

The OAIPMHDataLoader has the following properties:

- a) **fieldMap**: Same as above, the mapping between the input keys holding the metadata and the ones that we want to have internal in BTE.
- b) **serverAddress**: The base address of the OAI provider (server). Base address can be specified also in the "-s" option of the command prompt. If is specified in both places, the one specified from the command line is preferred.
- c) **prefix**: The metadata prefix to be used in OAI requests.

Since DSpace administrators may have incorporated their own metadata schema within DSpace (apart from the default Dublin Core schema), they may need to configure BTE to match their custom schemas.

So, in case you need to process more metadata fields than those that are specified by default, you need to change the data loader configuration and the output map.

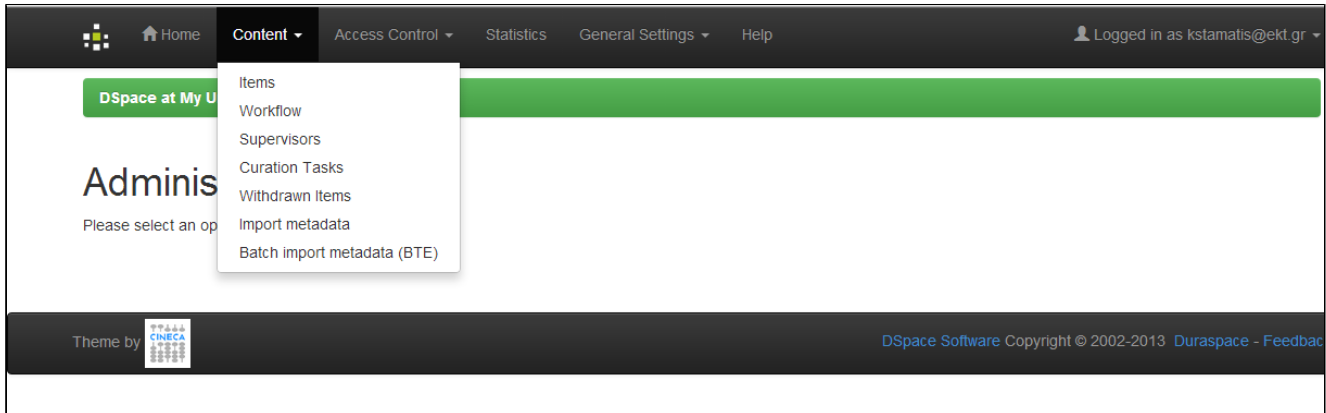


I can see more beans in the configuration file that are not explained above. Why is this?

The configuration file hosts options for two services. BatchImport service and [SubmissionLookup service](#). Thus, some beans that are not used for the latter, are not mentioned in this documentation. However, since both services are based on the BTE, some beans are used by both services.

UI for administrators

Batch import of files can be done via the administrative UI. While logged in as administrator, visit "Administer" link and then, under the "Content" drop down menu, choose "Batch import metadata (BTE)"



In the screen that follows, select the file to upload, select the data type of the file to be uploaded (bibtext, csv, etc .) and finally, select the collections the data need to be inserted to.

Keep in mind, that the type drop down menu includes all the supported data loaders declared in the configuration XML file that are of type "file". Thus, OAI data loader is not included in this list and in case you need to create your own data loader you are advised to extend the "FileDataLoader" abstract class rather than implement the "DataLoader" interface, as mentioned in previous paragraph.

The whole procedure can take long time to complete, in case of large input files, so the whole procedure runs in the background in a separate thread. When the thread is completed (either successfully or erroneously), the user is informed via email for the status of the import.

Case Studies

1) I have my data in a format different from the ones that are supported by this functionality. What can I do?

Either you try to easily transform your data to one of the supported formats or you need to create a new data loader. To do this, create a new Java class that implements the following Java interface from BTE:

```
gr.ekt.bte.core.DataLoader
```

You will need to implement the following method:

```
public RecordSet getRecords() throws MalformedSourceException
```

in which you have to create records - most probably you will need to create your own Record class (by implementing the `gr.ekt.bte.core.Record` interface) and fill a `RecordSet`. Feel free to add whatever code you like in this method, even to read data from multiple sources. All you need is just to return a `RecordSet` of Records.

You may also extend the abstract class

```
gr.ekt.bte.core.dataloader.FileDataLoader
```


if you want to create a "file" data loader in which you need to pass a filepath to the file that the loader will read the data from. Normally, a simple data loader is enough for the system to work, but file data loaders are also utilized in the administration UI discussed later in this documentation.

After that, you will need to declare the new DataLoader in the Spring XML configuration file (in the bean with id=" org.dspace.app.itemimport.BTEBatchImportService ") using your own unique key. Use this key as a value for option "-i" in the batch import in order to specify that the specific data loader must run.

2) I need to filter some of the input records or modify some value from records before outputting them

In this case you will need to create your own filters and modifiers.

To create a new filter, you need to extend the following BTE abstract class:

```
gr.ekt.bte.core.AbstractFilter
```

You will need to implement the following method:

```
public abstract boolean isIncluded ( Record record )
```

Return false if the specified record needs to be filtered, otherwise return true.

To create a new modifier, you need to extend the following BTE abstract class:

```
gr.ekt.bte.core.AbstractModifier
```

You will need to implement the following method:

```
public abstract Record modify ( Record record )
```

within you can make any changes you like in the record. You can use the Record methods to get the values for a specific key and load new ones (For the later, you need to make the Record mutable)

After you create your own filters or modifiers you need to add them in the Spring XML configuration file as in the following example:

```
<bean id="customfilter" class="org.mypackage.MyFilter" />
<bean id="batchImportLinearWorkflow" class="gr.ekt.bte.core.LinearWorkflow">
  <property name="process">
    <list>
```

```

        <ref bean="customfilter" />
    </list>
</property>
</bean>

```

You can add as many filters and modifiers you like to *batchImportLinearWorkflow*, they will run the one after the other in the specified order.

4.3.7 Importing Community and Collection Hierarchy

- 1 [Community and Collection Structure Importer](#)
 - 1.1 [Usage](#)
 - 1.2 [XML Import Format](#)
 - 1.3 [Limitations](#)

Community and Collection Structure Importer

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

Usage

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

XML Import Format

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

```

<import_structure>
  <community>
    <name>Community Name</name>
    <description>Descriptive text</description>
    <intro>Introductory text</intro>
    <copyright>Special copyright notice</copyright>
    <sidebar>Sidebar text</sidebar>
  </community>
  <community>
    <name>Sub Community Name</name>

```

```

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

```

The resulting output document will be as follows:

```

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

```

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

```

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

```

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

Limitations

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

4.3.8 SWORDv1 Server

SWORD (Simple Web-service Offering Repository Deposit) is a protocol that allows the remote deposit of items into repositories. DSpace implements the SWORD protocol via the 'sword' web application. The version of SWORD v1 currently supported by DSpace is 1.3. The specification and further information can be found at <http://swordapp.org>.

SWORD is based on the Atom Publish Protocol and allows service documents to be requested which describe the structure of the repository, and packages to be deposited.

- 1 [Enabling SWORD Server](#)
- 2 [Configuring SWORD Server](#)

Enabling SWORD Server

To enable DSpace's SWORD server, just make sure the `[dspace]/webapps/sword/` web application is available from your Servlet Container (usually Tomcat).

Configuring SWORD Server

Configuration File:	<code>[dspace]/config/modules/sword-server.cfg</code>
Property:	<code>mets-ingester.package-ingester</code>
Example Value:	<code>mets-ingester.package-ingester = METS</code>
Informational Note:	<p>The property key tell the SWORD METS implementation which package ingester to use to install deposited content. This should refer to one of the classes configured for:</p> <pre style="border: 1px dashed gray; padding: 5px; width: fit-content; margin: 10px auto;">plugin.named.org.dspace.content.packager.PackageIngester</pre>

Configuration File:	<code>[dspace]/config/modules/sword-server.cfg</code>
	The value of <code>sword.mets-ingester.package-ingester</code> tells the system which named plugin for this interface should be used to ingest SWORD METS packages.
Properties:	<code>mets.default.ingest.crosswalk.EPDCX</code> <code>mets.default.ingest.crosswalk.*</code> (NOTE: These configs are in the <code>dspace.cfg</code> file as they are used by many interfaces)
Example Value:	<code>mets.submission.crosswalk.EPDCX = EPDCX</code>
Informational Note:	Define the metadata types which can be accepted/handled by SWORD during ingest of a package. Currently, EPDCX (EPrints DC XML) is the recommended default metadata format, but others are supported.
Property:	<code>crosswalk.submission.EPDCX.stylesheet</code> (NOTE: This configuration is in the <code>dspace.cfg</code> file)
Example Value:	<code>crosswalk.submission.EPDCX.stylesheet = crosswalks/sword-swap-ingest.xsl</code>
Informational Note:	Define the stylesheet which will be used by the self-named XSLTIngestionCrosswalk class when asked to load the SWORD configuration (as specified above). This will use the specified stylesheet to crosswalk the incoming SWAP metadata to the DIM format for ingestion.
Property:	<code>deposit.url</code>
Example Value:	<code>deposit.url = http://www.myu.ac.uk/sword/deposit</code>
Informational Note:	The base URL of the SWORD deposit. This is the URL from which DSpace will construct the deposit location URLs for collections. The default is <code>\${dspace.baseUrl}/sword/deposit</code> (where <code>dspace.baseUrl</code> is defined in your <code>dspace.cfg</code> file). In the event that you are not deploying DSpace as the ROOT application in the servlet container, this will generate incorrect URLs, and you should override the functionality by specifying in full as shown in the example value.
Property:	<code>servicedocument.url</code>
Example Value:	<code>servicedocument.url = http://www.myu.ac.uk/sword/servicedocument</code>

File:	
Informational Note:	The base URL of the SWORD service document. This is the URL from which DSpace will construct the service document location URLs for the site, and for individual collections. The default is <code>\${dspace.baseUrl}/sword/servicedocument</code> (where <code>dspace.baseUrl</code> is defined in your <code>dspace.cfg</code> file). In the event that you are not deploying DSpace as the ROOT application in the servlet container, this will generate incorrect URLs, and you should override the functionality by specifying in full as shown in the example value.
Property:	<code>media-link.url</code>
Example Value:	<code>media-link.url = http://www.myu.ac.uk/sword/media-link</code>
Informational Note:	The base URL of the SWORD media links. This is the URL which DSpace will use to construct the media link URLs for items which are deposited via sword. The default is <code>\${dspace.baseUrl}/sword/media-link</code> (where <code>dspace.baseUrl</code> is defined in your <code>dspace.cfg</code> file). In the event that you are not deploying DSpace as the ROOT application in the servlet container, this will generate incorrect URLs, and you should override the functionality by specifying in full as shown in the example value.
Property:	<code>generator.url</code>
Example Value:	<code>generator.url = http://www.dspace.org/ns/sword/1.3.1</code>
Informational Note:	The URL which identifies the SWORD software which provides the sword interface. This is the URL which DSpace will use to fill out the <code>atom:generator</code> element of its atom documents. The default is: <code>{{http://www.dspace.org/ns/sword/1.3.1}}</code> . If you have modified your SWORD software, you should change this URI to identify your own version. If you are using the standard 'dspace-sword' module you will not, in general, need to change this setting.
Property:	<code>updated.field</code>
Example Value:	<code>updated.field = dc.date.updated</code>
Informational Note:	The metadata field in which to store the updated date for items deposited via SWORD.

Configuration File:	<code>[dspace]/config/modules/sword-server.cfg</code>
Property:	<code>slug.field</code>
Example Value:	<code>slug.field = dc.identifier.slug</code>
Informational Note:	The metadata field in which to store the value of the slug header if it is supplied.
Properties:	<pre>accept-packaging.METSDSpaceSIP.identifier accept-packaging.METSDSpaceSIP.q</pre>
Example Value:	<pre>accept-packaging.METSDSpaceSIP.identifier = http://purl.org/net/sword-types/ METSDSpaceSIP accept-packaging.METSDSpaceSIP.q = 1.0</pre>
Informational Note:	The accept packaging properties, along with their associated quality values where appropriate. This is a Global Setting; these will be used on all DSpace collections
Property:	<code>accepts</code>
Example Value:	<code>accepts = application/zip, foo/bar</code>
Informational Note:	A comma separated list of MIME types that SWORD will accept.
Properties:	<pre>accept-packaging.[handle].METSDSpaceSIP.identifier accept-packaging.[handle].METSDSpaceSIP.q</pre>
Example Value:	<pre>accept-packaging.[handle].METSDSpaceSIP.identifier = http://purl.org/net/ sword-types/METSDSpaceSIP accept-packaging.[handle].METSDSpaceSIP.q = 1.0</pre>
Informational Note:	Collection Specific settings: these will be used on the collections with the given handles.

Configuration File:	<code>[dspace]/config/modules/sword-server.cfg</code>
Property:	<code>expose-items</code>
Example Value:	<code>expose-items = false</code>
Informational Note:	Should the server offer up items in collections as sword deposit targets. This will be effected by placing a URI in the collection description which will list all the allowed items for the depositing user in that collection on request. NOTE: this will require an implementation of deposit onto items, which will not be forthcoming for a short while.
Property:	<code>expose-communities</code>
Example Value:	<code>expose-communities = false</code>
Informational Note:	Should the server offer as the default the list of all Communities to a Service Document request. If false, the server will offer the list of all collections, which is the default and recommended behavior at this stage. NOTE: a service document for Communities will not offer any viable deposit targets, and the client will need to request the list of Collections in the target before deposit can continue.
Property:	<code>max-upload-size</code>
Example Value:	<code>max-upload-size = 0</code>
Informational Note:	The maximum upload size of a package through the sword interface, in bytes. This will be the combined size of all the files, the metadata and any manifest data. It is NOT the same as the maximum size set for an individual file upload through the user interface. If not set, or set to 0, the sword service will default to no limit.
Property:	<code>keep-original-package</code>
Example Value:	<code>keep-original-package = true</code>
Informational Note:	Whether or not DSpace should store a copy of the original sword deposit package. NOTE: this will cause the deposit process to run slightly slower, and will accelerate the rate at which the repository consumes disk space. BUT, it will also mean that the deposited packages are recoverable in their original form. It is strongly recommended, therefore, to leave this option turned on. When set to "true", this requires that the configuration option <code>upload.temp.dir</code> (in <code>dspace.cfg</code>) is set to a valid location.
Property:	<code>bundle.name</code>

Configuration File:	<code>[dspace]/config/modules/sword-server.cfg</code>
Example Value:	<code>bundle.name = SWORD</code>
Informational Note:	The bundle name that SWORD should store incoming packages under if <code>sword.keep-original-package</code> is set to true. The default is "SWORD" if not value is set
Properties:	<code>keep-package-on-fail</code> <code>failed-package.dir</code>
Example Value:	<pre> keep-package-on-fail=true failed-package.dir=\${dspace.dir}/upload </pre>
Informational Note:	In the event of package ingest failure, provide an option to store the package on the file system. The default is false.
Property:	<code>identify-version</code>
Example Value:	<code>identify-version = true</code>
Informational Note:	Should the server identify the sword version in a deposit response. It is recommended to leave this unchanged.
Property:	<code>on-behalf-of.enable</code>
Example Value:	<code>on-behalf-of.enable = true</code>
Informational Note:	Should mediated deposit via sword be supported. If enabled, this will allow users to deposit content packages on behalf of other users.
Property:	<code>restore-mode.enable</code>
Example Value:	<code>restore-mode.enable = true</code>
Informational Note:	Should the sword server enable restore-mode when ingesting new packages. If this is enabled the item will be treated as a previously deleted item from the repository. If the item had previously been assigned a handle then that same handle will be restored to activity. If that item had not been previously assign a handle, then a new handle will be assigned.
Property:	<code>plugin.named.org.dspace.sword.SWORDingester</code>

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

4.3.9 SWORDv2 Server

SWORD (Simple Web-service Offering Repository Deposit) is a protocol that allows the remote deposit of items into repositories. DSpace implements the SWORD protocol via the 'sword' web application. The specification and further information can be found at <http://swordapp.org/>.

SWORD is based on the Atom Publish Protocol and allows service documents to be requested which describe the structure of the repository, and packages to be deposited.

- 1 [Enabling SWORD v2 Server](#)
- 2 [Configuring SWORD v2 Server](#)

Enabling SWORD v2 Server

To enable DSpace's SWORD v2 server, just make sure the `[dspace]/webapps/swordv2/` web application is available from your Servlet Container (usually Tomcat).

Configuring SWORD v2 Server

Configuration File:	<code>[dspace]/config/modules/swordv2-server.cfg</code>
Property:	<code>url</code>

Configuration File:	<code>[dspace]/config/modules/swordv2-server.cfg</code>
Example Value:	<code>url = http://www.myu.ac.uk/swordv2</code>
Informational Note:	The base url of the SWORD 2.0 system. This defaults to <code>\${dspace.baseUrl}/swordv2</code> (within <code>dspace.cfg</code> file).
Property:	<code>collection.url</code>
Example Value:	<code>collection.url = http://www.myu.ac.uk/swordv2/collection</code>
Informational Note:	The base URL of the SWORD collection. This is the URL from which DSpace will construct the defaults to <code>\${dspace.baseUrl}/swordv2/collection</code> (where <code>dspace.baseUrl</code> is defined in your <code>dspace.cfg</code> file).
Property:	<code>servicedocument.url</code>
Example Value:	<code>servicedocument.url = http://www.myu.ac.uk/swordv2/servicedocument</code>
Informational Note:	The service document URL of the SWORD collection. The base URL of the SWORD service document location urls for the site, and for individual collections. DSpace will construct the service document location urls for the site, and for individual collections. Defaults to <code>dspace.baseUrl/swordv2/servicedocument</code> (where <code>dspace.baseUrl</code> is defined in your <code>dspace.cfg</code> file).
Property:	<code>accept-packaging.collection</code>
Example Value:	<code>accept-packaging.collection.METSDSpaceSIP = http://purl.org/net/sword/package/METSDSpaceSIP</code> <code>accept-packaging.collection.SimpleZip = http://purl.org/net/sword/package/SimpleZip</code> <code>accept-packaging.collection.Binary = http://purl.org/net/sword/package/Binary</code>
Informational Note:	The accept packaging properties, along with their associated quality values where appropriate.
Property:	<code>accept-packaging.item</code>
Example Value:	<code>accept-packaging.item.METSDSpaceSIP = http://purl.org/net/sword/package/METSDSpaceSIP</code> <code>accept-packaging.item.SimpleZip = http://purl.org/net/sword/package/SimpleZip</code>

Configuration File:	<code>[dspace]/config/modules/swordv2-server.cfg</code>
	<code>accept-packaging.item.Binary = http://purl.org/net/sword/package/Binary</code>
Informational Note:	The accept packaging properties for items. It is possible to configure this for specific collections setting, for example <code>accept-packaging.collection.[handle].METSDSpaceSIP = http://purl.org/net/sword/package/Binary</code> METSDSpaceSIP
Property:	<code>accepts</code>
Example Value:	<code>accepts = application/zip, image/jpeg</code>
Informational Note:	A comma-separated list of MIME types that SWORD will accept. To accept all mimetypes, the value should be <code>*</code>
Property:	<code>expose-communities</code>
Example Value:	<code>expose-communities = false</code>
Informational Note:	Whether or not the server should expose a list of all the communities to a service document record collection, it is recommended to leave this set to false.
Property:	<code>max-upload-size</code>
Example Value:	<code>max-upload-size = 0</code>
Informational Note:	The maximum upload size of a package through the SWORD interface (measured in bytes). This includes the metadata, and manifest file in a package - this is different to the maximum size of a single bitstream. If this is set to 0, no maximum file size will be enforced.
Property:	<code>keep-original-package</code>
Example Value:	<code>keep-original-package = true</code>

Configuration File:	<code>[dspace]/config/modules/swordv2-server.cfg</code>
Informational Note:	<p>Should DSpace store a copy of the original SWORD deposit package?</p> <p>This will cause the deposit process to be slightly slower and for more disk to be used, however recommended to leave this option enabled.</p>
Property:	<code>bundle.name</code>
Example Value:	<code>bundle.name = SWORD</code>
Informational Note:	The bundle name that SWORD should store incoming packages within if <code>keep-original-pa</code>
Property:	<code>bundle.deleted</code>
Example Value:	<code>bundle.deleted = DELETED</code>
Informational Note:	The bundle name that SWORD should use to store deleted bitstreams if <code>versions.keep</code> is set to true. If individual files are updated or removed via SWORD. If the entire Media Resource (files in the C backed up in its entirety in a bundle of its own
Property:	<code>keep-package-on-fail</code>
Example Value:	<code>keep-package-on-fail = false</code>
Informational Note:	In the event of package ingest failure, provide an option to store the package on the file system set using the <code>failed-package-dir</code> setting.
Property:	<code>failed-package-dir</code>
Example Value:	<code>failed-package-dir = /dspace/upload</code>
Informational Note:	If <code>keep-package-on-fail</code> is set to true, this is the location where the package would be stored

Configuration File:	<code>[dspace]/config/modules/swordv2-server.cfg</code>
Property:	<code>on-behalf-of.enable</code>
Example Value:	<code>on-behalf-of.enable = true</code>
Informational Note:	Should DSpace accept mediated deposits? See the SWORD specification for a detailed explanation
Property:	<code>on-behalf-of.update.mediators</code>
Example Value:	<code>on-behalf-of.update.mediators = admin@myspace.edu, mediator@myspace.edu</code>
Informational Note:	Which user accounts are allowed to do updates on items which already exist in DSpace, on-behalf-of.update.mediators. If this is left blank, or omitted, then all accounts can mediate updates to items, which could be a security risk. It is recommended to check that the authenticated user is a "legitimate" mediator
Property:	<code>verbose-description.receipt.enable</code>
Example Value:	<code>verbose-description.receipt.enable = false</code>
Informational Note:	Should the deposit receipt include a verbose description of the deposit? For use by developers and production systems
Property:	<code>verbose-description.error.enable</code>
Example Value:	<code>verbose-description.error.enable = true</code>
Informational Note:	should the error document include a verbose description of the error? For use by developers, a value of "true" is recommended for production systems
Property:	<code>error.alternate.url</code>

Configuration File:	<code>[dspace]/config/modules/swordv2-server.cfg</code>
Example Value:	<code>error.alternate.url = http://myspace.edu/xmlui/contact</code>
Informational Note:	The error document can contain an alternate url, which the client can use to follow up any issue. Contact-Us page on the XMLUI
Property:	<code>error.alternate.content-type</code>
Example Value:	<code>error.alternate.content-type = text/html</code>
Informational Note:	The <code>error.alternate.url</code> may have an associated content type, such as <code>text/html</code> if it is provided to the client what content type it can expect if it follows that url.
Property:	<code>generator.url</code>
Example Value:	<code>generator.url = http://www.dspace.org/ns/sword/2.0/</code>
Informational Note:	The URL which identifies DSpace as the software that is providing the SWORD interface.
Property:	<code>generator.version</code>
Example Value:	<code>generator.version = 2.0</code>
Informational Note:	The version of the SWORD interface.
Property:	<code>auth-type</code>
Example Value:	<code>auth-type = Basic</code>
Informational Note:	Which form of authentication to use. Normally this is set to <code>Basic</code> in order to use HTTP Basic.

Configuration File:	<code>[dspace]/config/modules/swordv2-server.cfg</code>
Property:	<code>upload.tempdir</code>
Example Value:	<code>upload.tempd = /dspace/upload</code>
Informational Note:	The location where uploaded files and packages are stored while being processed.
Property:	<code>updated.field</code>
Example Value:	<code>updated.field = dc.date.updated</code>
Informational Note:	The metadata field in which to store the updated date for items deposited via SWORD.
Property:	<code>slug.field</code>
Example Value:	<code>slug.field = dc.identifier.slug</code>
Informational Note:	The metadata field in which to store the value of the slug header if it is supplied.
Property:	<code>author.field</code>
Example Value:	<code>author.field = dc.contributor.author</code>
Informational Note:	The metadata field in which to store the value of the atom entry author if it supplied.
Property:	<code>title.field</code>
Example Value:	<code>dc.title</code>

Configuration File:	<code>[dspace]/config/modules/swordv2-server.cfg</code>
Informational Note:	The metadata field in which to store the value of the atom entry title if it supplied.
Property:	<code>disseminate-packaging</code>
Example Value:	<pre>disseminate-packaging.METSDSpaceSIP = http://purl.org/net/sword/package/METSDSpaceSIP disseminate-packaging.SimpleZip = http://purl.org/net/sword/package/SimpleZip</pre>
Informational Note:	Supported packaging formats for the dissemination of packages.
Property:	<code>statement.bundles</code>
Example Value:	<code>statement.bundles = ORIGINAL, SWORD, LICENSE</code>
Informational Note:	Which bundles should the Statement include in its list of aggregated resources? The Statement which are in the bundle identified by the <code>#{bundle.name}</code> property, provided that bundle is all Deposits to be listed in the Statement then you should add the SWORD bundle to this list)
Property:	<code>plugin.single.org.dspace.sword2.WorkflowManager</code>
Example Value:	<pre>plugin.single.org.dspace.sword2.WorkflowManager = org.dspace.sword2.WorkflowManagerDe</pre>
Informational Note:	Which workflow manager to use.
Property:	<code>workflowmanagerdefault.always-update-metadata</code>
Example Value:	<code>workflowmanagerdefault.always-update-metadata = true</code>
Informational Note:	Should the WorkflowManagerDefault plugin allow updates to the item's metadata to take place workspace (e.g. in the workflow, archive, or withdrawn) ?
Property:	<code>workflowmanagerdefault.file-replace.enable</code>

Example Value	<code>workflowmanagerdefault.file-replace.enable = false</code>
Informational Note	<p>Should the server allow PUT to individual files?</p> <p>If this is enabled, then DSpace may be used with the DepositMO SWORD extensions, BUT the support Bitstream replace, so this is equivalent to a DELETE and then a POST, which violates file DOES NOT have the same identifier as the file it was replacing. As such it is STRONGLY R off unless working explicitly with DepositMO enabled client environments</p>
Property:	<code>mets-ingester.package-ingester</code>
Example Value:	<code>mets-ingester.package-ingester = METS</code>
Informational Note:	Which package ingester to use for METS packages.
Property:	<code>restore-mode.enable</code>
Example Value:	<code>restore-mode.enable = false</code>
Informational Note:	Should the SWORD server enable restore-mode when ingesting new packages. If this is enabled deleted item from the repository. If the item has previously been assigned a handle then that se
Property:	<code>simpledc.*</code>
Example Value:	<code>simpledc.abstract = dc.description.abstractsimpledc.date = dc.date simpledc.rights = d</code>
Informational Note:	Configuration of metadata field mapping used by the SimpleDCEntryIngester, SimpleDCEntryD
Property:	<code>atom.*</code>
Example Value	<code>atom.author = dc.contributor.author</code>

Configuration File:	<code>[dspace]/config/modules/swordv2-server.cfg</code>
Informational Note:	Configuration of metadata field mapping used by the SimpleDCEntryIngester, SimpleDCEntryD
Property:	<code>metadata.replaceable</code>
Example Value	<code>metadata.replaceable = dc.description.abstract, dc.rights, dc.title.al</code>
Informational Note	Used by SimpleDCEntryIngester: Which metadata fields can be replaced during a PUT to the It listed here are the ones which will be removed when a new PUT comes through (irrespective o replace them)
Property:	<code>multipart.entry-first</code>
Example Value:	<code>multipart.entry-first = false</code>
Informational Note:	The order of precedence for importing multipart content. If this is set to <code>true</code> then metadata in atom entry, otherwise the metadata in the atom entry will override that from the package.
Property:	<code>workflow.notify</code>
Example Value:	<code>workflow.notify = true</code>
Informational Note:	If the workflow gets started (the collection being deposited into has a workflow configured), sho
Property:	<code>versions.keep</code>
Example Value:	<code>versions.keep = true</code>
Informational Note:	When content is replaced, should the old version be kept? This creates a copy of the ORIGINAL where YYYY-MM-DD is the date the copy was created, and X is an integer from 0 upwards.
Property:	<code>state.*</code>

Configuration File:	<code>[dspace]/config/modules/swordv2-server.cfg</code>
Example Value:	<pre>state.workspace.uri = http://localhost:8080/xmlui/state/inprogress state.workspace.description = The item is in the user workspace state.workflow.uri = http://localhost:8080/xmlui/state/inreview state.workflow.description = The item is undergoing review prior to acceptance in the</pre>
Informational Note:	Pairs of states (URI and description) than items can be in. Typical states are <code>workspace</code> , <code>work</code> .
Property:	<code>workspace.url-template</code>
Example Value	<code>workspace.url-template = http://myspace.edu/xmlui/submit?workspaceID=</code>
Informational Note	URL template for links to items in the workspace (items in the archive will use the handle). The the workspace id of the item. The example above shows how to construct this URL for XMLUI.

Other configuration options exist that define the mapping between mime types, ingesters, and disseminators. A typical configuration looks like this:

```
plugin.named.org.dspace.sword2.SwordContentIngestor = \
  org.dspace.sword2.SimpleZipContentIngestor = http://purl.org/net/sword/package/SimpleZip, \
  org.dspace.sword2.SwordMETSIIngestor = http://purl.org/net/sword/package/METSdSpaceSIP, \
  org.dspace.sword2.BinaryContentIngestor = http://purl.org/net/sword/package/Binary
plugin.single.org.dspace.sword2.SwordEntryIngestor = \
  org.dspace.sword2.SimpleDCEntryIngestor
plugin.single.org.dspace.sword2.SwordEntryDisseminator = \
  org.dspace.sword2.SimpleDCEntryDisseminator
# note that we replace ";" with "_" as ";" is not permitted in the PluginManager names
plugin.named.org.dspace.sword2.SwordContentDisseminator = \
  org.dspace.sword2.SimpleZipContentDisseminator = http://purl.org/net/sword/package/SimpleZip, \
  org.dspace.sword2.FeedContentDisseminator = application/atom+xml, \
  org.dspace.sword2.FeedContentDisseminator = application/atom+xml_type_feed
# note that we replace ";" with "_" as ";" is not permitted in the PluginManager names
plugin.named.org.dspace.sword2.SwordStatementDisseminator = \
  org.dspace.sword2.AtomStatementDisseminator = atom, \
  org.dspace.sword2.OreStatementDisseminator = rdf, \
  org.dspace.sword2.AtomStatementDisseminator = application/atom+xml_type_feed, \
  org.dspace.sword2.OreStatementDisseminator = application/rdf+xml
```

4.3.10 Ingesting HTML Archives

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

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

Dealing with these issues is the topic of much active research. Currently, DSpace bites off a small, tractable chunk of this problem. DSpace can store and provide on-line browsing capability for *self-contained, non-dynamic* HTML documents. In practical terms, this means:

- No dynamic content (CGI scripts and so forth)
- All links to preserved content must be *relative links*, that do not refer to 'parents' above the 'root' of the HTML document/site:
 - *diagram.gif* is OK
 - *image/foo.gif* is OK
 - *../index.html* is only OK in a file that is at least a directory deep in the HTML document/site hierarchy
 - */stylesheet.css* is not OK (the link will break)
 - <http://somedomain.com/content.html> is not OK (the link will continue to link to the external site which may change or disappear)
- Any 'absolute links' (e.g. <http://somedomain.com/content.html>) are stored 'as is', and will continue to link to the external content (as opposed to relative links, which will link to the copy of the content stored in DSpace.) Thus, over time, the content referred to by the absolute link may change or disappear.

4.4 Items and Metadata

- [Authority Control of Metadata Values](#)
- [Batch Metadata Editing](#)
- [DOI Digital Object Identifier](#)
- [Item Level Versioning](#)
- [Mapping Items](#)
- [Metadata Recommendations](#)
- [Moving Items](#)
- [ORCID Integration](#)
- [PDF Citation Cover Page](#)
- [Updating Items via Simple Archive Format](#)

4.4.1 Authority Control of Metadata Values

- [1 WORK IN PROGRESS](#)
- [2 Introduction](#)
- [3 Simple choice management for DSpace submission forms](#)
 - [3.1 Example](#)
- [4 Hierarchical Taxonomies and Controlled Vocabularies](#)
 - [4.1 How to invoke a controlled vocabulary from input-forms.xml](#)
- [5 Authority Control: Enhancing DSpace metadata fields with Authority Keys](#)
 - [5.1 How it looks in the DSpace user interface](#)
 - [5.2 How it works](#)
 - [5.3 Original source:](#)

WORK IN PROGRESS

Introduction

With DSpace you can describe digital objects such as text files, audio, video or data to facility easy retrieval and high quality search results. These descriptions are organized into metadata fields that each have a specific designation, for example `dc.title` stores the title of an object, while `dc.subject` is reserved for subject keywords.

For many of these fields, including title and abstract, free text entry is the de facto choice as the values are likely to be unique. Other fields are likely to be associated with values that can occur across different items. Such fields include unique names, subject keywords, document types and other classifications. For those kinds of fields the overall quality of the repository metadata increases if values with the same meaning are normalized across all items. Additional benefits can be gained if unique identifiers are associated as well in addition to canonical text values associated with a particular metadata field.

This page covers features included in the DSpace submission forms that allow repository managers to enforce the usage of normalized terms for those fields where this is required in their institutional use cases. DSpace offers simple and straightforward features, such as definitions of simple text values for dropdowns, as well as more elaborate integrations with external vocabularies such as the Library of Congress Naming Authority.

Simple choice management for DSpace submission forms

The DSpace Submission forms, defined in the `input-forms.xml` file, allows the inclusion of value pairs that can be organized in lists in order to populate dropdowns or other multiple choice elements. If you explore the default `input-forms.xml` file, you can see that a number of such value pair lists are already pre defined.

Example

```
<value-pairs value-pairs-name="common_identifiers" dc-term="identifier">
  <pair>
    <displayed-value>Gov't Doc #</displayed-value>
    <stored-value>govdoc</stored-value>
  </pair>
  <pair>
    <displayed-value>URI</displayed-value>
    <stored-value>uri</stored-value>
  </pair>
  <pair>
    <displayed-value>ISBN</displayed-value>
    <stored-value>isbn</stored-value>
  </pair>
</value-pairs>
```

It generates the following HTML, which results in the menu widget below.

```
<select name="identifier_qualifier_0">
  <option VALUE="govdoc">Gov't Doc #</option>
  <option VALUE="uri">URI</option>
  <option VALUE="isbn">ISBN</option>
</select>
```

A list of value pairs has following required attributes:

- **value-pairs-name** – Name by which an *input-type* refers to this list.
- **dc-term** – Dublin Core field for which this choice list is selecting a value.

Each *value-pairs* element contains a sequence of *pair* sub-elements, each of which in turn contains two elements:

- **displayed-value** – Name shown (on the web page) for the menu entry.

- **stored-value** – Value stored in the DC element when this entry is chosen. Unlike the HTML *select* tag, there is no way to indicate one of the entries should be the default, so the first entry is always the default choice.

Hierarchical Taxonomies and Controlled Vocabularies

The value pairs system works great for short and flat lists of choices. DSpace offers a second way of structuring and managing more complex, hierarchical controlled vocabularies. In contrast to the value pairs system, these controlled vocabularies are managed in separate XML files in the `[dspace]/config/controlled-vocabularies/` directory instead of being entered straight into `input-forms.xml`

The taxonomies are described in XML according to this 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>
```

As you can see, each node element has an `id` and `label` attribute. It can contain the `isComposedBy` element, which in its turn, consists of a list of other nodes.

You are free to use any application you want to create your controlled vocabularies. A simple text editor should be enough for small projects. Bigger projects will require more complex tools. You may use Protegé to create your taxonomies, save them as OWL and then use a XML Stylesheet (XSLT) to transform your documents to the appropriate format. Future enhancements to this add-on should make it compatible with standard schemas such as OWL or RDF.

How to invoke a controlled vocabulary from `input-forms.xml`

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>
```



```
<repeatable>true</repeatable>
<label>Subject Keywords</label>
<input-type>onebox</input-type>
<hint>Enter appropriate subject keywords or phrases below.</hint>
<required></required>
<vocabulary>srsc</vocabulary>
</field>
```

The vocabulary element has an optional boolean attribute closed that can be used to force input only with the Javascript of controlled-vocabulary add-on. The default behaviour (i.e. without this attribute) is as set closed="false". This allow the user also to enter values as free text, not selecting them from the controlled vocabulary.

The following vocabularies are currently available by default:

- **nsi** - *nsi.xml* - The Norwegian Science Index
- **srsc** - *srsc.xml* - Swedish Research Subject Categories

Authority Control: Enhancing DSpace metadata fields with Authority Keys


The aforementioned features only deal with text representations of controlled values. DSpace also offers support for adding authority keys and confidence values to a specific text value entered in a metadata field. The following terminology applies in the description of this area of DSpace functionality:

- **Authority** An *authority* is an external source of fixed values for a given domain, each unique value identified by a *key*. For example, [the OCLC LC Name Authority Service](#), ORCID or VIAF.
- **Authority Record** The information associated with one of the values in an authority; may include alternate spellings and equivalent forms of the value, etc.
- **Authority Key** An opaque, hopefully persistent, identifier corresponding to exactly one record in the authority.

The fact that this functionality deals with **external** sources of authority makes it inherently different from the functionality for controlled vocabularies. Another difference is that the authority control is asserted *everywhere* metadata values are changed, including unattended/batch submission, LNI and SWORD package submission, and the administrative UI.

How it looks in the DSpace user interface

The difference between an authority controlled metadata field and a non-authority controlled metadata field can be seen in the Edit interface for an accepted item.

Remove	Name	Value
<input type="checkbox"/>	dc.contributor.author	<input type="text" value="Smith, Dona"/> <input type="text" value="no 97030482"/>  <input type="button" value="Lookup"/>

Authority controlled author field edit

This example shows a value for an author name that has been linked with an authority key. The green thumb represents the associated confidence value "Accepted": This authority value has been confirmed as accurate by an interactive user or authoritative policy.

How it works

TODO

Original source:

[Authority Control of Metadata Values](#) original development proposal for DSpace 1.6

4.4.2 Batch Metadata Editing

- 1 [Batch Metadata Editing Tool](#)
 - 1.1 [Export Function](#)
 - 1.1.1 [Web Interface Export](#)
 - 1.1.2 [Command Line Export](#)
 - 1.2 [Import Function](#)
 - 1.2.1 [Web Interface Import](#)
 - 1.2.2 [Command Line Import](#)
 - 1.3 [CSV Format](#)
 - 1.3.1 [File Structure](#)
 - 1.4 [Editing the CSV](#)
 - 1.4.1 [Editing Collection Membership](#)
 - 1.4.2 [Adding Metadata-Only Items](#)
 - 1.4.3 [Deleting Metadata](#)
 - 1.4.4 [Performing 'actions' on items](#)
 - 1.4.5 [Migrating Data or Exchanging data.](#)

Batch Metadata Editing Tool

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

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

For information about configuration options for the Batch Metadata Editing tool, see [Batch Metadata Editing Configuration](#)

Export Function

Web Interface Export

Batch metadata exports (to CSV) can be performed from the Administrative menu:

- Login as an Administrative user
- Browse to the Community or Collection you wish to export to CSV
 - NOTE: in the JSPUI, it is also possible to export search results to CSV. Just perform a search, and click on the "Export Metadata" button above the search results.
- Click "Export Metadata" link to export to a downloadable CSV
 - In XMLUI, "Export Metadata" can be found in the "Context" menu on a Community/Collection homepage
 - In JSPUI, "Export Metadata" can be found in the "Admin Tools" menu on a Community/Collection homepage

Please see below documentation for more information on the [CSV format](#) and actions that can be performed by [editing the CSV](#).

Command Line Export

The following table summarizes the basics.

Command used :	<code>[dspace]/bin/dspace metadata-export</code>
Java class:	<code>org.dspace.app.bulkedit.MetadataExport</code>
Arguments short and (long) forms):	Description
<code>-f</code> or <code>--file</code>	Required. The filename of the resulting CSV.
<code>-i</code> or <code>--id</code>	

	The Item, Collection, or Community handle or Database ID to export. If not specified, all items will be exported.
<code>-a</code> or <code>--all</code>	Include all the metadata fields that are not normally changed (e.g. provenance) or those fields you configured in the <code>[dspace]/config/modules/bulkedit.cfg</code> to be ignored on export.
<code>-h</code> or <code>--help</code>	Display the help page.

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

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

Example:

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

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

Please see below documentation for more information on the [CSV format](#) and actions that can be performed by [editing the CSV](#).

Import Function



Importing large CSV files

It is not recommended to import CSV files of more than 1,000 lines (i.e. 1,000 items). When importing files larger than this, it may be difficult for an Administrator to accurately verify the changes that the import tool states it will make. In addition, depending on the memory available to the DSpace site, large files may cause 'Out Of Memory' errors part way through the import process.

Web Interface Import

Batch metadata imports (from CSV) can be performed from the Administrative menu:

- First, complete all [editing of the CSV](#) and save your changes
- Login as an Administrative User
- Click "Import Metadata" and select the CSV file
 - In XMLUI, "Import Metadata" can be found under the "Administrative" menu on any page

- In JSPUI, "Import Metadata" can be found under the "Administer" menu (under your user account dropdown). On the Administration Tools page, select "Import Metadata" from the "Content" dropdown
- After uploading the CSV, you will be presented with a summary of all changes that will be performed in the system. You can review these changes and choose whether to apply them or cancel.

Command Line Import

The following table summarizes the basics.

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

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

To run the batch importer, at the command line:

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

Example

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

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

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

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

CSV Format

The CSV (comma separated values) files that this tool can import and export abide by the [RFC4180](#) CSV format. This means that new lines, and embedded commas can be included by wrapping elements in double quotes. Double quotes can be included by using two double quotes. The code does all this for you, and any good csv editor such as Excel or OpenOffice will comply with this convention.

All CSV files are also in UTF-8 encoding in order to support all languages.

File Structure

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

A typical heading row looks like:

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

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


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

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

```
Horses | Dogs | Cats
```

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

Editing the CSV

 **If you are editing with Microsoft Excel, be sure to open the CSV in Unicode/UTF-8 encoding**

By default, Microsoft Excel may not correctly open the CSV in Unicode/UTF-8 encoding. This means that special characters may be improperly displayed and also can be "corrupted" during re-import of the CSV.

You need to tell Excel this CSV is Unicode, by importing it as follows. (*Please note these instructions are valid for MS Office 2007 and 2013. Other Office versions may vary*)

- First, open Excel (with an empty sheet/workbook open)
- Select "Data" tab
- Click "From Text" button (in the "External Data" section)
- Select your CSV file
- Wizard Step 1
 - Choose "Delimited" option
 - Start import at row: 1
 - In the "File origin" selectbox, select "65001 : Unicode (UTF-8)"
 - NOTE: these encoding options are sorted alphabetically, so "Unicode (UTF-8)" appears near the bottom of the list.
 - Click Next
- Wizard Step 2
 - Select "Comma" as the only delimiter
 - Click Next
- Wizard Step 3
 - Select "Text" as the "Column data format"
 - Click Finish
- Choose whether to open CSV in the existing sheet or a new one

Tips to Simplify the Editing Process

When editing a CSV, here's a couple of basic tips to keep in mind:

1. The "id" column MUST remain intact. This column also must always have a value in it.
2. To simplify the CSV, you can simply remove any columns you do NOT wish to edit (except for "id" column, see #1). Don't worry, removing the entire column won't delete metadata (see #3)
3. When importing a CSV file, the importer will *overlay* the metadata onto what is already in the repository to determine the differences. It *only* acts on the contents of the CSV file, rather than on the complete item metadata. This means that the CSV file that is exported can be manipulated quite substantially before being re-imported. Rows (items) or Columns (metadata elements) can be removed and will be ignored.
 - a. For example, if you only want to edit "dc.subject", you can remove ALL columns EXCEPT for "id" and "dc.subject" so that you can just manipulate the "dc.subject" field. On import, DSpace will see that you've only included the "dc.subject" field in your CSV and therefore will only update the "dc.subject" metadata field for any items listed in that CSV.

4. Because removing an entire column does NOT delete metadata value(s), if you actually wish to delete a metadata value you should leave the column intact, and simply clear out the appropriate row's value (in that column).

Editing Collection Membership

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

Adding Metadata-Only Items

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

Deleting Metadata

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

Performing 'actions' on items

It is possible to perform certain 'actions' on items. This is achieved by adding an 'action' column to the CSV file (after the id, and collection columns). There are three possible actions:

1. **'expunge'** This permanently deletes an item. Use with care! This action must be enabled by setting 'allowexpunge = true' in `modules/bulkedit.cfg`
2. **'withdraw'** This withdraws an item from the archive, but does not delete it.
3. **'reinstate'** This reinstates an item that has previously been withdrawn.

If an action makes no change (for example, asking to withdraw an item that is already withdrawn) then, just like metadata that has not changed, this will be ignored.

Migrating Data or Exchanging data.

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

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

Batch Metadata Editing Configuration

The [Batch Metadata Editing Tool](#) allows the administrator to extract from the DSpace database a set of records for editing via a CSV file. It provides an easier way of editing large collections.

A full list of all available Batch Metadata Editing Configurations:

Configuration File:	<code>[dspace]/config/modules/bulkedit.cfg</code>
Property:	<code>valueseparator</code>
Example Value:	<code>valueseparator = </code>
Informational note	The delimiter used to separate values within a single field. For example, this will place the double pipe between multiple authors appearing in one record (Smith, William Johannsen, Susan). This applies to any metadata field that appears more than once in a record. The user can change this to another character.
Property:	<code>fieldseparator</code>
Example Value:	<code>fieldseparator = ,</code>
Informational note	The delimiter used to separate fields (defaults to a comma for CSV). Again, the user could change it something like '\$'. If you wish to use a tab, semicolon, or hash (#) sign as the delimiter, set the value to be <code>tab</code> , <code>semicolon</code> or <code>hash</code> . <pre>fieldseparator = tab</pre>
Property:	<code>gui-item-limit</code>
Example Value:	<code>gui-item-limit = 20</code>
Informational note	When using the WEBUI, this sets the limit of the number of items allowed to be edited in one processing. There is no limit when using the CLI.

Configuration File:	<code>[dspace]/config/modules/bulkedit.cfg</code>
Property:	<code>ignore-on-export</code>
Example Value:	<pre>ignore-on-export = dc.date.accessioned, \ dc.date.available, \ dc.date.updated, dc.description.provenance</pre>
Informational note	Metadata elements to exclude when exporting via the user interfaces, or when using the command line version and not using the -a (all) option.

4.4.3 DOI Digital Object Identifier

- [Persistent Identifier](#)
- [DOI Registration Agencies](#)
- [Configure DSpace to use the DataCite API](#)
 - [dspace.cfg](#)
 - [Metadata conversion](#)
 - [Identifier Service](#)
 - [Command Line Interface](#)
 - ['cron' job for asynchronous reservation/registration](#)
 - [Limitations of DOI support](#)
- [Configure DSpace to use EZID service for registration of DOIs](#)
- [Adding support for other Registration Agencies](#)

Persistent Identifier

It is good practice to use Persistent Identifiers to address items in a digital repository. There are many different systems for Persistent Identifiers: [Handle](#) , [DOI](#) , [urn:nbn](#), [purl](#) and many more. It is far out of the scope of this document to discuss the differences of all these systems. For several reasons the Handle System is deeply integrated in DSpace, and DSpace makes intensive use of it. With DSpace 3.0 the [Identifier Service](#) was introduced that makes it possible to also use external identifier services within DSpace.

DOIs are Persistent Identifiers like Handles are, but as many big publishing companies use DOIs they are quite well-known to scientists. Some journals ask for DOIs to link supplemental material whenever an article is submitted. Beginning with DSpace 4.0 it is possible to use DOIs in parallel to the Handle System within DSpace. By "using DOIs" we mean automatic generation, reservation and registration of DOIs for every item that enters the repository. These newly registered DOIs will not be used as a means to build URLs to DSpace items. Items will still rely on handle assignment for the item urls.

DOI Registration Agencies

To register a DOI one has to enter into a contract with a DOI registration agency which is a member of the International DOI Foundation. Several such agencies exist. Different DOI registration agencies have different policies. Some of them offer DOI registration especially or only for academic institutions, others only for publishing companies. Most of the registration agencies charge fees for registering DOIs, and all of them have different rules describing for what kind of item a DOI can be registered. To make it quite clear: to register DOIs with DSpace you have to enter into a contract with a DOI registration agency.

[DataCite](#) is an international initiative to promote science and research, and a member of the International DOI Foundation. The members of DataCite act as registration agencies for DOIs. Some DataCite members provide their own APIs to reserve and register DOIs; others let their clients use the DataCite API directly. Starting with version 4.0 DSpace supports the administration of DOIs by using the DataCite API directly or by using the API from EZID (which is a service of the University of California Digital Library). This means you can administer DOIs with DSpace if your registration agency allows you to use the DataCite API directly or if your registration agency is EZID.

Configure DSpace to use the DataCite API

If you use a DOI registration agency that lets you use the DataCite API directly, you can follow the instructions below to configure DSpace. In case EZID is your registration agency the configuration of DSpace is documented here: [Configure DSpace to use EZID service for registration of DOIs](#).

To use DOIs within DSpace you have to configure several parts of DSpace:

- enter your DOI prefix and the credentials to use the API from DataCite in `dspace.cfg`,
- configure the script which generates some metadata,
- activate the DOI mechanism within DSpace,
- configure a cron job which transmits the information about new and changed DOIs to the registration agency.

`dspace.cfg`

After you enter into a contract with a DOI registration agency, they'll provide you with user credentials and a DOI prefix. You have to enter these in the `dspace.cfg`. Here is a list of DOI configuration options in `dspace.cfg`:

Configuration File:	<code>[dspace]/config/dspace.cfg</code>
Property:	<code>identifier.doi.user</code>
Example Value:	<code>identifier.doi.user = user123</code>

Configuration File:	[dspace]/config/dspace.cfg
Informational Note:	Username to login into the API of the DOI registration agency. You'll get it from your DOI registration agency.
Property:	<code>identifier.doi.password</code>
Example Value:	<code>identifier.doi.password = top-secret</code>
Informational Note:	Password to login into the API of the DOI registration agency. You'll get it from your DOI registration agency.
Property:	<code>identifier.doi.prefix</code>
Example Value:	<code>identifier.doi.prefix = 10.5072</code>
Informational Note:	The prefix you got from the DOI registration agency. All your DOIs start with this prefix, followed by a slash and a suffix generated from DSpace. The prefix can be compared with a namespace within the DOI system.
Property:	<code>identifier.doi.namespaceseparator</code>
Example Value:	<code>identifier.doi.namespaceseparator = dspace-</code>
Informational Note:	This property is optional. If you want to use the same DOI prefix in several DSpace installations or with other tools that generate and register DOIs it is necessary to use a namespace separator. All the DOIs that DSpace generates will start with the DOI prefix, followed by a slash, the namespace separator and some number generated by DSpace. For example, if your prefix is 10.5072 and you want all DOIs generated by DSpace to look like 10.5072/dspace-1023 you have to set this as in the example value above.



Please don't use the test prefix 10.5072 with DSpace. The test prefix 10.5072 differs from other prefixes: It answers GET requests for all DOIs even for DOIs that are unregistered. DSpace checks

that it mint only unused DOIs and will create an Error: "Register DOI ... failed: DOI_ALREADY_EXISTS". Your registration agency can provide you an individual test prefix, that you can use for tests.

Metadata conversion

To reserve or register a DOI, DataCite requires that metadata be supplied which describe the object that the DOI addresses. The file [dspace]/config/crosswalks/DIM2DataCite.xsl controls the conversion of metadata from the DSpace internal format into the DataCite format. You have to add the name of your institution to this file:

`[dspace]/config/crosswalks/DIM2DataCite.xsl`

```

<!--
  Document      : DIM2DataCite.xsl
  Created on   : January 23, 2013, 1:26 PM
  Author      : pbecker, ffuerste
  Description  : Converts metadata from DSpace Intermediat Format (DIM) into
                metadata following the DataCite Schema for the Publication and
                Citation of Research Data, Version 2.2
-->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:dspace="http://www.dspace.org/xmlns/dspace/dim"
                xmlns="http://datacite.org/schema/kernel-2.2"
                version="1.0">
  <!-- CONFIGURATION -->
  <!-- The content of the following variable will be used as element publisher. -->
  <xsl:variable name="publisher">My University</xsl:variable>
  <!-- The content of the following variable will be used as element contributor with
  contributorType datamanager. -->
  <xsl:variable name="datamanager"><xsl:value-of select="$publisher" /></xsl:variable>
  <!-- The content of the following variable will be used as element contributor with
  contributorType hostingInstitution. -->
  <xsl:variable name="hostinginstitution"><xsl:value-of select="$publisher" /></xsl:variable>
  <!-- Please take a look into the DataCite schema documentation if you want to know how to use
  these elements.
                http://schema.datacite.org -->
  <!-- DO NOT CHANGE ANYTHING BELOW THIS LINE EXCEPT YOU REALLY KNOW WHAT YOU ARE DOING! -->
  ...

```

Just change the value in the variable named "publisher".

If you want to know more about the DataCite Schema, have a look at the [documentation](#). If you change this file in a way that is not compatible with the DataCite schema, you won't be able to reserve and register DOIs anymore. Do not change anything if you're not sure what you're doing.

Identifier Service

The Identifier Service manages the generation, reservation and registration of identifiers within DSpace. You can configure it using the config file located in [dspace]/config/spring/api/identifier-service.xml. In the file you should already find the code to configure DSpace to register DOIs. Just read the comments and remove the comment signs around the two appropriate beans.

After removing the comment signs the file should look something like this (I removed the comments to make the listing shorter):

[dspace]/config/spring/api/identifier-service.xml

```

<!--
  Copyright (c) 2002-2010, DuraSpace. All rights reserved
  Licensed under the DuraSpace License.
  A copy of the DuraSpace License has been included in this
  distribution and is available at: http://www.dspace.org/license
-->
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">
  <bean id="org.dspace.identifier.IdentifierService"
    class="org.dspace.identifier.IdentifierServiceImpl"
    autowire="byType"
    scope="singleton"/>
  <bean id="org.dspace.identifier.DOIIdentifierProvider"
    class="org.dspace.identifier.DOIIdentifierProvider"
    scope="singleton">
    <property name="configurationService"
      ref="org.dspace.services.ConfigurationService" />
    <property name="DOIConnector"
      ref="org.dspace.identifier.doi.DOIConnector" />
  </bean>
  <bean id="org.dspace.identifier.doi.DOIConnector"
    class="org.dspace.identifier.doi.DataCiteConnector"
    scope="singleton">
    <property name='DATACITE_SCHEME' value='https' />
    <property name='DATACITE_HOST' value='mds.datacite.org' />
    <property name='DATACITE_DOI_PATH' value='/doi/' />
    <property name='DATACITE_METADATA_PATH' value='/metadata/' />
    <property name='disseminationCrosswalkName' value="DataCite" />
  </bean>
</beans>

```

If you use other IdentifierProviders beside the DOIIdentifierProvider there will be more beans in this file.

Please pay attention to configure the property DATACITE_HOST. Per default it is set to the DataCite test server. To reserve real DOIs you will probably have to change it to mds.datacite.org. Ask your registration agency if you're not sure about the correct address. Unfortunately the test and the production server have different paths

to the API. For the test server you have to set the `DATAcite_Doi_Path` to `"/mds/doi/"` and the `DATAcite_Metadata_Path` to `"/mds/doi/"`, for the production server you have to remove the leading `/mds` from both properties.

DSpace should send updates to DataCite whenever the metadata of an item changes. To do so you have to change the `dspace.cfg` again. You should remove the comments in front of the two following properties or add them to the `dspace.cfg`:

```
[\dspace]/config/dspace.cfg
```

```
event.consumer.doi.class = org.dspace.identifier.doi.DOIConsumer
event.consumer.doi.filters = Item+Modify_Metadata
```

Then you should add 'doi' to the property `event.dispatcher.default.consumers`. After adding it, this property may look like this:

```
[\dspace]/config/dspace.cfg
```

```
event.dispatcher.default.consumers = versioning, discovery, eperson, harvester, doi
```

Command Line Interface

To make DSpace resistant to outages of DataCite we decided to separate the DOI support into two parts. When a DOI should be generated, reserved or minted, DSpace does this in its own database. To perform registration and/or reservation against the DOI registration agency a job has to be started using the command line. Obviously this should be done by a cron job periodically. In this section we describe the command line interface, in case you ever want to use it manually. In the next section you'll see the cron job that transfers all DOIs designated for reservation and/or registration.

The command line interface in general is documented here: [Command Line Operations](#). The command used for DOIs is 'doi-organiser'. You can use the following options:

Option (short)	Option (long)	Parameter	Description
-d	--delete-all		Transmit information to the DOI registration agency about all DOIs that were deleted.
	--delete-doi	DOI	Transmit information to the DOI registration agency that the specified DOI was deleted. The DOI must already be marked for deletion; you cannot use this command to delete a DOI for an existing item.
-h	--help		Print online help.
-l	--list		List all DOIs whose changes were not committed to the registration agency yet.

Option (short)	Option (long)	Parameter	Description
-q	--quiet		The doi-organiser sends error reports to the mail address configured in the property alert.recipient in dspace.cfg. If you use this option no output should be given to stdout. If you do not use this option the doi-organiser writes information about successful and unsuccessful operations to stdout and stderr. You can find information in dspace.log of course.
-r	--register-all		Transmit information about all DOIs that should be registered.
	--register-doi	DOI ItemID handle	If a DOI is marked for registration, you can trigger the registration at the DOI registration agency by this command. Specify either the DOI, the ID of the item, or its handle.
-s	--reserve-all		Transmit to the DOI registration agency information about all DOIs that should be reserved.
	--reserve-doi	DOI ItemID handle	If a DOI is marked for registration, you can trigger the registration at the DOI registration agency by this command. Specify either the DOI, the ID of the item, or its handle.
-u	--update-all		If a DOI is reserved for an item, the metadata of the item will be sent to DataCite. This command transmits new metadata for items whose metadata were changed since the DOI was reserved.
	--update-doi	DOI ItemID handle	If a DOI needs an update of the metadata of the item it belongs to, you can trigger this update with this command. Specify either the DOI, the ID of the item, or its handle.

Currently you cannot generate new DOIs with this tool. You can only send information about changes in your local DSpace database to the registration agency.

'cron' job for asynchronous reservation/registration

When a DOI should be reserved, registered, deleted or its metadata updated, DSpace just writes this information into its local database. A command line interface is supplied to send the necessary information to the registration agency. This behavior makes it easier to react to outages or errors while using the API. This information should be sent regularly, so it is a good idea to set up a cron job instead of doing it manually.

There are four commands that should be run regularly:

- Update the metadata of all items that have changed since their DOI was reserved.
- Reserve all DOIs marked for reservation
- Register all DOIs marked for registration
- Delete all DOIs marked for deletion

In DSpace, a DOI can have the state "registered", "reserved", "to be reserved", "to be registered", "needs update", "to be deleted", or "deleted". After updating an item's metadata the state of its assigned DOI is set back to the last state it had before. So, e.g., if a DOI has the state "to be registered" and the metadata of its item changes, it will be set to the state "needs update". After the update is performed its state is set to "to be registered" again. Because of this behavior **the order of the commands above matters**: the update command must be executed before all of the other commands above.

The cron job should perform the following commands with the rights of the user your DSpace installation runs as :

```
[dspace]/bin/dspace doi-organiser -u -q
[dspace]/bin/dspace doi-organiser -s -q
[dspace]/bin/dspace doi-organiser -r -q
[dspace]/bin/dspace doi-organiser -d -q
```

The doi-organiser sends error messages as email and logs some additional information. The option -q tells DSpace to be quiet. If you don't use this option the doi-organiser will print messages to stdout about every DOI it successfully reserved, registered, updated or deleted. Using a cron job these messages would be sent as email.

In case of an error, consult the log messages. If there is an outage of the API of your registration agency, DSpace will not change the state of the DOIs so that it will do everything necessary when the cron job starts the next time and the API is reachable again.

The frequency the cron job runs depends on your needs and your hardware. The more often you run the cron job the faster your new DOIs will be available online. If you have a lot of submissions and want the DOIs to be available really quickly, you probably should run the cron job every fifteen minutes. If there are just one or two submissions per day, it should be enough to run the cron job twice a day.

To set up the cron job, you just need to run the following command as the *dspace* UNIX user:

```
crontab -e
```

The following line tells cron to run the necessary commands twice a day, at 1am and 1pm. Please notice that the line starting with the numbers is one line, even it it should be shown as multiple lines in your browser.

```
# Send information about new and changed DOIs to the DOI registration agency:
0 1,13 * * * [dspace]/bin/dspace doi-organiser -u -q ; [dspace]/bin/dspace doi-organiser -s -q ; [
dspace]/bin/dspace doi-organiser -r -q ; [dspace]/bin/dspace doi-organiser -d -q
```

Limitations of DOI support



Every DSpace installation expects to be the only application that generates DOIs which start with the prefix and the namespace separator you configured. DSpace does not check whether a DOI it generates is reserved or registered already.

That means if you want to use other applications or even more than one DSpace installation to register DOIs with the same prefix, you'll have to use a unique namespace separator for each of them. Also you should not generate DOIs manually with the same prefix and namespace separator you configured within DSpace. For example, if your prefix is 10.5072 you can configure one DSpace installation to generate DOIs starting with 10.5072/papers-, a second installation to generate DOIs starting with 10.5072/data- and another application to generate DOIs starting with 10.5072/results-.

DOIs will be used in addition to Handles. This implementation does not replace Handles with DOIs in DSpace. That means that DSpace will still generate Handles for every item, every collection and every community, and will use those Handles as part of the URL of items, collections and communities.

DSpace currently generates DOIs for items only. There is no support to generate DOIs for Communities and collections yet.

When using DSpaces support for the DataCite API probably not all information would be restored when using the AIP Backup and Restore (see [DS-1836-doi_seq](#) in `update-sequences.sql` missing More Details Needed). The DOIs included in metadata of Items will be restored, but DSpace won't update the metadata of those items at DataCite anymore. You can even get problems when minting new DOIs after you restored older once using AIP.

Configure DSpace to use EZID service for registration of DOIs

The EZID IdentifierProvider operates synchronously, so there is much less to configure. You will need to un-comment the `EZIDIdentifierProvider` bean in `config/spring/api/identifier-service.xml` to enable DOI registration through EZID.

In `config/dspace.cfg` you will find a small block of settings whose names begin with `identifier.doi.ezid`. You should uncomment these properties and give them appropriate values. Sample values for a test account are supplied.

name	meaning
<code>identifier.doi.ezid.shoulder</code>	The "shoulder" is the DOI prefix issued to you by the EZID service. DOIs minted by this instance of DSpace will be the concatenation of the "shoulder" and a locally unique token.
<code>identifier.doi.ezid.user</code>	The username and password by which you authenticate to EZID.
<code>identifier.doi.ezid.password</code>	

name	meaning
identifier.doi.ezid.publisher	You may specify a default value for the required <code>datacite.publisher</code> metadatum, for use when the Item has no publisher.

In `config/spring/api/identifier-service.xml` you will see some other configuration of the `EZIDIdentifierProvider` bean. You may not need to change any of it. But here you can alter the mapping between DSpace and EZID metadata, should you choose. The `crosswalk` property is a map from DSpace metadata fields to EZID fields, and can be extended or changed. The `key` of each `entry` is the name of an EZID metadata field; the `value` is the name of the corresponding DSpace field, from which the EZID metadata will be populated.

You can also supply transformations to be applied to field values using the `crosswalkTransform` property. Each `key` is the name of an EZID metadata field, and its `value` is the name of a Java class which will convert the value of the corresponding DSpace field to its EZID form. The only transformation currently provided is one which converts a date to the year of that date, named `org.dspace.identifier.ezid.DateToYear`. In the configuration as delivered, it is used to convert the date of issue to the year of publication. You may create new Java classes with which to supply other transformations, and map them to metadata fields here. If an EZID metadatum is not named in this map, the default mapping is applied: the string value of the DSpace field is copied verbatim.

Normally, you should not change the value of the `EZID_SCHEME` property of the `EZIDRequestFactory` bean. If you registered with the CDLib provider, you should not change the value of `EZID_HOST`; if you registered with another provider, such as Purdue, you should set the `EZID_HOST` appropriately (for example, <https://ezid.lib.purdue.edu>). If your provider provides EZID service at a particular path on its host, you may set that in `EZID_PATH`.

Adding support for other Registration Agencies

If you want DSpace to support other registration agencies, you just have to write a Java class that implements the interface `DOIConnector` (`[dspace-source]/dspace-api/src/main/java/org/dspace/identifier/doi/DOIConnector.java`). You might use the `DataCiteConnector` (`[dspace-source]/dspace-api/src/main/java/org/dspace/identifier/doi/DataCiteConnector.java`) as an example. After developing your own `DOIConnector`, you configure DSpace as if you were using the DataCite API directly. Just use your `DOIConnector` when configuring the `IdentifierService` instead of the `DataCiteConnector`.

4.4.4 Item Level Versioning

- 1 [What is Item Level Versioning?](#)
- 2 [Important warnings - read before enabling](#)
- 3 [Enabling Item Level Versioning](#)

- 3.1 [Steps for XML UI](#)
- 3.2 [Steps for JSP UI](#)
- 4 [Initial Requirements](#)
- 5 [User Interface](#)
 - 5.1 [General behaviour: Linear Versioning](#)
 - 5.2 [Creating a new version of an item](#)
 - 5.3 [View the history and older versions of an item](#)
- 6 [Architecture](#)
 - 6.1 [Versioning model](#)
 - 6.2 [Services to support Versioning and Alternative Identifiers](#)
 - 6.2.1 [Versioning Service](#)
 - 6.2.2 [Identifier Service](#)
- 7 [Configuration](#)
 - 7.1 [Versioning Service Override](#)
 - 7.2 [Identifier Service Override](#)
 - 7.3 [Version History Visibility](#)
- 8 [Identified Challenges & Known Issues in DSpace 4.0](#)
 - 8.1 [Only Administrators and Collection/Community Administrators can add new versions](#)
 - 8.2 [Conceptual compatibility with Embargo](#)
 - 8.3 [Conceptual compatibility with Item Level Statistics](#)
 - 8.4 [Exposing version history](#)
- 9 [Credits](#)

What is Item Level Versioning?

Versioning is a new functionality to build the history of an item. Users will have the opportunity to create new version of an existing item any time they will make a change.

Important warnings - read before enabling

AIP Backup & Restore functionality only works with the Latest Version of Items

If you are using the [AIP Backup and Restore](#) functionality to backup / restore / migrate DSpace Content, you must be aware that the "Item Level Versioning" feature is **not yet compatible** with AIP Backup & Restore. **Using them together may result in accidental data loss.** Currently the AIPs that DSpace generates only store the *latest version* of an Item. Therefore, past versions of Items will always be lost when you perform a restore / replace using AIP tools. See [DS-1382](#).

Versioning history exposes data that may be considered personal

If you enable versioning, the **name and email of the submitter are shown to all users** by default in Version history. The only way to circumvent this is to make Version history visible only to admins by setting `item.history.view.admin=false` in `[dspace]/config/modules/versioning.cfg`. See [DS-1349](#) for ongoing work on a better solution.

Enabling Item Level Versioning

By default, Item Level Versioning is disabled in DSpace 3, 4 and 5.

 Starting from DSpace 4.0, Item Level Versioning is also supported in JSPUI.

Steps for XML UI

If you wish to enable this feature, you just have to uncomment the "Versioning" aspect in your `[dspace]/config/xmlui.xconf` file (and restart your servlet container):

```
<!-- =====
      Item Level Versioning
      ===== -->
<!-- To enable Item Level Versioning features, uncomment this aspect. -->
<aspect name="Versioning Aspect" path="resource://aspects/Versioning/" />
```

Steps for JSP UI

If you wish to enable this feature, you just have to edit the `enabled` settings in your `[dspace]/config/modules/versioning.cfg` file (and restart your servlet container):

```
#-----#
#----- VERSIONING CONFIGURATIONS -----#
#-----#
# These configs are used by the versioning system #
#-----#
#Parameter 'enabled' is used only by JSPUI
enabled=false
```

Initial Requirements

The Item Level Versioning implementation in DSpace 3.0 builds on following requirements identified by the stakeholders who supported this contribution: [Initial Requirements Analysis](#)

1. What should be *Versionable*
 - a. Versioning happens at the level of an Individual Item

- b. Versioning should preserve the current state of *metadata*, *bitstreams* and *resource policies* attached to the item.
2. Access, Search and Discovery
 - a. Only the most recent version of an item is available via the search interface
 - b. Previous versions of Items should continue to be visible, citable and accessible
 - c. The Bitstreams for previous versions are retained. If something was once retrievable, it should always be retrievable.
3. Identifiers
 - a. Each version of an Item is represented by a separate "*versioned*" identifier
 - b. A base "*versionhistory*" Identifier points to the most recent version of the Item.
 - c. A revision identifier also exists that is unique to the specific version.
 - d. When a new version of an Item is deposited, a new revision identifier will be created.
4. Presentation
 - a. On the item page, there is a link to view previous/subsequent versions.
 - b. By examining the metadata or identifiers, it is possible to determine whether an item is the most recent version, the original version, or an intermediate version.
5. Access Control and Rights
 - a. Certain roles should be able to generate a new version of the item via submission.
 - b. To submitters, collection manager, administrators will be given to option to create new version of an item.
 - c. Rights to access a specific Item should transmute as well to previous versions
 - d. Rights to access a specific Bitstream should also transmute to previous versions.
6. Data Integrity
 - a. The relationships between versions should not be brittle and breakable by manipulating Item metadata records.
 - b. The relationships between versions should be preserved and predictable in various Metadata Exports (OAI, Packagers, ItemExport)
 - c. The relationships between versions should be maintained in SWORD, LNI and AIP packaging and be maintained in updates and restorations.

User Interface

General behaviour: Linear Versioning

From the user interface, DSpace offers **linear** versioning. As opposed to hierarchical versioning, linear version has following properties:

- A new version can only be created started from the latest available version
- When new version has been created and still needs to pass certain steps of the workflow, it is temporarily impossible to create another new version until the workflow steps are finished and the new version has replaced the previous one.

Creating a new version of an item

Administrators and collection/community administrators can create new versions of an item from the Item View page.

1. Click "Create a new version" from the Context Menu in the navigation bar.
2. Provide the reason for creating a new version that will later on be stored and displayed in the version summary.

Create new version of item: 123456789/127

Reason for creating new version:

/ /

3. Your new version is now creates as a new Item in your Workspace. It requires you to go through the submission and workflow steps like you would do for a normal, new submission to the collection. The rationale behind this is that if you are adding new files or metadata, you will also need to accept the license for them. In addition to this, the versioning functionality does not bypass any quality control embedded in the workflow steps.

Item submission

My Test Item
Doe, Jane

Date: 2012-11-21

Abstract:

This is the test abstract for my new versioned item. When I resume this, I go through the submission steps and the workflow before my new version becomes archived in the repository.

Files in this item

Files	Size	Format	View
There are no files associated with this item.			

The following license files are associated with this item:

- [Creative Commons](#)

After the submission steps and the execution of subsequent workflow steps, the new version becomes available in the repository.

View the history and older versions of an item

An overview of the version history, including links to older versions of an item, is available at the bottom of an Item View page. The repository administrator can decide whether the version history should be available to all users or restricted to administrators.

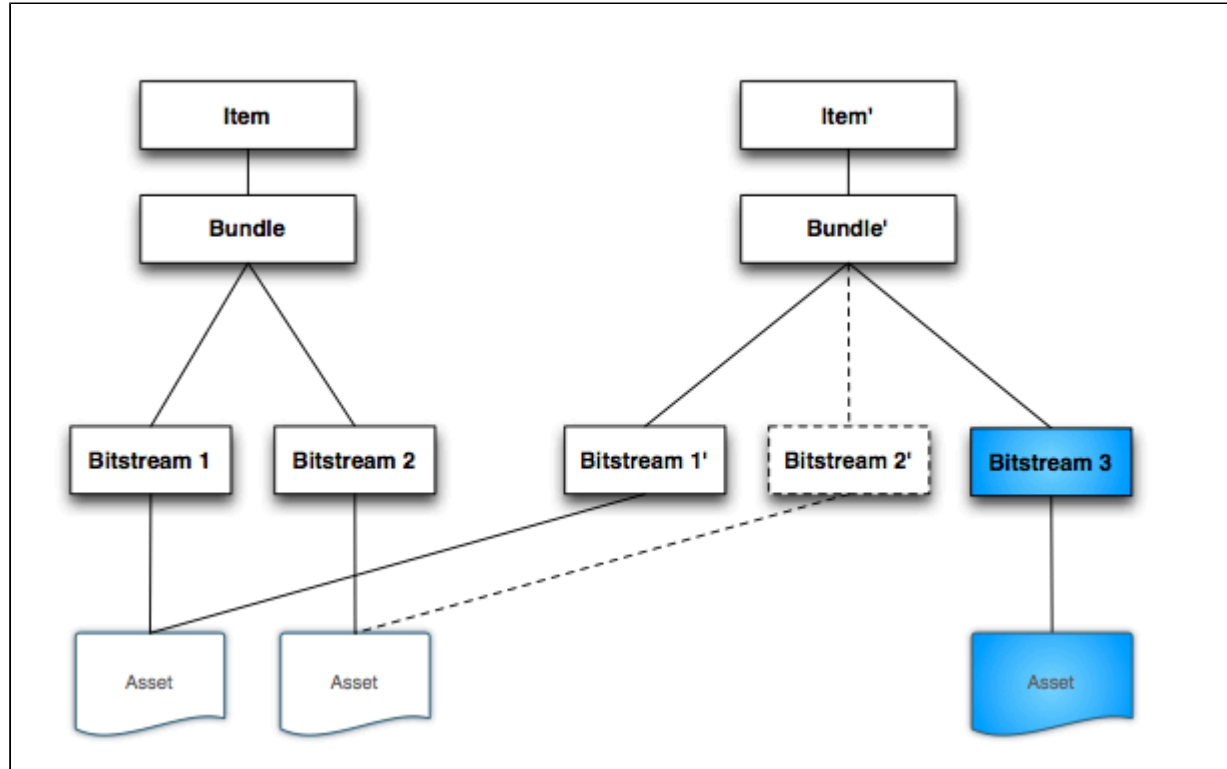
Version History				
Version	Item	Editor	Date	Summary
2	10673/138*	Demo Administrator	2012-10-23T12:11:46Z	Second version of this item - nothing actually changed
1	10673/138.1	Demo Administrator	2012-10-23T12:10:04Z	

*Selected version

Architecture

Versioning model

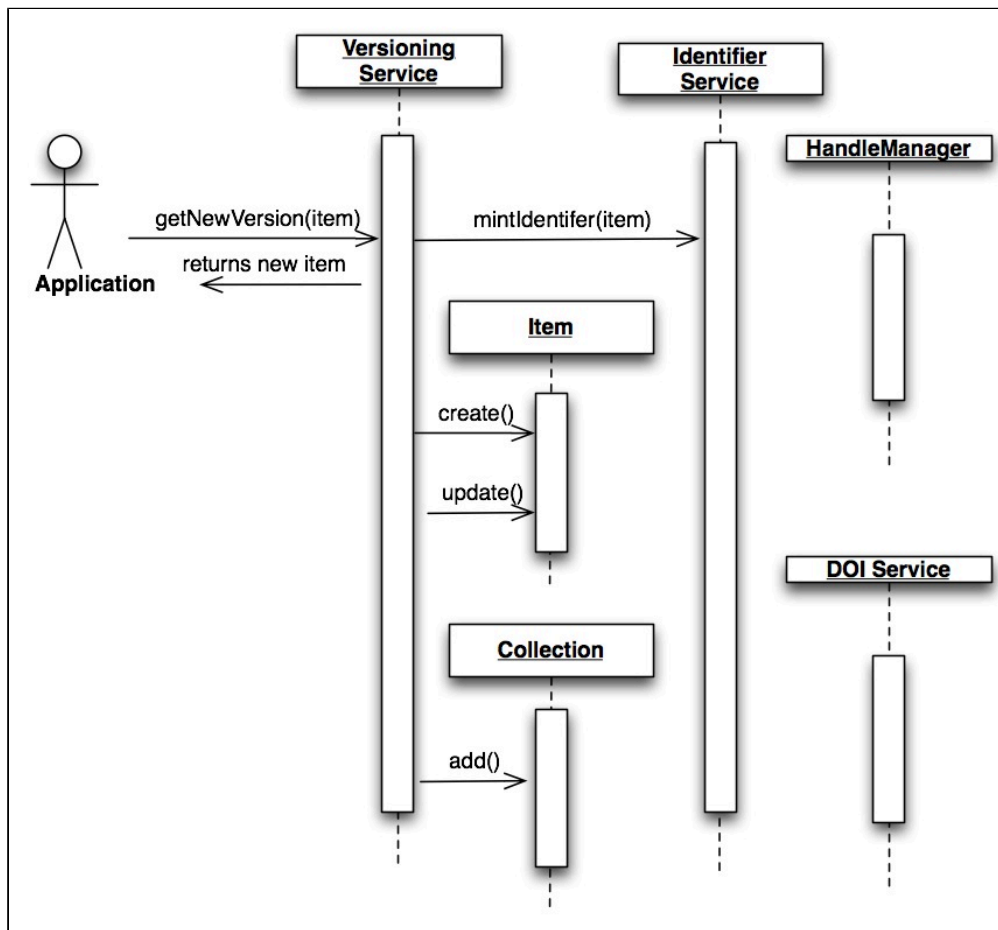
For every new Version a separate DSpace Item will be created that replicates the metadata, bundle and bitstream records. The bitstream records will point to the same file on the disk.



The *Cleanup* method has been modified to retain the file if another Bitstream record point to it (the dotted lines in the diagram represent a bitstream deleted in the new version), in other words the file will be deleted only if the Bitstream record processed is the only one to point to the file ($count(INTERNAL_ID)=1$).

Services to support Versioning and Alternative Identifiers

DSpace Item Versioning will be encapsulated as an Extensible Service that may be reimplemented by the local repository maintainers to produce alternate versioning behaviors and Identifier Schemes. Versioning Services layer on top of IdentifierServices dedicated to Encoding, Resolution, Minting and Registration of Identifiers for specific DSpace Items and Bitstreams. It is through this highly extensible layering of functionality where local developers can alter the versioning behavior and introduce their own local enhancements. The DSpace Service Manager, based on the Spring Framework, provides the key leverage for this flexibility.



Versioning Service

The *Versioning* Service will be responsible for the replication of one or more Items when a new version is requested. The new version will not yet be preserved in the Repository, it will be preserved when the databases transactional window is completed, thus when errors arise in the *versioning* process, the database will be properly kept in its original state and the application will alert that an exception has occurred that is in need of correction.

The *Versioning Service* will rely on a generic *IdentifierService* that is described below for minting and registering any identifiers that are required to track the revision history of the Items.

```
public interface VersioningService {
    Version createNewVersion(Context c, int itemId);
    Version createNewVersion(Context c, int itemId, String summary);
    void removeVersion(Context c, int versionID);
    void removeVersion(Context c, Item item);
    Version getVersion(Context c, int versionID);
    Version restoreVersion(Context c, int versionID);
    Version restoreVersion(Context c, int versionID, String summary);
    VersionHistory findVersionHistory(Context c, int itemId);
    Version updateVersion(Context c, int itemId, String summary);
    Version getVersion(Context c, Item item);
}
```

Identifier Service

The Identifier Service maintains an extensible set of *IdentifierProvider* services that are responsible for two important activities in Identifier management:

1. Resolution: *IdentifierService* act in a manner similar to the existing HandleManager in DSpace, allowing for resolution of DSpace Items from provided identifiers.
2. Minting: Minting is the act of reserving and returning an identifier that may be used with a specific DSpaceObject.
3. Registering: Registering is the act of recording the existence of a minted identifier with an external persistent resolver service. These services may reside on the local machine (HandleManager) or exist as external services (PURL or EZID DOI registration services)

```
public interface IdentifierService {
    /**
     *
     * @param context
     * @param dso
     * @param identifier
     * @return
     */
    String lookup(Context context, DSpaceObject dso, Class<? extends Identifier> identifier);
    /**
     *
     * This will resolve a DSpaceObject based on a provided Identifier. The Service will
     interrogate the providers in
     * no particular order and return the first successful result discovered. If no
     resolution is successful,
     * the method will return null if no object is found.
     *
     * TODO: Verify null is returned.
     *
     * @param context
     */
}
```

```

    * @param identifier
    * @return
    * @throws IdentifierNotFoundException
    * @throws IdentifierNotResolvableException
    */
    DSpaceObject resolve(Context context, String identifier) throws
IdentifierNotFoundException, IdentifierNotResolvableException;
/**
 *
 * Reserves any identifiers necessary based on the capabilities of all providers in the
service.
 *
 * @param context
 * @param dso
 * @throws org.dspace.authorize.AuthorizeException
 * @throws java.sql.SQLException
 * @throws IdentifierException
 */
void reserve(Context context, DSpaceObject dso) throws AuthorizeException, SQLException,
IdentifierException;
/**
 *
 * Used to Reserve a Specific Identifier (for example a Handle, hdl:1234.5/6) The
provider is responsible for
 * Detecting and Processing the appropriate identifier, all Providers are interrogated,
multiple providers
 * can process the same identifier.
 *
 * @param context
 * @param dso
 * @param identifier
 * @throws org.dspace.authorize.AuthorizeException
 * @throws java.sql.SQLException
 * @throws IdentifierException
 */
void reserve(Context context, DSpaceObject dso, String identifier) throws
AuthorizeException, SQLException, IdentifierException;
/**
 *
 * @param context
 * @param dso
 * @return
 * @throws org.dspace.authorize.AuthorizeException
 * @throws java.sql.SQLException
 * @throws IdentifierException
 */
void register(Context context, DSpaceObject dso) throws AuthorizeException, SQLException,
IdentifierException;
/**
 *
 * Used to Register a Specific Identifier (for example a Handle, hdl:1234.5/6) The
provider is responsible for

```

```

    * Detecting and Processing the appropriate identifier, all Providers are interrogated,
multiple providers
    * can process the same identifier.
    *
    * @param context
    * @param dso
    * @param identifier
    * @return
    * @throws org.dspace.authorize.AuthorizeException
    * @throws java.sql.SQLException
    * @throws IdentifierException
    */
    void register(Context context, DSpaceObject dso, String identifier) throws
AuthorizeException, SQLException, IdentifierException;
    /**
    * Delete (Unbind) all identifiers registered for a specific DSpace item. Identifiers are
"unbound" across
    * all providers in no particular order.
    *
    * @param context
    * @param dso
    * @throws org.dspace.authorize.AuthorizeException
    * @throws java.sql.SQLException
    * @throws IdentifierException
    */
    void delete(Context context, DSpaceObject dso) throws AuthorizeException, SQLException,
IdentifierException;
    /**
    * Used to Delete a Specific Identifier (for example a Handle, hdl:1234.5/6) The
provider is responsible for
    * Detecting and Processing the appropriate identifier, all Providers are interrogated,
multiple providers
    * can process the same identifier.
    *
    * @param context
    * @param dso
    * @param identifier
    * @throws org.dspace.authorize.AuthorizeException
    * @throws java.sql.SQLException
    * @throws IdentifierException
    */
    void delete(Context context, DSpaceObject dso, String identifier) throws
AuthorizeException, SQLException, IdentifierException;
  }

```

Configuration

Versioning Service Override

You can override the default behaviour of the Versioning Service using following XML configuration file, deployed under your dspace installation directory:

[\[dspace_installation_dir\]/config/spring/api/versioning-service.xml](#)

In this file, you can specify which metadata fields are automatically "reset" (i.e. cleared out) during the creation of a new item version. By default, all metadata values (and bitstreams) are copied over to the newly created version, with the exception of **dc.date.accessioned** and **dc.description.provenance**. You may specify additional metadata fields to reset by adding them to the "ignoredMetadataFields" property in the "versioning-service.xml" file:

```

<!-- Default Item Versioning Provider, defines behavior for replicating
     Item, Metadata, Budles and Bitstreams. Autowired at this time. -->
<bean class="org.dspace.versioning.DefaultItemVersionProvider">
  <property name="ignoredMetadataFields">
    <set>
      <value>dc.date.accessioned</value>
      <value>dc.description.provenance</value>
    </set>
  </property>
</bean>

```

Identifier Service Override

You can override the default behaviour of the Identifier Service using following XML configuration file, deployed under your dspace installation directory:

[\[dspace_installation_dir\]/config/spring/api/identifier-service.xml](#)

No changes to this file are required to enable Versioning. This file is currently only relevant if you aim to develop your own implementation of versioning.

Version History Visibility

Version History

Version	Item	Editor	Date	Summary
2	10673/138*	Demo Administrator	2012-10-23T12:11:46Z	Second version of this item - nothing actually changed
1	10673/138.1	Demo Administrator	2012-10-23T12:10:04Z	

*Selected version

By default, **all** users will be able to see the version history. To ensure that only administrators can see the Version History, enable `item.history.view.admin` in following configuration file:

[\[dspace_installation_dir\]/config/modules/versioning.cfg](#)

```
item.history.view.admin=false
```

Identified Challenges & Known Issues in DSpace 4.0

Item Level Versioning has a substantial conceptual impact on many DSpace features. Therefore it has been accepted into DSpace 3.0 as an optional feature and it is still an option feature in DSpace 4.0. Following challenges have been identified in the current implementation. As an early adopter of the Item Level Versioning feature, your input is greatly appreciated on any of these.

Only Administrators and Collection/Community Administrators can add new versions

There is currently no configuration option to allow submitters to create new versions of an item. This functionality is restricted to Administrators and Collection/Community Administrators. In a context where original submission of DSpace items is done by non-administrator users, it might also make sense to allow them to create new versions. Especially given the fact that new versions have to pass through the workflow anyway.

Conceptual compatibility with Embargo

Lifting an embargo currently does not interact with Item Level Versioning. Additional implementation would be required to ensure that lifting an embargo actually creates a new version of the item.

Conceptual compatibility with Item Level Statistics

Both on the level of pageviews and downloads, different versions of an item are treated as different items. As a result, an end user will have the impression that the stats are being "reset" once a new version is installed, because the previous downloads and pageviews are allocated to the previous version.

One possible solution would be to present an end user with aggregated statistics across all versions, and give administrators the possibility to view statistics per version.

Exposing version history

The version history is added on the bottom of a versioned item page. A repository administrator can either decide to show this to all users, or restrict it to admins only. If it is shown to admins only, an end user will have no way to find the way to an older version. On the other hand, if a repository administrator does decide to expose version history to all users, the name and email address of the editor are exposed as well. This might actually be useful if the editor account is a generic institutional email address, but may conflict with local privacy laws if any personal details are included in this account.

Therefore, discussion has illustrated that there is a usecase for an intermediate exposure of version history that hides the Editor column.

You can join the discussion or contribute a new code here:

[JIRA DS-1349 - Item Level Versioning exposes personal data](#)

Version History				
Version	Item	Editor	Date	Summary
2	10673/138*	Demo Administrator	2012-10-23T12:11:46Z	Second version of this item - nothing actually changed
1	10673/138.1	Demo Administrator	2012-10-23T12:10:04Z	

*Selected version

Credits

The initial contribution of Item Level Versioning to DSpace 3.0 was implemented by [@mire](#) with kind support from:

- [MBLWHOI Library](#)
- [Woods Hole Oceanographic Institution](#)
- [Marine Biology Laboratory, Center for Library and Informatics, History and Philosophy of Science program](#)
- [Arizona State University, Center for Biology and Society](#)
- [Dryad](#)

The JSP UI compatibility has been added in DSpace 4.0 by [CINECA](#)

4.4.5 Mapping Items

- [Introduction](#)
- [Using the Item Mapper](#)
- [Implications](#)
 - [Mapping collection vs Owning collection](#)
 - [Mapping an item does not modify access rights](#)

Introduction

The Item Mapper is a tool in the DSpace web user interface allowing repository managers to display the same item in multiple collections at once. Thanks to this feature, a repository manager is not forced to duplicate items to display them in different collections

Using the Item Mapper

In the XML User Interface, the item mapper can be accessed from the "Context" menu from a collection homepage.

In the JSP User Interface, the item mapper can be accessed from the "Admin Tools" menu on the right side of a collection homepage.

[DSpace Demo Repository](#) >

Item Mapper - Map Items from Other Collections

Collection: "Peer reviewed articles"

There are 0 items owned by this collection, and 0 items mapped in from other collections.

Import By Author Match

Enter part of an author's name for a list of matching items

Browse Items Imported From Collections:

Click on collection names to browse for items to remove that were mapped in from that collection.

This collection has no items mapped into it.

Item Mapper - JSPUI Interface

Item Mapper - Map Items from Other Collections

Collection: "Peer reviewed articles"

This is the item mapper tool that allows collection administrators to map items from other collections into this collection. You can search for items from other collections and map them, or browse the list of currently mapped items.

Statistics: 0 of 0 items in this collection are mapped in from other collections

Search:

Item Mapper - XMLUI Interface

The item mapper offers an interface to search for items in the repository with the goal of mapping them to the collection from where you accessed the Item Mapper. While the JSPUI only offers a search for author names, the XMLUI Item Mapper offers a broader search.

The list of items mapped into the current collection can be consulted through the Item Mapper page. While JSPUI immediately shows the list of mapped items, the XMLUI requires you to click "Browse mapped items" in order to access the list.

The list of mapped items provides the functionality to remove the mapping for selected items.

Implications

Mapping collection vs Owing collection

The relation between an item and the collection in which it is mapped is different from the relation that this item has with the collection to which it was originally submitted. This second collection is referred to as the "owning" collection. When an item is deleted from the owning collection, it automatically disappears from the mapping collection. From within the mapping collection, the only thing that can be deleted is the mapping relation. Removing this mapping relation does not affect the presence of the item in the owning collection.

Mapping an item does not modify access rights

When an item gets mapped into a collection, it does not receive new access rights. It retains the authorizations that it inherited from the collection that "owns" it. Collection admins who do not have read access to an item will not be able to map them to other collections.

4.4.6 Metadata Recommendations

```
/*<![CDATA[* / div.rbtoc1424986269829 {padding: 0px;}
div.rbtoc1424986269829 ul {list-style: none;margin-left: 0px;}
div.rbtoc1424986269829 li {margin-left: 0px;padding-left: 0px;} /*]]>*/
```

- 1 [Recommended Metadata Fields](#)
- 2 [Local Fields](#)

Recommended Metadata Fields

DSpace provides a broad list of metadata fields out of the box (see: [Metadata and Bitstream Format Registries](#)), and a variety of options for adding content to DSpace (both from the UI and from other services). No matter which Ingest option you use, DSpace recommends ensuring that the following metadata fields are specified:

- **Title** (`dc.title`)
 - When submitting an Item via the DSpace web user interface, this field is **required**.
 - If you add an Item to DSpace through another means (SWORD, etc), it is recommended to specify a title for an Item. Without a title, the Item will show up in DSpace as "Untitled".
- **Publication Date** (`dc.date.issued`)
 - When submitting an Item via the DSpace web user interface, this field is **required** (by default).
 - However, your System Administrator can choose to enable the "Initial Questions" step within the [Submission User Interface](#). Enabling this step will cause the following to occur: If the item is said to be "published", then the Publication Date will be required. If the item is said to be "unpublished" then the Publication Date will be auto-set to today's date (date of submission). *WARNING:* Google Scholar has recommended against automatically assigning this "dc.date.issued" field to the date of submission as it often results in incorrect dates in Google Scholar results. See [DS-1481](#) and [DS-1745](#) for more details.
 - If you add an Item to DSpace through another means (SWORD, etc), it is recommended to specify the date in which an Item was published, in [ISO-8601](#) (e.g. 2007, 2008-01, or 2011-03-04). This ensures DSpace can accurately report the publication date to services like Google Scholar. If an item is unpublished, you can either choose to leave this blank, or pass in the literal string "today" (which will tell DSpace to automatically set it to the date of ingest)



DSpace will not auto-assign a "dc.date.issued"

As of DSpace 4.0, the system will not assign a "dc.date.issued" when unspecified. Previous versions of DSpace (3.0 or below) would set "dc.date.issued" to the date of accession (dc.date.accessioned), if it was unspecified during ingest.

If you are adding content to DSpace without using the DSpace web user interface, there are two recommended options for assigning "dc.date.issued"

- If the item is previously published before, please set "dc.date.issued" to the date of publication in [ISO-8601](#) (e.g. 2007, 2008-01, or 2011-03-04)
- If the item has never been previously published, you may set "dc.date.issued='today'" (the literal string "today"). This will cause DSpace to automatically assign "dc.date.issued" to the date of accession (dc.date.accessioned), as it did previously
 - You can also chose to leave "dc.date.issued" as unspecified, but then the new Item will have an empty date within DSpace.

Obviously, we recommend specifying as much metadata as you can about a new Item. For a full list of supported metadata fields, please see: [Metadata and Bitstream Format Registries](#)

Local Fields

You may encounter situations in which you will require an appropriate place to store information that does not immediately fit with the description of a field in the default registry. The recommended practice in this situation is to create new fields in a separate schema. You can choose your own name and prefix for this schema such as *local*. or *myuni*.

It is generally discouraged to use any of the fields from the default schema as a place to store information that doesn't correspond with the fields description. This is especially true if you are ever considering the option to open up your repository metadata for external harvesting.

4.4.7 Moving Items

- 1 [Moving Items via Web UI](#)
- 2 [Moving Items via the Batch Metadata Editor](#)

Moving Items via Web UI

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

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

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

Moving Items via the Batch Metadata Editor

Items may also be moved in bulk by using the CSV batch metadata editor (see [Editing Collection Membership](#) section under [Batch Metadata Editing](#)).

4.4.8 ORCID Integration

- 1 [Introduction](#)
- 2 [Use case and high level benefits](#)
- 3 [Enabling the ORCID authority control](#)
- 4 [Importing existing authors & keeping the index up to date](#)
 - 4.1 [Different possible use cases for Index-authority script](#)
 - 4.1.1 [Metadata value WITHOUT authority key in metadata](#)
 - 4.1.2 [Metadata that already has an authority key from an external source \(NOT auto-generated by DSpace\)](#)
 - 4.1.3 [Metadata that has already a new dspace generated uid authority key](#)
 - 4.1.4 [Processing on records in the authority cache](#)
 - 4.2 [Submission of new DSpace items - Author lookup](#)
 - 4.3 [Admin Edit Item](#)
 - 4.4 [Editing existing items using Batch CSV Editing](#)
 - 4.5 [Storage of related metadata](#)
- 5 [Configuration](#)
- 6 [Adding additional fields under ORCID](#)
- 7 [Integration with other systems beside ORCID](#)
- 8 [FAQ](#)
 - 8.1 [Which information from ORCID is currently indexed in the authority cache?](#)
 - 8.2 [How can I index additional fields in the authority cache?](#)
 - 8.3 [How can I use the information stored in the authority cache?](#)
 - 8.4 [How to add additional metadata fields in the authority cache that are not related to ORCID?](#)
 - 8.5 [What happens to data if another authority control was already present?](#)

Introduction

The ORCID integration adds ORCID compatibility to the existing solutions for [Authority control in DSpace](#). String names of authors are still being stored in DSpace metadata. The authority key field is leveraged to store a uniquely generated internal ID that links the author to more extended metadata, including the ORCID ID and alternative author names.

This extended metadata is stored and managed in a dedicated SOLR index, the DSpace authority cache.

Use case and high level benefits

The vision behind this project consists of the following two aspects:

Lowering the threshold to adopt ORCID for the members of the DSpace community

ORCID's API has enabled developers across the globe to build points of integration between ORCID and third party applications. Up until today, this meant that members of the DSpace community were still required to implement front-end and back-end modifications to the DSpace source code in order to leverage these APIs. As DSpace aims to provide turnkey Institutional Repository functionality, the platform is expected to provide more functionality out of the box. Only an elite selection of members in the DSpace community has software development resources readily available to implement this kind of functionality. By contributing a solution directly to the core DSpace codebase, this threshold to adopt ORCID functionality in DSpace repositories is effectively lowered. The ultimate goal is to allow easy adoption of ORCID without customization of the DSpace software, by allowing repository administrators to enable or disable functionality by means of user friendly configuration.

Address generic use cases with appealing end user functionality

This proposal aims to provide user friendly features for both repository administrators as well as non- technical end users of the system. The addition of ORCID functionality to DSpace should not come at the cost of making the system more difficult for administrators and end users to use. Scope With this vision in mind, the project partners wanted to tackle the first phases for repository managers of existing DSpace repositories: ensuring that ORCID's are properly associated with new works entering the system, as well as providing functionality to efficiently batch-update content already existing in the system, with unambiguous author identity information.

Enabling the ORCID authority control

JSPUI Support

In DSpace 5.0 the functionality only includes user interface functionality for the DSpace XML User Interface.

XMLUI Theme Support

In DSpace 5.0 the functionality only adds support for the XMLUI Mirage and Mirage 2 themes. Older XMLUI themes including Kubrick, Reference and Classic are currently unsupported.

If you wish to enable this feature, some changes are required to the `dspace.cfg` file. The first step is to activate the authority as a valid option for authority control, this is done by adding/setting an additional plugin in the `plugin.named.org.dspace.content.authority.ChoiceAuthority` property. An example of this can be found below.

```
plugin.named.org.dspace.content.authority.ChoiceAuthority = \  
org.dspace.content.authority.SolrAuthority = SolrAuthorAuthority
```

The feature relies on the following configuration parameters in `dspace.cfg`. To activate the default settings it suffices to remove the comment hashes ("`#`") for the following lines. See the section at the bottom of this page what these parameters mean exactly and how you can tweak the configuration.

```
solr.authority.server=${solr.server}/authority  
choices.plugin.dc.contributor.author = SolrAuthorAuthority  
choices.presentation.dc.contributor.author = authorLookup  
authority.controlled.dc.contributor.author = true  
authority.author.indexer.field.1=dc.contributor.author
```

The final part of configuration is to add the authority consumer in front of the list of event consumers. Add "authority" in front of the list as displayed below.

```
event.dispatcher.default.consumers = authority, versioning, discovery, eperson, harvester
```

Importing existing authors & keeping the index up to date

When first enabled the authority index will be empty, to populate the authority index run the following script:

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

This will iterate over every metadata under authority control and create records of them in the authority index. The metadata without an authority key will each be updated with an auto generated authority key. These will not be matched in any way with other existing records. The metadata with an authority key that does not already exist in the index will be indexed with those authority keys. The metadata with an authority key that already exist in the index will be re-indexed the same way. These records remain unchanged.

Different possible use cases for Index-authority script

Metadata value WITHOUT authority key in metadata

"Luyten, Bram" is present in the metadata without any authority key.

GOAL: "Luyten, Bram" gets added in the cache ONCE

All occurrences of "Luyten, Bram" in the DSpace item metadata will become linked with the same generated uid.

Metadata that already has an authority key from an external source (NOT auto-generated by DSpace)

"Snyers, Antoine" is present with authority key "u12345"

The old authority key needs to be preserved in the item metadata and duplicated in the cache. “u12345” will be copied to the authority cache and used as the authority key there.

Metadata that has already a new dspace generated uid authority key

Item metadata already contains an author with name “Haak, Danielle” and a uid in the authority field 3dda2571-6be8-4102-a47b-5748531ae286

This uid is preserved and no new record is being created in the authority index.

Processing on records in the authority cache

Running this script again will update the index and keep the index clean. For example if an author occurs in a single item and that item is deleted the script will need to be run again to remove it from the index. When run again it will remove all records that no longer have a link to existing authors in the database.

Usage in DSpace

Submission of new DSpace items - Author lookup

The submissions forms have not changed much. The only thing you can notice is an extra button next to the input fields for the author names. Next to the Add button, which is common for all repeatable fields, there is the Lookup & Add button.

Item submission

Describe Describe Access Upload Review CC License License Complete

Describe Item

Authors:

Last name, e.g. *Smith*

First name(s) + "Jr", e.g. *Donald Jr*

Add

Lookup

Enter the names of the authors of this item below.

Person lookup ×

Search:

Name	
Hancock, Roeland	<ul style="list-style-type: none"> ◦ last name: Hancock ◦ first name: Roeland ◦ orcid: 0000-0001-8932-6872 <p>Items in this repository: view items</p> <p style="text-align: right;"><input type="button" value="Add This Person"/></p>
<i>Hancock, Yvette</i>	
<i>Hancock, Katy</i>	
<i>Kleiner-Hancock, Heather</i>	
<i>Hancock, Mark</i>	
<i>Hancock, Steven</i>	
<i>Hancock, Karen</i>	
<i>Hancock, Kenneth</i>	
<i>Hancock, Travis</i>	
<i>HANCOCK, GALEN</i>	

Showing 10 results.

It's by clicking on that button that the Look-up User Interface appears. If an author name was filled in but not added yet, the Lookup User Interface will immediately perform a search for that name. Otherwise the search field remains empty and a list of known authors is displayed. The list of authors is updated as you type in the search box.

Authors that already appear somewhere in the repository are differentiated from the authors that have been retrieved from ORCID.

The authors retrieved from ORCID have their name italicized and they're listed after the authors that are found in the repository.

Name
Fingert, John
Chamberland, John
John Hunter, John Hunter
Coupland, John
Finnegan, John
wilbanks, john
Roemer, John
Wagner, John
Sproule, John
Volkman, John
<i>Vuchetich, John</i>
<i>Gales, John</i>

Click on one of these names to see more information about them. The message "There's no one selected" will vanish, making room for the author's information. The available information can vary: Authors imported from ORCID have an orcid where the others do not. Authors that have been added without look-up only show their last name and first name.

To add an author from the Look-up User Interface, you select the author in the list and then you click on the "Add This Person" button.

To add an author without look-up, you don't go through the Look-up User Interface. Instead you simply use the "Add" button in the submissions forms.

Admin Edit Item

In the edit metadata page, under the values for the dc.contributor.author fields, an extra line shows the author ID together with a lock icon and a Lookup button. The author ID cannot be changed manually. However the Lookup button will help you change the author name and ID at the same time.

Clicking the Lookup button brings back the Lookup User Interface. This works just the same way as in the submission forms.

Metadata

Remove	Name	Value
<input type="checkbox"/>	dc.contributor.author	<div style="border: 1px solid #ccc; padding: 5px; min-height: 40px;">Dutey, Jean</div> <div style="display: flex; align-items: center; margin-top: 5px;"> ↶ 263117b9-c40d-42e7-8 Lookup </div>

Person lookup
✕

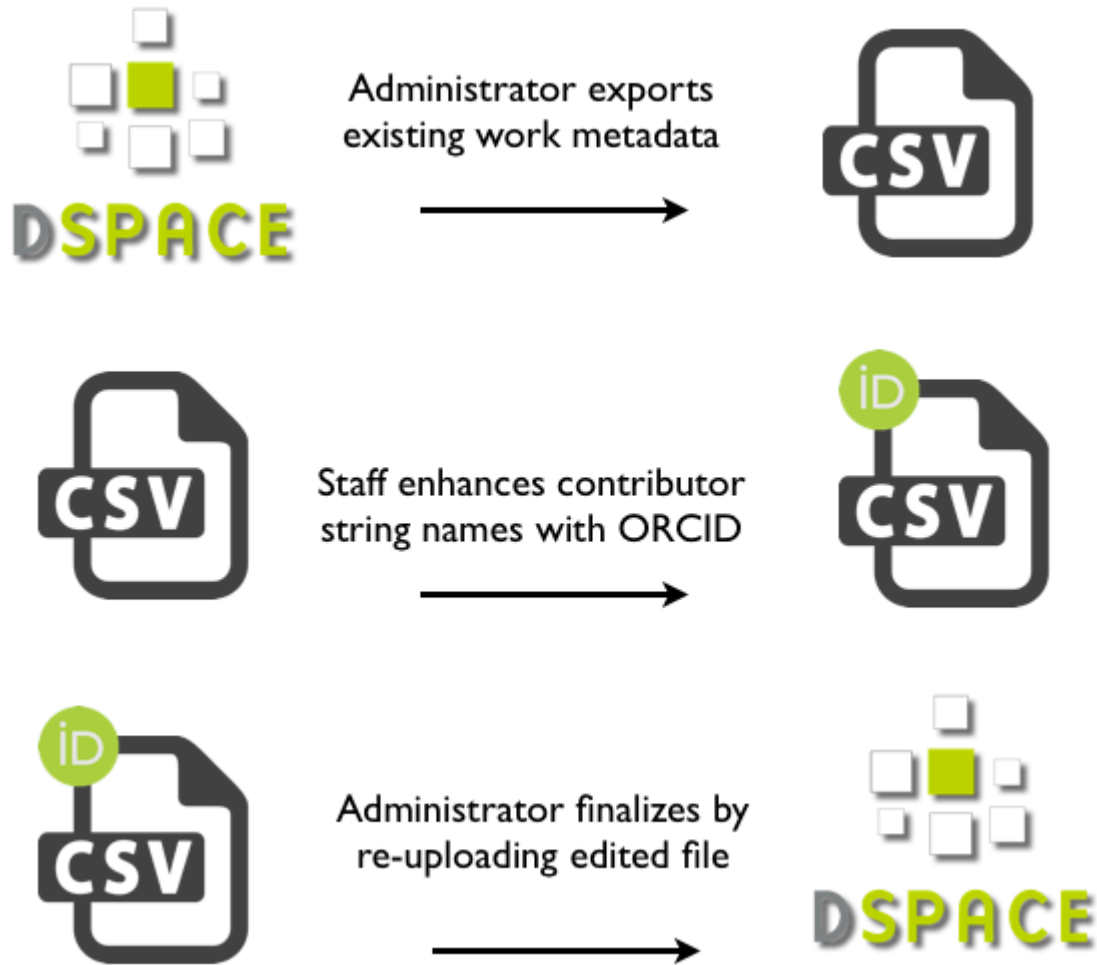
Search:

Name	
Dutey, Jean	<ul style="list-style-type: none"> ◦ last name: Dutey ◦ first name: Jean <p>Items in this repository: view items</p> <div style="text-align: right; margin-top: 10px;"> Add This Person </div>

Showing 1 results.
show more

Editing existing items using Batch CSV Editing

Instructions on how to use the Batch CSV Editing are found [on the Batch Metadata Editing documentation page](#).



ORCID Integration is provided through the Batch CSV Editing feature with an extra available headers "ORCID: dc.contributor.author". The usual CSV headers only contain the metadata fields: e.g. "dc.contributor.author". In addition to the traditional header, another dc.contributor.author header can be added with the "ORCID:" prefix. The values in this column are supposed to be ORCID IDs.

id	collection	dc.title	ORCID: dc.contributor.author	dc.contributor.author
+	123456789/2	ORCID CSV import example	0000-0001-8932-6872 0000-0001-9152-849X	Smith, Benjamin

For each of the ORCID authors a lookup will be done and their names will be added to the metadata. All the non-ORCID authors will be added as well. The authority keys and solr records are added when the reported changes are applied.

Notice

Upload successful

Import Metadata

Pending changes are listed below for review

New item

Add to owning collection 123456789/2 (uiop)

Add: (dc.title) ORCID CSV import example

Add: (dc.contributor.author) Smith, Benjamin

Add: (dc.contributor.author) Hancock, Roeland

Add: (dc.contributor.author) Vandekerckhove, Bram

Apply changes

Return

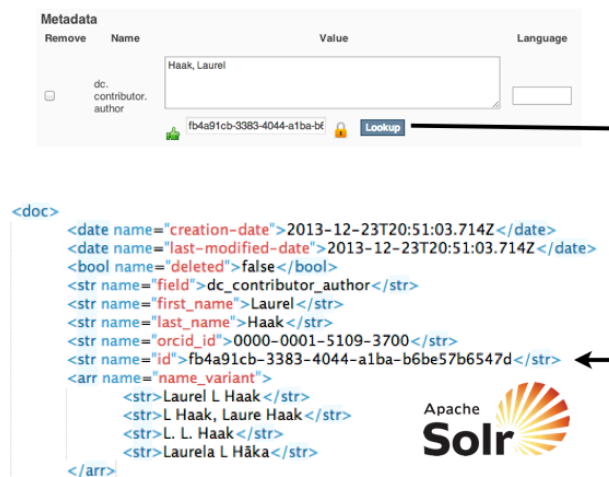
Storage of related metadata

ORCID authorities not only link a digital identifier to a name. It regroups a load of metadata going from alternative names and email addresses to keywords about their works and much more. The metadata is obtained by querying the ORCID web services. In order to avoid querying the ORCID web services every time, all these related metadata is gathered in a "metadata authority cache" that DSpace can access directly.

In practice the cache is provided by an apache solr server. When a look-up is made and an author is chosen that is not yet in the cache, a record is created from an ORCID profile and added to the cache with the list of related metadata. The value of the Dublin Core metadata is based on the first and last name as they are set in the ORCID profile. The authority key for this value links directly to the solr document's id. DSpace does not provide a way to edit these records manually.

String representation of the name in standard Dublin Core metadata enhanced with DSpace compliant authority id

Second level "metadata authority cache" stores extended contributor metadata, including ORCID ID and alternative names



```

<doc>
  <date name="creation-date">2013-12-23T20:51:03.714Z</date>
  <date name="last-modified-date">2013-12-23T20:51:03.714Z</date>
  <bool name="deleted">>false</bool>
  <str name="field">dc_contributor_author</str>
  <str name="first_name">Laurel</str>
  <str name="last_name">Haak</str>
  <str name="orcid_id">0000-0001-5109-3700</str>
  <str name="id">fb4a91cb-3383-4044-a1ba-b6be57b6547d</str>
  <arr name="name_variant">
    <str>Laurel L Haak</str>
    <str>L Haak, Laure Haak</str>
    <str>L. L. Haak</str>
    <str>Laurela L Häka</str>
  </arr>
</doc>
  
```

The information in the authority cache can be updated by running the following command line operation:

Command used:	<code>[dspace]/bin/dspace dsrun org.dspace.authority.UpdateAuth</code>
Arguments	description
-i	update specific solr records with the given internal ids (comma-separated)
-h	prints this help message

This will iterate over every solr record currently in use (unless the -i argument is provided), query the ORCID web service for the latest data and update the information in the cache. If configured, the script will also update the metadata of the items in the repository where applicable.

The configuration property can be set in `config/modules/solrauthority.cfg`:

```
auto-update-items = false | true
```

When set to true and this script is run, if an authority record's information is updated the whole repository will be scanned for this authority. Every metadata field with this authority key will be updated with the value of the updated authority record.

Configuration

In the [Enabling the ORCID authority control](#) section, you have been told to add this block of configuration.

```
solr.authority.server=${solr.server}/authority
choices.plugin.dc.contributor.author = SolrAuthorAuthority
choices.presentation.dc.contributor.author = authorLookup
authority.controlled.dc.contributor.author = true
authority.author.indexer.field.1=dc.contributor.author
```

The ORCID Integration feature is an extension on the authority control in DSpace. Most of these properties are extensively explained [on the Authority Control of Metadata Values documentation page](#). These will be revisited but first we cover the properties that have been newly added.

- The `solr.authority.server` is the url to the solr core. Usually this would be on the `solr.server` next to the oai, search and statistics cores.
- `authority.author.indexer.field.1` and the subsequent increments configure which fields will be indexed in the authority cache. However before adding extra fields into the solr cache, please read the section about [Adding additional fields under ORCID](#).

That's it for the novelties. Moving on to the generic authority control properties:

- With the `authority.controlled` property every metadata field that needs to be authority controlled is configured. This involves every type of authority control, not only the fields for ORCID integration.
- The `choices.plugin` should be configured for each metadata field under authority control. Setting the value on `SolrAuthorAuthority` tells DSpace to use the solr authority cache for this metadatafield, cfr. [Storage of related metadata](#).
- The `choices.presentation` should be configured for each metadata field as well. The traditional values for this property are `select|suggest|lookup`. A new value, `authorLookup`, has been added to be used in combination with the `SolrAuthorAuthority` choices plugin. While the other values can still be used, the `authorLookup` provides a richer user interface in the form of a popup on the submission page.
- The browse indexes need to point to the new authority-controlled index: `webui.browse.index.2 = author:metadata:dc.contributor.*,dc.creator:text` should become **`webui.browse.index.2 = author:metadataAuthority:dc.contributor.author:text`**
- More existing configuration properties are available but their values are independent of this feature and their default values are usually fine: `choices.closed`, `authority.required`, `authority.minconfidence`.

For the cache update script, one property can be set in `config/modules/solrauthority.cfg`:

```
auto-update-items = false | true
```

The default value for when the property is missing is false.

The final part of configuration is to add the authority consumer in front of the list of event consumers. Add "authority" in front of the list as displayed below.

```
event.dispatcher.default.consumers = authority, versioning, discovery, eperson, harvester
```

Without the consumer there is no automatic indexing of the authority cache and the metadata will not even have authority keys.

Changes to the configuration always require a server restart before they're in effect.

Adding additional fields under ORCID

Other metadata fields besides "dc.contributor.author" can benefit from the ORCID authority control at the same time. Here is an example of how to get the same ORCID functionality for the "dc.contributor.editor" metadata field assuming that "dc.contributor.author" is already configured correctly. It can be achieved by modifying configuration files only.

First add the same configuration fields that have been added for the "dc.contributor.author"

```
choices.plugin.dc.contributor.editor = SolrAuthorAuthority
```

```

choices.presentation.dc.contributor.editor = authorLookup
authority.controlled.dc.contributor.editor = true
authority.author.indexer.field.1=dc.contributor.author
authority.author.indexer.field.2=dc.contributor.editor
  
```

This is enough to get the look-up interface on the submission page and on the edit metadata page. The authority keys will be added and indexed with the information from orcid just as it happens with the Authors.

But you're not completely done yet, There is one more configuration step. Because now when adding new editors in the metadata that are not retrieved through the external look-up, their first and last name will not be displayed in the look-up interface next time you look for them.

To fix this, open the file at `config/spring/api/authority-services.xml` and find this spring bean:

```

<bean name="AuthorityTypes" class="org.dspace.authority.AuthorityTypes">
  <property name="types">
    <list>
      <bean class="org.dspace.authority.orcid.OrcidAuthorityValue"/>
      <bean class="org.dspace.authority.PersonAuthorityValue"/>
    </list>
  </property>
  <property name="fieldDefaults">
    <map>
      <entry key="dc_contributor_author">
        <bean class="org.dspace.authority.PersonAuthorityValue"/>
      </entry>
    </map>
  </property>
</bean>
  
```

The map inside the "fieldDefaults" property needs an additional entry for the editor field:

```

<entry key="dc_contributor_editor">
  <bean class="org.dspace.authority.PersonAuthorityValue"/>
</entry>
  
```

With this last change everything is set up to work correctly. The rest of this configuration file is meant for JAVA developers that wish to provide [integration with other systems beside ORCID](#). Developers that wish to display other fields than first and last name can also have a look in that section.

Note: Each metadata field has a separate set of authority records. Authority keys are not shared between different metadata fields. E. g. multiple `dc.contributor.author` can have the same authority key and point to the same authority record in the cache. But when an ORCID is chosen for a `dc.contributor.editor` field, a separate record is made in the cache. Both records are updated from the same source and will contain the same information. The difference is that when performing a look-up of a person that has been introduced as an

authority for an author field but not yet as an editor, it will show as record that is not yet present in the repository cache.

Integration with other systems beside ORCID

The authority cache and look-up functionality can be extended to use other sources than ORCID or to show more information in the look-up interface. However some JAVA development is necessary for this. Specific instructions can be found in the readme file of the [org.dspace.authority package](#).

FAQ

Which information from ORCID is currently indexed in the authority cache?

Here is a breakdown of the fields stored in the solr cache.

The system/dspace related fields are: *id, field, value, deleted, creation_date, last_modified_date, authority_type*.

The fields with data coming directly from ORCID are: *first_name, last_name, name_variant, orcid_id, label_researcher_url, label_keyword, label_external_identifier, label_biography, label_country*. The field *all_labels* contains all the values from the other fields starting with "label_".

How can I index additional fields in the authority cache?

There is currently no configuration to control which fields are indexed. The only way to achieve this is to modify the source code.

List of the files at work for this job:

config/spring/api/orcid-authority-services.xml: OrcidSource contains the URL for orcid's REST API.
org.dspace.authority.orcid.Orcid makes the REST call
+ org.dspace.authority.orcid.xml.XMLtoBio converts the received XML to a java object (Bio).
+ org.dspace.authority.orcid.model.Bio
+ org.dspace.authority.orcid.OrcidAuthorityValue#create(org.dspace.authority.orcid.model.Bio) inserts all the values from Bio into the AuthorityValue subclass.
+ org.dspace.authority.orcid.OrcidAuthorityValue#getSolrInputDocument defines what's included in solr.

The files preceded with a '+' would be necessary to modify to add more info into the cache.

How can I use the information stored in the authority cache?

The look-up UI is currently the only place this information is sent to. However just a limited number of fields are sent. The place in the source code to modify to get more fields there is `org.dspace.authority.orcid.OrcidAuthorityValue#choiceSelectMap`. This is also documented in [the readme of the org.dspace.authority package](#).

How to add additional metadata fields in the authority cache that are not related to ORCID?

Make the same configuration step as for [adding additional fields under ORCID](#). Currently the ORCID suggestions cannot be turned off for specific fields, that would require custom code.

What happens to data if another authority control was already present?

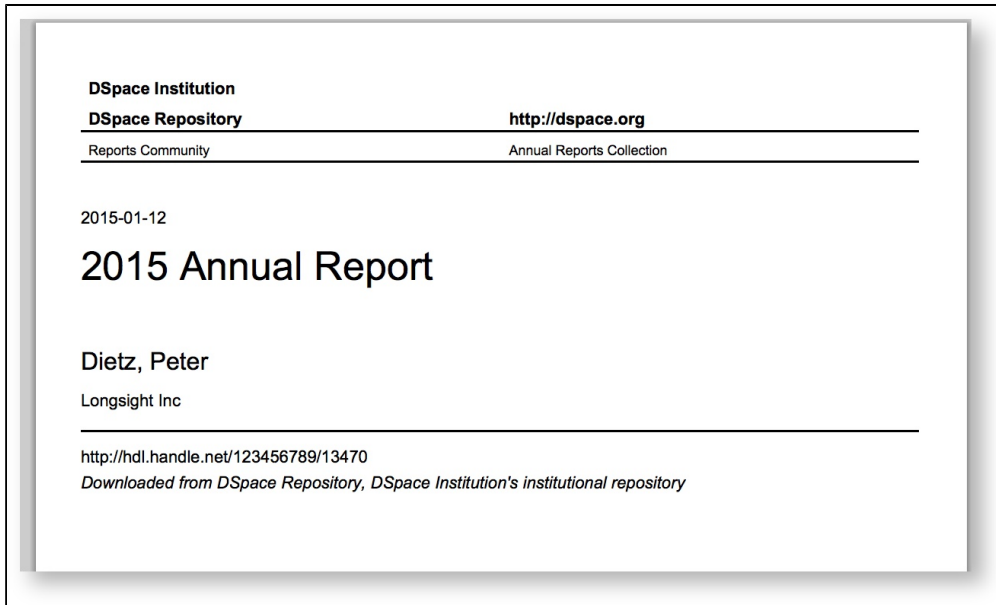
As long as the metadata does not get indexed, there will be no changes. However every time any metadata of an item is modified, the metadata under authority control for that item will be re-indexed. When that happens a record will be inserted in the solr cache. That record's ID will be the authority key of the metadata. This can be done for all metadata at once with the index-authority script.

In short: authority keys that exist prior to enabling the solrauthority are kept. They just won't show in the look-up until they are indexed.

4.4.9 PDF Citation Cover Page

Adding a cover page to retrieved documents from DSpace that include additional citation information has been sought, as documents uploaded to the repository might have had their context stripped from them, when they are just a PDF. Context that might have surrounded the document would be the journal, publisher, edition, and more. Without that information, the document might just be a few pages of text, with no way to piece it together. Since repository policy might be to include this information as metadata to the Item, this metadata can be added to the citation cover page, so that the derivative PDF includes all of this information.

The citation cover page works by only storing the original PDF in DSpace, and then generating the citation-cover-page PDF on-the-fly. An alternative set up would be to run the PDF Citation Coverpage Curation Task on the DSpace repository contents, and then disseminate the pre-generated citation-version instead of generating it on the fly.



Screenshot of generated citation cover page

Configuration settings for Citation Cover Page

In the `{dspace.dir}/config/modules/disseminate-citation.cfg` file review the following fields to make sure they are uncommented:

Property:	enable_globally
Example Values:	<code>enable_globally = true</code>
Informational Note:	Boolean to determine is citation-functionality is enabled globally for entire site. This will enable the citation cover page generator for all PDFs. Default: disabled
Property:	enabled_collections
Example Values:	<code>enabled_collections = 1811/123, 1811/234</code>
Informational Note:	List of collection handles to enable the cover page generator for bitstreams within. Default: blank
Property:	enabled_communities
Example Values:	<code>enabled_communities = 1811/222, 1811/333</code>

Property:	enable_globally
Informational Note:	List of community handles to enable the cover page generator for bitstreams within. Default: blank
Property:	citation_as_first_page
Example Values:	citation_as_first_page = true
Informational Note:	Should the citation page be the first page cover (true), or the last page (false). Default: true, (first page)
Property:	header1
Example Values:	header1 = University of Higher Education
Informational Note:	First row of header, perhaps for institution / university name Default: DSpace Institution
Property:	header2
Example Values:	header2 = Scholar Archive, http://archive.example.com
Informational Note:	Second row of header, perhaps put your DSpace instance branded name, and url to your DSpace. A comma is used to separate instance name, and the URL Default: DSpace Repository, http://dspace.org
Property:	fields
Example Values:	fields = dc.date.issued, dc.title, dc.creator, dc.contributor.author, dc.publisher, _line_, dc.identifier.citation, dc.identifier.uri
Informational Note:	Metadata fields to display on the citation PDF. Specify in schema.element.qualifier form, and separate fields by a comma. If you want to have a horizontal line break, use _line_ Default: dc.date.issued,dc.title,dc.creator,dc.contributor.author,dc.publisher,_line_, dc.identifier.citation,dc.identifier.uri
Property:	footer
Example Values:	footer = Downloaded from Scholar Archive at University of Higher Education, an open access institutional repository. All Rights Reserved.

Property:	enable_globally
Informational Note:	Footer text at the bottom of the citation page. It might be some type of license or copyright information, or just letting the recipient know where they downloaded the file from. Default: Downloaded from DSpace Repository, DSpace Institution's institutional repository

A known issue with the current implementation of the PDF Citation Cover Page is that primarily only English/Roman characters are supported. This is due to a limitation in the tool used to generate PDFs.

4.4.10 Updating Items via Simple Archive Format

- 1 [Item Update Tool](#)
 - 1.1 [DSpace Simple Archive Format](#)
 - 1.2 [ItemUpdate Commands](#)
 - 1.2.1 [CLI Examples](#)

Item Update Tool

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

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

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

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

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

DSpace Simple Archive Format

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

The user is referred to the previous section [DSpace Simple Archive Format](#).

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

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

ItemUpdate Commands

Command used:	<code>[dspace]/bin/dspace itemupdate</code>
Java class:	<code>org.dspace.app.itemupdate.ItemUpdate</code>
Arguments short and (long) forms:	Description
<code>-a</code> or <code>--addmetadata [metadata element]</code>	Repeatable for multiple elements. The metadata element should be in the form <code>dc.x</code> or <code>dc.x.y</code> . The mandatory argument indicates the metadata fields in the <code>dublin_core.xml</code> file to be added unless already present (multiple fields should be separated by a semicolon ';'). However, duplicate fields will not be added to the item metadata without warning or error.
<code>-d</code> or <code>--deletemetadata [metadata element]</code>	Repeatable for multiple elements. All metadata fields matching the element will be deleted.
<code>-A</code> or <code>--addbitstreams</code>	Adds bitstreams listed in the contents file with the bitstream metadata cited there.
<code>-D</code> or <code>--deletebitstreams [filter plugin classname or alias]</code>	Not repeatable. With no argument, this operation deletes bitstreams listed in the <code>deletes_contents</code> file. Only bitstream IDs are recognized identifiers for this operation. The optional filter argument is the classname of an implementation of <code>org.dspace.app.itemupdate.BitstreamFilter</code> class to identify files for deletion or one of the aliases (e.g. ORIGINAL, ORIGINAL_AND_DERIVATIVES, TEXT, THUMBNAIL) which reference existing filters based on membership in a bundle of that name. In this case, the <code>delete_contents</code> file is not required for any item. The filter properties file will contains properties pertinent to the particular filter used. Multiple filters are not allowed.
<code>-h</code> or <code>--help</code>	Displays brief command line help.
<code>-e</code> or <code>--eperson</code>	Email address of the person or the user's database ID (Required)
<code>-s</code> or <code>--source</code>	Directory archive to process (Required)
<code>-i</code> or <code>--itemfield</code>	Specifies the metadata field that contains the item's identifier; Default value is " <code>dc.identifier.uri</code> " (Optional)
<code>-t</code> or <code>--test</code>	Runs the process in test mode with logging. But no changes applied to the DSpace instance. (Optional)
<code>-P</code> or <code>--provenance</code>	

	Prevents any changes to the provenance field to represent changes in the bitstream content resulting from an Add or Delete. In other words, when this flag is specified, no new provenance information is added to the DSpace Item when adding/deleting a bitstream. No provenance statements are written for thumbnails or text derivative bitstreams, in keeping with the practice of MediaFilterManager. (Optional)
-F or -- filter-properties	The filter properties files to be used by the delete bitstreams action (Optional)
-v or --verbose	Turn on verbose logging.

CLI Examples

Adding Metadata:

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

This will update all DSpace Items listed in your archive directory, adding a new dc.description metadata field. Items will be located in DSpace based on the handle found in 'dc.identifier.uri' (since the -i argument wasn't used, the default metadata field, dc.identifier.uri, from the dublin_core.xml file in the archive folder, is used).

4.5 Managing Community Hierarchy

- 1 [Sub-Community Management](#)

4.5.1 Sub-Community Management

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

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

Command used:

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

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

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

(or using the short form)

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

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

The reverse operation looks like this:

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

(or using the short form)

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

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

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

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

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

4.6 Statistics and Metrics

4.6.1 DSpace Google Analytics Statistics

Google Analytics Recording

For a number of years now it has been possible to record User Interface traffic by enabling the recording of Google Analytics data within DSpace using the `jspui.google.analytics.key` or `xmlui.google.analytics.key` in the DSpace configuration file `dspace.cfg`. Until DSpace version 5.0 only User Interface activity could be recorded, that is to say that downloads initiated straight from a Google search (or any other search engine) were not recorded. As of DSpace version 5.0 downloads are now recorded as Google 'Events', so that all item page views and bitstream downloads are now recorded.

Google Analytics Reporting

As of DSpace version 5.0 it has also become possible to expose that recorded Google Analytics data within DSpace. At present this is only available to those sites using themes based on the XMLUI Mirage2 theme but it is hoped that further development will result in it being available for other XMLUI themes and for the JSPUI. The data is retrieved from Google using the Google Analytics Reporting API v3. This feature is disabled by default, to enable it please follow the instructions below.

Please read the documentation found at <https://developers.google.com/analytics/devguides/reporting/core/v3/> and <https://developers.google.com/accounts/docs/OAuth2ServiceAccount>. It is the definitive documentation, however, it is over detailed for our purposes so the critical steps are summarised below. The theory is that as a developer you would create a Google project, write your application and store the code in the Google code repository, then create a Google Service Account which your application could use to retrieve data from the Google Analytics API. In our case we already have our application, DSpace, but we still have to go through the motions of creating a project in order to be able to generate the Service Account which we need to allow DSpace to talk to the Google Analytics API.

1. Enable the Google Analytics XMLUI aspect by editing the configuration file `{dspace.dir}/config/xmlui.xconf`.
2. Logon to the Google Developers Console <https://console.developers.google.com/project> with whatever email address you use to access/manage your existing Google Analytics account(s).

Informational Note:	The email address automatically generated when you created the Service Account.
Property:	certificate.location
Example Value:	/home/example/dslweb--privatekey.p12
Informational Note:	The certificate file automatically generated when you created the Service Account.
Property:	authorization.admin.usage
Example Value:	true
Informational Note:	Control if the statistics pages should be only shown to authorized users. If enabled, only the administrators for the DSpaceObject will be able to view the statistics. If disabled, anyone with READ permissions on the DSpaceObject will be able to view the statistics.

4.6.2 Elasticsearch Usage Statistics

Added in DSpace 3.0 is an optional statistics engine using [Elasticsearch](#), which may be enabled as an alternative to the default [SOLR Statistics](#) engine (based on [Apache SOLR](#)). The motivation for adding Elasticsearch was to find an alternative statistics processing engine that would handle the workload of a large amount of statistics data. Additionally, the Elasticsearch statistics display offers another method for creating statistical queries against your data. Elasticsearch Usage Statistics has been contributed by Peter Dietz of Ohio State University's Knowledge Bank. The data source for Elasticsearch Statistics is DSpace Usage Events, where Usage Event is a view or download of a DSpace Object (Bitstream, Item Page, Collection Page, Community Page). Elasticsearch Statistics is bundled with DSpace, and requires no additional installation of software, it just needs to be enabled. Elasticsearch is only available for use with XMLUI.

What data is being recorded?

The default information below is what DSpace will record about a Usage Event. In DSpace 3.0 the fields of data collected is not configurable through a configuration setting.

Information about the User Requesting the Content

- IP Address
- Time of Request
- DNS / Hostname
- User Agent
- isBot, a flag that DSpace thinks that user is a robot or not

- Geographical Information about where the user is located:
 - Continent
 - Country
 - Country Code
 - City
 - Geographical Latitude/Longitude

Information about the DSpace Resource that was used

- DSpace Object ID
- DSpace Object Type: (Item, Bitstream, Collection, or Community)
- If it is relevant, we also store the hierarchy of where this object exists within DSpace
 - Owning Community
 - Owning Collection
 - Owning Item

Enabling Elasticsearch Statistics

Elasticsearch Statistics is disabled by default in DSpace 3.0, the following steps will enable Elasticsearch so that you can collect data, and present statistics reports.

Modify `dspace/config/xmlui.xconf`, and uncomment the aspect, Statistics Elasticsearch.

Enable Elastic Search Statistics in `dspace/config/xmlui.xconf`

```

<!--
  If you prefer to use "Elasticsearch" Statistics, you can uncomment the below
  aspect and COMMENT OUT the default "Statistics" aspect above.
  You must also enable the ElasticsearchLoggerEventListener.
-->
<!-- <aspect name="Statistics - Elasticsearch" path="resource://aspects/
StatisticsElasticSearch/" /> -->

```

Modify `dspace-xmlui/src/main/webapp/WEB-INF/spring/applicationContext.xml` and uncomment the following code block for `ElasticSearchLoggerEventListener`

Enable `ElasticSearchLoggerEventListener`

```

<!-- Elasticsearch -->
<!--<bean class="org.dspace.statistics.ElasticSearchLoggerEventListener">
  <property name="eventService">
    <ref bean="dspace.eventService" />
  </property>
</bean>-->

```

After making these two changes, you will then need to rebuild and restart DSpace.

Importing Legacy Data into Elasticsearch Statistics

Once Elasticsearch Statistics has been enabled, it will begin adding all new Usage Events to its data store. To import your legacy data, you will need to import the data from the `dspace.log` files. There is no tool yet that converts SOLR statistics data to Elasticsearch statistics data.

From the (Windows / Linux) terminal, you will need to use the DSpace Command Launcher to convert the `dspace.log` files to a statistics log format. Then you will need to import the statistics log format files into DSpace Statistics.

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

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

An example form of this command would be `[dspace]/bin/dspace stats-log-converter -i dspace.log -o statistics.log -m -n`

The Log Importer program takes the intermediate format data produced in the previous step, and imports it into Elasticsearch Statistics.

Command used:	<code>[dspace]/bin/dspace stats-log-importer-elasticsearch</code>
Java class:	<code>org.dspace.statistics.util.StatisticsImporterElasticSearch</code>
Arguments short and long forms):	Description
<code>-i</code> or <code>--in</code>	Input file
<code>-m</code> or <code>--multiple</code>	Adds a wildcard at the end of input and output, so it would mean if <code>-i statistics.log -m</code> was specified, <code>statistics.log*</code> would be imported. (i.e. all of the following: <i>statistics.log</i> , <i>statistics.log.1</i> , <i>statistics.log.2</i> , <i>statistics.log.3</i> , etc.)
<code>-s</code> or <code>--skipdns</code>	To skip the reverse DNS lookups that work out where a user is from. (The DNS lookup finds the information about the host from its IP address, such as geographical location, etc. This can be slow, and wouldn't work on a server not connected to the internet.)
<code>-v</code> or <code>--verbose</code>	Display verbose output (helpful for debugging)
<code>-h</code> or <code>--help</code>	Help

An example form of this command would be `[dspace]/bin/dspace stats-log-importer-elasticsearch -i statistics.log -m`

Viewing Data in Elasticsearch Statistics

In XMLUI, while logged in as an administrator, the Context Panel will have an additional "View Statistics" link when you browse to a Community, Collection, or Item.

The Statistics Report includes:

- Bitstreams with Most Downloads, for all time.
- Bitstreams with Most Downloads, previous month.
- Total Number of Downloads to Bitstreams within this container, broken down by month.
- Number of hits per Country

This data is presented as either a Table or Line Graph, and requires JavaScript to draw the graphics.

4.6.3 SOLR Statistics

DSpace 1.6 and newer versions uses the Apache SOLR application underlying the statistics. SOLR enables performant searching and adding to vast amounts of (usage) data.

Unlike previous versions, enabling statistics in DSpace does not require additional installation or customization. All the necessary software is included.

- 1 [What is exactly being logged ?](#)
 - 1.1 [Common stored fields for all usage events](#)
 - 1.2 [Unique stored fields for bitstream downloads](#)
 - 1.3 [Unique stored fields for search queries](#)
 - 1.4 [Unique stored fields for workflow events](#)
- 2 [Web User Interface Elements](#)
 - 2.1 [Pageview and Download statistics](#)
 - 2.1.1 [Home page](#)
 - 2.1.2 [Community home page](#)
 - 2.1.3 [Collection home page](#)
 - 2.1.4 [Item home page](#)
 - 2.2 [Search Query Statistics](#)
 - 2.3 [Workflow Event Statistics](#)
- 3 [Architecture](#)
- 4 [Configuration settings for Statistics](#)
 - 4.1 [Pre-1.6 Statistics settings](#)
- 5 [Upgrade Process for Statistics](#)
- 6 [Statistics Administration](#)
 - 6.1 [Converting older DSpace logs into SOLR usage data](#)
 - 6.2 [Statistics Client Utility](#)
- 7 [Statistics differences between DSpace 1.7.x and 1.8.0](#)
 - 7.1 [Displayed file statistics bundle configurable](#)
- 8 [Statistics differences between DSpace 1.6.x and 1.7.0](#)
 - 8.1 [SOLR optimization added](#)
 - 8.2 [SOLR Autocommit](#)
- 9 [Web UI Statistics Modification \(XMLUI Only\)](#)
 - 9.1 [Modifying the number of months, for which statistics are displayed](#)
- 10 [Custom Reporting - Querying SOLR Directly](#)
 - 10.1 [Resources](#)
 - 10.2 [Examples](#)
 - 10.2.1 [Top downloaded items by a specific user](#)
- 11 [Manually Installing/Updating GeoLite Database File](#)

What is exactly being logged ?

DSpace 1.6 and newer

After the introduction of the SOLR Statistics logging in DSpace 1.6, every pageview and file download is logged in a dedicated SOLR statistics core.

DSpace 3.0 and newer

In addition to the already existing logging of pageviews and downloads, DSpace 3.0 now also logs search queries users enter in the DSpace search dialog and workflow events.

JSP UI Search Query logging

Due to the very recent addition of Discovery for search & faceted browsing in JSPUI, these search queries are **not** yet logged. Regular (non-discovery) search queries **are** being logged in JSP UI.

Workflow Events logging

Only workflow events, initiated and executed by a physical user are being logged. Automated workflow steps or ingest procedures are currently **not** being logged by the workflow events logger.

The logging happens at the server side, and doesn't require a javascript like Google Analytics does, to provide usage data. Definition of which fields are to be stored happens in the file **dspace/solr/statistics/conf/schema.xml**.

Although they are stored in the same index, the stored fields for views, search queries and workflow events are different. A new field, `statistics_type` determines which kind of a usage event you are dealing with. The three possible values for this field are **view**, **search** and **workflow**.

```
<field name="statistics_type" type="string" indexed="true" stored="true" required="true" />
```

Common stored fields for all usage events

```
<field name="type" type="integer" indexed="true" stored="true" required="true" />
<field name="id" type="integer" indexed="true" stored="true" required="true" />
<field name="ip" type="string" indexed="true" stored="true" required="false" />
<field name="time" type="date" indexed="true" stored="true" required="true" />
<field name="epersonid" type="integer" indexed="true" stored="true" required="false" />
<field name="continent" type="string" indexed="true" stored="true" required="false"/>
<field name="country" type="string" indexed="true" stored="true" required="false"/>
<field name="countryCode" type="string" indexed="true" stored="true" required="false"/>
<field name="city" type="string" indexed="true" stored="true" required="false"/>
<field name="longitude" type="float" indexed="true" stored="true" required="false"/>
<field name="latitude" type="float" indexed="true" stored="true" required="false"/>
<field name="owningComm" type="integer" indexed="true" stored="true" required="false" multiValued="true"/>
```

```

<field name="owningColl" type="integer" indexed="true" stored="true" required="false" multiValued="
true"/>
<field name="owningItem" type="integer" indexed="true" stored="true" required="false" multiValued="
true"/>
<field name="dns" type="string" indexed="true" stored="true" required="false"/>
<field name="userAgent" type="string" indexed="true" stored="true" required="false"/>
<field name="isBot" type="boolean" indexed="true" stored="true" required="false"/>
<field name="referrer" type="string" indexed="true" stored="true" required="false"/>
<field name="uid" type="uuid" indexed="true" stored="true" default="NEW" />
<field name="statistics_type" type="string" indexed="true" stored="true" required="true" default="
view" />

```

The combination of [type](#) and [id](#) determines which resource (either community, collection, item page or file download) has been requested.

Unique stored fields for bitstream downloads

```

<field name="bundleName" type="string" indexed="true" stored="true" required="false" multiValued="
true" />

```

Unique stored fields for search queries

```

<field name="query" type="string" indexed="true" stored="true" required="false" multiValued="true"/
>
<field name="scopeType" type="integer" indexed="true" stored="true" required="false" />
<field name="scopeId" type="integer" indexed="true" stored="true" required="false" />
<field name="rpp" type="integer" indexed="true" stored="true" required="false" />
<field name="sortBy" type="string" indexed="true" stored="true" required="false" />
<field name="sortOrder" type="string" indexed="true" stored="true" required="false" />
<field name="page" type="integer" indexed="true" stored="true" required="false" />

```

Unique stored fields for workflow events

```

<field name="workflowStep" type="string" indexed="true" stored="true" required="false" multiValued=
"true"/>
<field name="previousWorkflowStep" type="string" indexed="true" stored="true" required="false"
multiValued="true"/>
<field name="owner" type="string" indexed="true" stored="true" required="false" multiValued="true"/
>
<field name="submitter" type="integer" indexed="true" stored="true" required="false" />
<field name="actor" type="integer" indexed="true" stored="true" required="false" />
<field name="workflowItemId" type="integer" indexed="true" stored="true" required="false" />

```


Web User Interface Elements

Pageview and Download statistics

In the XMLUI, pageview and download statistics can be accessed from the lower end of the navigation menu. In the JSPUI, a view statistics button appears on the bottom of pages for which statistics are available.

If you are not seeing these links or buttons, it's likely that they are only enabled for administrators in your installation. Change the configuration parameter "authorization.admin.usage" in usage-statistics.cfg to false in order to make statistics visible for all repository visitors.

Home page

Starting from the repository homepage, the statistics page displays the top 10 most popular items of the entire repository.

Community home page

The following statistics are available for the community home pages:

- Total visits of the current community home page
- Visits of the community home page over a timespan of the last 7 months
- Top 10 country from where the visits originate
- Top 10 cities from where the visits originate

Collection home page

The following statistics are available for the collection home pages:

- Total visits of the current collection home page
- Visits of the collection home over a timespan of the last 7 months
- Top 10 country from where the visits originate
- Top 10 cities from where the visits originate

Item home page

The following statistics are available for the item home pages:

- Total visits of the item
- Total visits for the bitstreams attached to the item
- Visits of the item over a timespan of the last 7 months
- Top 10 country views from where the visits originate
- Top 10 cities from where the visits originate

Search Query Statistics

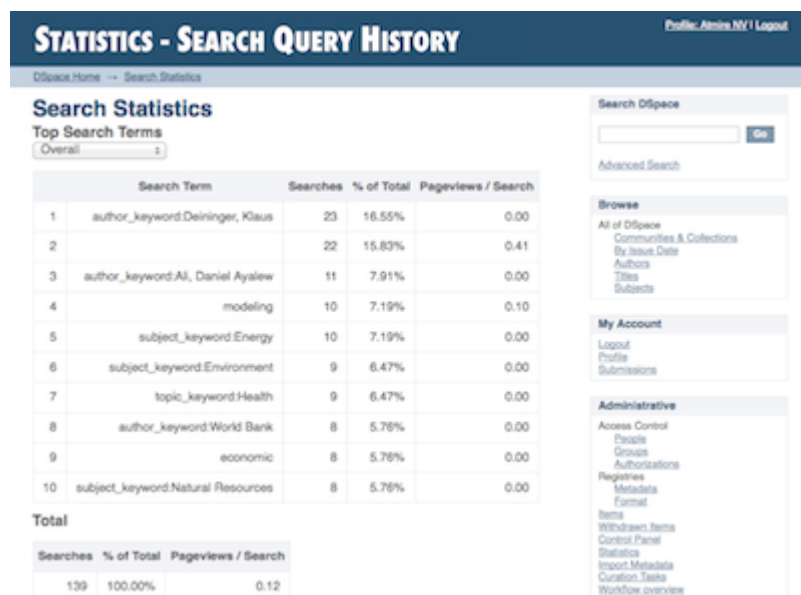
In the XMLUI, search query statistics can be accessed from the lower end of the navigation menu.

If you are not seeing the link labelled "search statistics", it is likely that they are only enabled for administrators in your installation. Change the configuration parameter "authorization.admin.search" in usage-statistics.cfg to false in order to make statistics visible for all repository visitors.

The dropdown on top of the page allows you to modify the time frame for the displayed statistics.

The Pageviews/Search column tracks the amount of pages visited after a particular search term. Therefore a zero in this column means that after executing a search for a specific keyword, not a single user has clicked a single result in the list.

If you are using Discovery, note that clicking the [facets](#) also counts as a search, because clicking a [facet](#) sends a search query to the Discovery index.

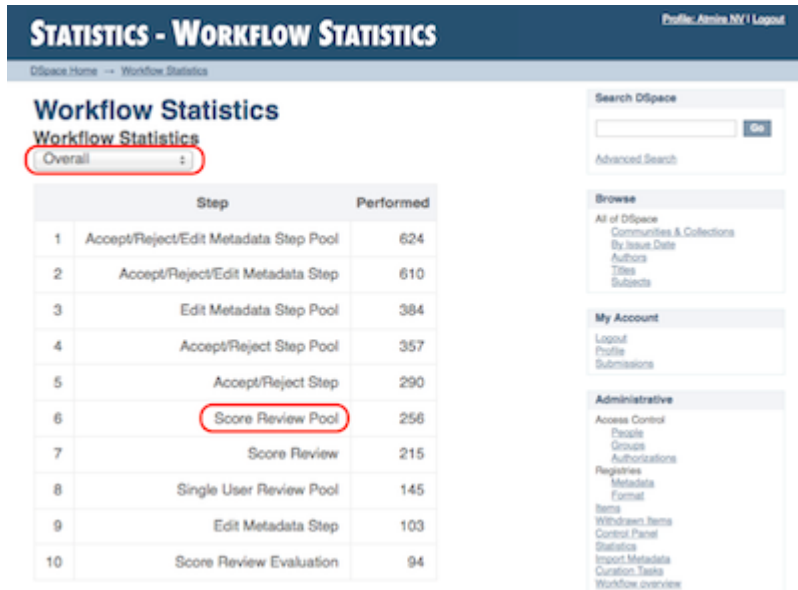


Workflow Event Statistics

In the XMLUI, search query statistics can be accessed from the lower end of the navigation menu.

If you are not seeing the link labelled "Workflow statistics", it is likely that they are only enabled for administrators in your installation. Change the configuration parameter "authorization.admin.workflow" in usage-statistics.cfg to false in order to make statistics visible for all repository visitors.

The dropdown on top of the page allows you to modify the time frame for the displayed statistics.



Architecture

The DSpace Statistics Implementation is a Client/Server architecture based on Solr for collecting usage events in the JSPUI and XMLUI user interface applications of DSpace. Solr runs as a separate webapplication and an instance of Apache Http Client is utilized to allow parallel requests to log statistics events into this Solr instance.

Configuration settings for Statistics

In the `{dspace.dir}/config/modules/solr-statistics.cfg` file review the following fields to make sure they are uncommented:

Property:	server
Example Values:	server = http://127.0.0.1/solr/statistics server = <code>\${solr.server}/statistics</code>
Informational Note:	<p>Is used by the SolrLogger Client class to connect to the Solr server over http and perform updates and queries. In most cases, this can (and should) be set to localhost (or 127.0.0.1).</p> <p>To determine the correct path, you can use a tool like <code>wget</code> to see where Solr is responding on your server. For example, you'd want to send a query to Solr like the following:</p> <pre>wget http://127.0.0.1/solr/statistics/select?q=*:*</pre> <p>Assuming you get an HTTP 200 OK response, then you should set <code>solr.log.server</code> to the <code>' /statistics'</code> URL of <code>'http://127.0.0.1/solr/statistics'</code> (essentially removing the <code>"/select?q=:"</code> query off the end of the responding URL.)</p>

Property:	query.filter.bundles
Example Value:	query.filter.bundles=ORIGINAL
Informational Note:	A comma seperated list that contains the bundles for which the file statistics will be displayed.
Property:	solr.statistics.query.filter.spiderIp
Example Value:	solr.statistics.query.filter.spiderIp = false
Informational Note:	If true, statistics queries will filter out spider IPs -- use with caution, as this often results in extremely long query strings.
Property:	solr.statistics.query.filter.isBot
Example Value:	solr.statistics.query.filter.isBot = true
Informational Note:	If true, statistics queries will filter out events flagged with the "isBot" field. This is the recommended method of filtering spiders from statistics.
Property:	spiderips.urls
Example Value:	spiderips.urls = <pre style="border: 1px dashed gray; padding: 10px;"> http://iplists.com/google.txt, \ http://iplists.com/inktomi.txt, \ http://iplists.com/lycos.txt, \ http://iplists.com/infoseek.txt, \ http://iplists.com/altavista.txt, \ http://iplists.com/excite.txt, \ http://iplists.com/misc.txt, \ http://iplists.com/non_engines.txt </pre>
Informational Note:	<p>List of URLs to download spiders files into [dspace]/config/spiders. These files contain lists of known spider IPs and are utilized by the SolrLogger to flag usage events with an "isBot" field, or ignore them entirely.</p> <p>The "stats-util" command can be used to force an update of spider files, regenerate "isBot" fields on indexed events, and delete spiders from the index. For usage, run:</p>

	<pre>dspace stats-util -h</pre>
	<p>from your [dspace]/bin directory</p>

In the {dspace.dir}/config/modules/**usage-statistics**.cfg file review the following fields to make sure they are uncommented:

Property:	dbfile
Example Value:	dbfile = \${dspace.dir}/config/GeoLiteCity.dat
Informational Note:	The following refers to the GeoLiteCity database file utilized by the LocationUtils to calculate the location of client requests based on IP address. During the Ant build process (both fresh_install and update) this file will be downloaded from http://www.maxmind.com/app/geolitecity if a new version has been published or it is absent from your [dspace]/config directory.
Property:	resolver.timeout
Example Value:	resolver.timeout = 200
Informational Note:	Timeout in milliseconds for DNS resolution of origin hosts/IPs. Setting this value too high may result in solr exhausting your connection pool.
Property:	useProxies
Example Value:	useProxies = true
Informational Note:	Will cause Statistics logging to look for X-Forward URI to detect clients IP that have accessed it through a Proxy service (e.g. the Apache mod_proxy). Allows detection of client IP when accessing DSpace. [Note: This setting is found in the DSpace Logging section of dspace.cfg]
Property:	authorization.admin.usage
Example Value:	authorization.admin.usage = true
Informational Note:	When set to true, only general administrators, collection and community administrators are able to access the pageview and download statistics from the web user interface. As a result, the links to access statistics are hidden for non logged-in admin users. Setting this property to "false" will display the links to access statistics to anyone, making them publicly available.

Property:	authorization.admin.search
Example Value:	authorization.admin.search = true
Informational Note:	When set to true, only system, collection or community administrators are able to access statistics on search queries.
Property:	authorization.admin.workflow
Example Value:	authorization.admin.workflow = true
Informational Note:	When set to true, only system, collection or community administrators are able to access statistics on workflow events.
Property:	logBots
Example Value:	logBots = true
Informational Note:	When this property is set to false, and IP is detected as a spider, the event is not logged. When this property is set to true, the event will be logged with the "isBot" field set to true. (see solr.statistics.query.filter.* for query filter options)

Pre-1.6 Statistics settings

Older versions of DSpace featured static reports generated from the log files. They still persist in DSpace today but are completely independent from the SOLR based statistics.

The following configuration parameters applicable to these reports can be found in dspace.cfg.

```
##### Statistical Report Configuration Settings #####
# should the stats be publicly available? should be set to false if you only
# want administrators to access the stats, or you do not intend to generate
# any
report.public = false
# directory where live reports are stored
report.dir = ${dspace.dir}/reports/
```

These fields are not used by the new 1.6 Statistics, but are only related to the Statistics from previous DSpace releases

Upgrade Process for Statistics

Example of rebuild and redeploy DSpace (only if you have configured your distribution in this manner)

First approach the traditional DSpace build process for updating

```
cd [dspace-source]/dspace
mvn package
cd [dspace-source]/dspace/target/dspace-installer
ant -Dconfig=[dspace]/config/dspace.cfg update
cp -R [dspace]/webapps/* [TOMCAT]/webapps
```

The last step is only used if you do not follow the recommended practice of configuring `[dspace]/webapps` as location for webapps in your servlet container (Tomcat, Resin or Jetty). If you only need to build the statistics, and don't make any changes to other web applications, you can replace the copy step above with:

```
cp -R dspace/webapps/solr TOMCAT/webapps
```

Again, only if you are not mounting `[dspace]/webapps` directly into your Tomcat, Resin or Jetty host (the recommended practice)

Restart your webapps (Tomcat/Jetty/Resin)

Statistics Administration

Converting older DSpace logs into SOLR usage data

If you have upgraded from a previous version of DSpace, converting older log files ensures that you carry over older usage stats from before the upgrade.

Statistics Client Utility

The command line interface (CLI) scripts can be used to clean the usage database from additional spider traffic and other maintenance tasks. In DSpace 3.0, a script has been added to split up the monolithic SOLR core into individual cores each containing a year of statistics.

Statistics differences between DSpace 1.7.x and 1.8.0

Displayed file statistics bundle configurable

In DSpace 1.6.x & 1.7.x the file download statistics were generated without regard to the bundle in which the file was located. In DSpace 1.8.0 it is possible to configure the bundles for which the file statistics are to be shown by using the **query.filter.bundles** property. If required the old file statistics can also be upgraded to include the bundle name so that the old file statistics are fixed.



Backup Your statistics data first

Applying this change will involve dumping all the old file statistics into a file and re uploading these. Therefore it is wise to create a backup of the {dspace.dir}/solr/statistics/data directory. It is best to create this backup when the Tomcat/Jetty/Resin server program isn't running.

When a backup has been made start the Tomcat/Jetty/Resin server program.

The update script has one optional command which will if given not only update the broken file statistics but also delete file statistics for files that were removed from the system (if this option isn't active these statistics will receive the "BITSTREAM_DELETED" bundle name).

```
#The -r is optional  
[dspace]/bin/dspace stats-util -b -r
```

Statistics differences between DSpace 1.6.x and 1.7.0

SOLR optimization added

If required, the solr server can be optimized by running

```
{dspace.dir}/bin/stats-util -o
```

More information on how these solr server optimizations work can be found here: http://wiki.apache.org/solr/SolrPerformanceFactors#Optimization_Considerations.

SOLR Autocommit

In DSpace 1.6.x, each solr event was committed to the solr server individually. For high load DSpace installations, this would result in a huge load of small solr commits resulting in a very high load on the solr server

This has been resolved in dspace 1.7 by only committing usage events to the solr server every 15 minutes. This will result in a delay of the storage of a usage event of maximum 15 minutes. If required, this value can be altered by changing the maxTime property in the

```
{dspace.dir}/solr/statistics/conf/solrconfig.xml
```

Web UI Statistics Modification (XMLUI Only)

Modifying the number of months, for which statistics are displayed

Modify line 205 in the StatisticsTransformer.java file

https://github.com/DSpace/DSpace/blob/dspace-3_x/dspace-xmlui/src/main/java/org/dspace/app/xmlui/aspect/statistics/StatisticsTransformer.java#L205

-6 is the default setting, displaying the past 6 months of statistics. When reducing this to a smaller natural number, less months are being displayed.

Related: [DatasetTimeGenerator Javadoc](#)

Custom Reporting - Querying SOLR Directly

When the web user interface does not offer you the statistics you need, you can greatly expand the reports by querying the SOLR index directly.

Resources

- <http://www.lucidimagination.com/Community/Hear-from-the-Experts/Articles/Faceted-Search-Solr>
- <http://my.safaribooksonline.com/9781847195883/Cover>

Examples

Top downloaded items by a specific user

Query:

```
http://localhost:8080/solr/statistics/select?indent=on&version=2.2&start=0&rows=10&fl=*%2Cscore&qt=standard&wt=standard&explainOther=&hl.fl=&facet=true&facet.field=epersonid&q=type:0
```

Explained:

facet.field=epersonid — You want to group by epersonid, which is the user id.

type:0 — Interested in bitstreams only

```
<lst name="facet_counts">
  <lst name="facet_fields">
    <lst name="epersonid">
      <int name="66">1167</int>
      <int name="117">251</int>
      <int name="52">42</int>
      <int name="19">36</int>
      <int name="88">20</int>
      <int name="112">18</int>
      <int name="110">9</int>
      <int name="96">0</int>
    </lst>
  </lst>
</lst>
```

Manually Installing/Updating GeoLite Database File

The GeoLite Database file (at [dSPACE]/config/GeoLiteCity.dat) is used by the Statistics engine to generate location/country based reports. (*Note: If you are not using DSpace Statistics, this file is not needed.*)

In most cases, this file is installed automatically when you run `ant fresh_install`. However, if the file cannot be downloaded & installed automatically, you may need to manually install it.

As this file is also sometimes updated by MaxMind.com, you may also wish to update it on occasion.

You have three options to install/update this file:

1. Attempt to re-run the automatic installer from your DSpace Source Directory ([dSPACE-source]). This will attempt to automatically download the database file, unzip it and install it into the proper location:

```
ant update_geolite
```

- NOTE: If the location of the GeoLite Database file is known to have changed, you can also run this auto-installer by passing it the new URL of the GeoLite Database File: `ant -Dgeolite=[full-URL-of-geolite] update_geolite`
2. OR, you can manually install the file by performing these steps yourself:
 - First, download the latest GeoLite Database file from <http://geolite.maxmind.com/download/geoip/database/GeoLiteCity.dat.gz>
 - Next, unzip that file to create a file named GeoLiteCity.dat
 - Finally, move or copy that file to your DSpace installation, so that it is located at [dSPACE]/config/GeoLiteCity.dat.
 3. OR, you can combine the two alternatives above, by first downloading the GeoLiteCity.dat.gz file to a location accessible to you, and then configure a `.dSPACE.properties` file in your home folder. For example, create a `.dSPACE.properties` file in the home folder of the user who is running ant to deploy dSPACE, and add the following line to it:

```
.dSPACE.properties
```

```
geolite=file:///path/to/your/downloaded/GeoLiteCity.dat.gz
```

This leaves the original downloading behavior intact, but overrides the URL for the GeoLite Database file from the maxmind.com site to your own location. This typically speeds up the "download" step to about 1 second.

SOLR Statistics Maintenance

- 1 [DSpace Log Converter](#)
- 2 [Filtering and Pruning Spiders](#)

- [3 Backup or Export SOLR records to intermediate format](#)
- [4 Routine Solr Index Maintenance](#)
- [5 Solr Sharding By Year](#)
 - [5.1 Technical implementation details](#)

DSpace Log Converter

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

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

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

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

Command used:	<code>[dspace]/bin/dspace stats-log-importer</code>
Java class:	<code>org.dspace.statistics.util.StatisticsImporter</code>

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

Although the DSpace Log Converter applies basic spider filtering (googlebot, yahoo slurp, msnbot), it is far from complete. Please refer to [Filtering and Pruning Spiders](#) for spider removal operations, after converting your old logs.

Filtering and Pruning Spiders

Command used:	<code>[dspace]/bin/dspace stats-util</code>
Java class:	<code>org.dspace.statistics.util.StatisticsClient</code>
Arguments (short and long forms) :	Description
<code>-b</code> or <code>--reindex-bitstreams</code>	Reindex the bitstreams to ensure we have the bundle name
<code>-r</code> or <code>--remove-deleted-bitstreams</code>	While indexing the bundle names remove the statistics about deleted bitstreams
<code>-u</code> or <code>--update-spider-files</code>	Update Spider IP Files from internet into <code>[dspace]/config/spiders</code> . Downloads Spider files identified in <code>dspace.cfg</code> under property <code>solr.spiderips.urls</code> . See Configuration settings for Statistics
<code>-f</code> or <code>--delete-spiders-by-flag</code>	Delete Spiders in Solr By isBot Flag. Will prune out all records that have <code>isBot:true</code>
<code>-i</code> or <code>--delete-spiders-by-ip</code>	Delete Spiders in Solr By IP Address, DNS name, or Agent name. Will prune out all records that match spider identification patterns.
<code>-m</code> or <code>--mark-spiders</code>	Update isBot Flag in Solr. Marks any records currently stored in statistics that have IP addresses matched in spiders files
<code>-h</code> or <code>--help</code>	Calls up this brief help table at command line.

Notes:

The usage of these options is open for the user to choose. If you want to keep spider entries in your repository, you can just mark them using `-m` and they will be excluded from statistics queries when `"solr.statistics.query.filter.isBot = true"` in the `dspace.cfg`. If you want to keep the spiders out of the solr repository, just use the `-i` option and they will be removed immediately.

Spider IPs are specified in files containing one pattern per line. A line may be a comment (starting with `"#"` in column 1), empty, or a single IP address or DNS name. If a name is given, it will be resolved to an address. Unresolvable names are discarded and will be noted in the log.

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

Spiders may also be excluded by DNS name or Agent header value. Place one or more files of patterns in the directories `[dspace]/config/spiders/domains` and/or `[dspace]/config/spiders/agents`. Each line in a pattern file should be either empty, a comment starting with `"#"` in column 1, or a regular expression which matches some names to be recognized as spiders.

Backup or Export SOLR records to intermediate format

Command used:	<code>[dspace]/bin/dspace stats-util</code>
Java class:	<code>org.dspace.statistics.util.StatisticsClient</code>
Arguments (short and long forms):	Description
<code>-e</code> or <code>--export</code>	Export SOLR view statistics data to usage statistics intermediate format

This exports the records to `dspace / temp / usagestats_0.csv`. This will chunk the files at 10,000 records to new files. This can be imported with `stats-log-importer` to SOLR or `stats-log-importer-elasticsearch` to Elastic Search.

Routine Solr Index Maintenance

Command used:	<code>[dspace]/bin/dspace stats-util</code>
Java class:	<code>org.dspace.statistics.util.StatisticsClient</code>
Arguments (short and long forms):	Description
<code>-o</code> or <code>--optimize</code>	Run maintenance on the SOLR index. Recommended to run daily, to prevent your servlet container from running out of memory

Notes:

The usage of this option is strongly recommended, you should run this script daily (from crontab or your system's scheduler), to prevent your servlet container from running out of memory.

Solr Sharding By Year

Command used:	<code>[dspace]/bin/dspace stats-util</code>
Java class:	<code>org.dspace.statistics.util.StatisticsClient</code>
Arguments (short and long forms):	Description
<code>-s</code> or <code>--shard-solr-index</code>	Splits the data in the main core up into a separate solr core for each year, this will upgrade the performance of the solr.

Notes:

Yearly Solr sharding is a routine that can drastically improve the performance of your DSpace SOLR statistics. It was introduced in DSpace 3.0 and is not backwards compatible. The routine decreases the load created by the logging of new usage events by reducing the size of the SOLR Core in which new usage data are being logged. By running the script, you effectively split your current SOLR core, containing all of your usage events, into different SOLR cores that each contain the data for one year. In case your DSpace has been logging usage events for less than one year, you will see no notable performance improvements until you run the script after the start of a new year. Both writing new usage events as well as read operations should be more performant over several smaller SOLR Shards instead of one monolithic one.

It is highly recommended that you execute this script once at the start of every year. To ensure this is not forgotten, you can include it in your crontab or other system scheduling software. Here's an example cron entry (just replace [dspace] with the full path of your DSpace installation):

```
# At 12:00AM on January 1, "shard" the DSpace Statistics Solr index. Ensures each year has its own
Solr index - this improves performance.
0 0 1 1 * [dspace]/bin/dspace stats-util -s
```

Technical implementation details

After sharding, the SOLR data cores are located in the [dspace.dir]/solr directory. There is no need to define the location of each individual core in solr.xml because they are automatically retrieved at runtime. This retrieval happens in the *static* method located in the *org.dspace.statistics.SolrLogger* class. These cores are stored in the *statisticYearCores* list each time a query is made to the solr these cores are added as shards by the *addAdditionalSolrYearCores* method. The cores share a common configuration copied from your original *statistics* core. Therefore, no issues should be resulting from subsequent ant updates.

The actual sharding of the of the original solr core into individual cores by year is done in the *shardSolrIndex* method in the *org.dspace.statistics.SolrLogger* class. The sharding is done by first running a facet on the time to get the facets split by year. Once we have our years from our logs we query the main solr data server for all information on each year & download these as csv's. When we have all data for one year we upload it to the newly created core of that year by using the [update csv](#) handler. One all data of one year has been uploaded that data is removed from the main solr (by doing it this way if our solr crashes we do not need to start from scratch).

4.7 User Interfaces

4.7.1 Discovery

- [1 What is DSpace Discovery](#)
 - [1.1 What is a Sidebar Facet](#)
 - [1.2 What is a Search Filter](#)
 - [1.3 What is a tag cloud facet](#)

- 2 [Discovery Changelist](#)
 - 2.1 [DSpace 5.0](#)
 - 2.2 [DSpace 4.0](#)
 - 2.3 [DSpace 3.0](#)
 - 2.4 [DSpace 1.8](#)
 - 2.5 [DSpace 1.7](#)
- 3 [Enabling Discovery](#)
 - 3.1 [Removing Legacy Browse Tables \(bi_*\) from your Database](#)
- 4 [Configuration files](#)
- 5 [General Discovery settings \(config/modules/discovery.cfg\)](#)
- 6 [Modifying the Discovery User Interface \(config/spring/api/discovery.xml\)](#)
 - 6.1 [Structure Summary](#)
 - 6.2 [Default settings](#)
 - 6.3 [Non indexed metadata fields](#)
 - 6.4 [Search filters & sidebar facets Customization](#)
 - 6.4.1 [Hierarchical \(taxonomies based\) sidebar facets](#)
 - 6.5 [Sort option customization for search results](#)
 - 6.6 [DiscoveryConfiguration](#)
 - 6.6.1 [Configuring lists of sidebarFacets and searchFilters](#)
 - 6.6.2 [Configuring and customizing search sort fields](#)
 - 6.6.3 [Adding default filter queries \(OPTIONAL\)](#)
 - 6.6.4 [Access Rights Awareness](#)
 - 6.6.4.1 [Access Rights Awareness - technical details](#)
 - 6.6.5 [Customizing the Recent Submissions display](#)
 - 6.6.6 [Customizing hit highlighting & search snippets](#)
 - 6.6.6.1 [Hit highlighting technical details](#)
 - 6.6.7 ["More like this" configuration](#)
 - 6.6.7.1 ["More like this" technical details](#)
 - 6.6.8 ["Did you mean" spellcheck aid for search configuration](#)
 - 6.6.8.1 ["Did you mean" spellcheck aid for search technical details](#)
 - 6.6.9 [Customizing the "Tag Cloud" facet](#)
- 7 [Discovery Solr Index Maintenance](#)
 - 7.1 [Routine Discovery Solr Index Maintenance](#)
- 8 [Advanced Solr Configuration](#)

What is DSpace Discovery

The Discovery Module enables faceted searching & browsing for your repository.

Although these techniques are new in DSpace, they might feel familiar from other platforms like Aquabrowser or Amazon, where facets help you to select the right product according to facets like price and brand. DSpace Discovery offers very powerful browse and search configurations that were only possible with code customization in the past.

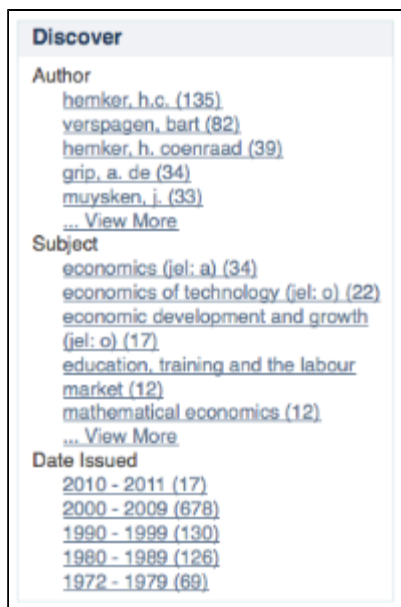
[Watch the DSpace Discovery introduction video](#)

i Since DSpace 4.0 Discovery is the default Search and Browse infrastructure for both XMLUI and JSPUI.

What is a Sidebar Facet

From the user perspective, faceted search (also called faceted navigation, guided navigation, or parametric search) breaks up search results into multiple categories, typically showing counts for each, and allows the user to "drill down" or further restrict their search results based on those facets.

When you have successfully enabled Discovery in your DSpace, you will notice that the different enabled facets are visualized in a "Discover" section in your sidebar, by default, right below the Browse options.



In this example, there are 3 Sidebar Facets, Author, Subject and Date Issued. It's important to know that multiple metadata fields can be included in one facet. For example, the Author facet above includes values from both `dc.contributor.author` as well as `dc.creator`.

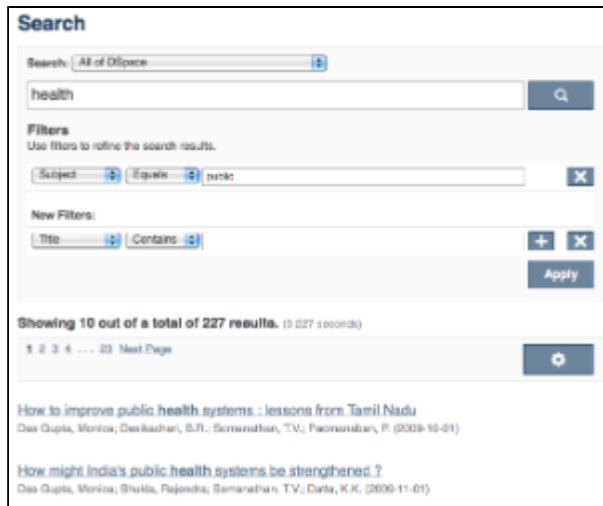
Another important property of Sidebar Facets is that their contents are automatically updated to the context of the page. On collection homepages or community homepages it will include information about the items included in that particular collection or community.

What is a Search Filter

In a standard search operation, a user specifies his complete query prior to launching the operation. If the results are not satisfactory, the user starts over again with a (slightly) altered query.

In a faceted search, a user can modify the list of displayed search results by specifying additional "filters" that will be applied on the list of search results. In DSpace, a filter is a condition applied to specific facets. In

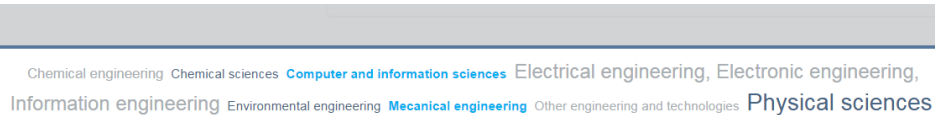
the example below, a user started with the search term "health", which yielded 500 results. After applying the filter "public" on the facet "Subject", only 227 results remain. Each time a user selects a sidebar facet it will be added as a filter. Active filters can be altered or removed in the 'filters' section of the search interface.



Another example: Using the standard search, a user would search for something like **[wetland + "dc.author=Mitsch, William J" + dc.subject="water quality"]**. With filtered search, they can start by searching for **[wetland]**, and then filter the results by the other attributes, author and subject.

What is a tag cloud facet

Tag cloud facet is another way to display facets of your repository in a "tag cloud" form in which the importance of each tag is shown with font size or color. This format is useful for quickly perceiving the most prominent terms.



This is a classic "tag cloud" facet in a DSpace repository.

Discovery Changelist


DSpace 5.0

The new JSPUI-only tag cloud facet feature is disabled by default. In order to enable it, you will need to set up the corresponding processor that the PluginManager will load to actually perform the tag cloud query on the relevant pages. This is configured in the dspace.cfg configuration file using the following properties:

- plugin.sequence.org .dspace.plugin.CommunityHomeProcessor
- plugin.sequence.org .dspace.plugin.CollectionHomeProcessor
- plugin.sequence.org .dspace.plugin.SiteHomeProcessor

The tag cloud has been declared there for you but it is commented out.


DSpace 4.0

 Starting from DSpace 4.0, Discovery is the default search and browse solution for DSpace.

General improvements:

- Browse interfaces now also use Discovery index (rather than the legacy Lucene index)
- "Did you means" spell check aid for search

DSpace 3.0

 Starting from DSpace 3.0, Discovery is also supported in JSPUI.

General improvements:

- Hierarchical facets sidebar facets
- Improved & more intuitive user interface
- [Access Rights Awareness](#) (enabled by default). Access restricted or embargoed content is hidden from anonymous search/browse.
- Authority control & variants awareness (homonyms are shown separately in a facet if they have different authority ID). All variant forms as recognized by the authority framework are indexed. See [Authority Framework](#)

XMLUI-only:

- Hit highlighting and search snippets support
- "More like this" (related items)

Bugfixes and other changes

- Auto-complete functionality has been removed in XMLUI from search queries due to performance issues. JSPUI still supports auto-complete functionality without performance issues.

DSpace 1.8

- Configuration moved from `dspace.cfg` into `config/modules/discovery.cfg` and `config/spring/api/discovery.xml`
- Individual communities and collections can have their own Discovery configuration.
- Tokenization for Auto-complete values (see `SearchFilter`)
- Alphanumeric sorting for Sidebarfacets
- Possibility to avoid indexation of specific metadata fields.

- Grouping of multiple metadata fields under the same SidebarFacet

DSpace 1.7

- Sidebar browse facets that can be configured to use contents from any metadata field
 - Dynamically generated timespans for dates
- Customizable "recent submissions" view on the repository homepage, collection and community pages
- Hit highlighting & search snippets

Enabling Discovery

Because Discovery was adopted as the default infrastructure for search and browse in DSpace 4, no manual steps are required to enable Discovery. If you want to enable Discovery on older versions of DSpace, please refer to the DSpace documentation for that particular version.

Removing Legacy Browse Tables (bi_*) from your Database

If you have upgraded from an older version of DSpace, your database may still include outdated "bi_*" tables (where "bi" = "browse index"). When Discovery is enabled, these tables are no longer necessary, as Discovery takes over this browse index function.

To clean up all these old "bi_*" tables, simply run:

```
[dspace]/bin/dspace index-db-browse -f -d
```

Configuration files

The configuration for discovery is located in 2 separate files.

- **General settings:** The `discovery.cfg` file located in the `[dspace-install-dir]/config/modules` directory.
- **User Interface Configuration:** The `discovery.xml` file is located in `[dspace-install-dir]/config/spring/api/` directory.

General Discovery settings (config/modules/discovery.cfg)

The `discovery.cfg` file is located in the `[dspace-install-dir]/config/modules` directory and contains following properties:

Property:	search.server
Example Value:	<code>search.server=[http://localhost:8080/solr/search]</code>

Informational Note:	Discovery relies on a Solr index for storage and retrieval of its information. This parameter determines the location of the Solr index.
Property:	index.authority.ignore[.field]
Example Value:	<code>index.authority.ignore=true</code> <code>index.authority.ignore.dc.contributor.author=false</code>
Informational Note:	By default, Discovery will use the authority information in the metadata to disambiguate homonyms. Setting this property to false will make the indexing process the same as the metadata doesn't include authority information. The configuration can be different on a field (<schema>.<element>.<qualifier>) basis, the property without field set the default value.
Property:	index.authority.ignore-prefered[.field]
Example Value:	<code>index.authority.ignore-prefered=true</code> <code>index.authority.ignore-prefered.dc.contributor.author=false</code>
Informational Note:	By default, Discovery will use the authority information in the metadata to query the authority for the preferred label. Setting this property to false will make the indexing process the same as the metadata doesn't include authority information (i.e. the preferred form is the one recorded in the metadata value). The configuration can be different on a field (<schema>.<element>.<qualifier>) basis, the property without field set the default value. If the authority is a remote service, disabling this feature can greatly improve performance.
Property:	index.authority.ignore-variants[.field]
Example Value:	<code>index.authority.ignore-variants=true</code> <code>index.authority.ignore-variants.dc.contributor.author=false</code>
Informational Note:	By default, Discovery will use the authority information in the metadata to query the authority for variants. Setting this property to false will make the indexing process the same, as the metadata doesn't include authority information. The configuration can be different on a per-field (<schema>.<element>.<qualifier>) basis, the property without field set the default value. If authority is a remote service, disabling this feature can greatly improve performance.

Modifying the Discovery User Interface (config/spring/api/discovery.xml)

The `discovery.xml` file is located in the `[dspace-install-dir]/config/spring/api` directory.

Structure Summary

This file is in XML format, you should be familiar with XML before editing this file. The configurations are organized together in beans, depending on the purpose these properties are used for.

This purpose can be derived from the class of the beans. Here's a short summary of classes you will encounter throughout the file and what the corresponding properties in the bean are used for.

[Download the configuration file and review it together with the following parameters](#)

Class:	DiscoveryConfigurationService
Purpose :	Defines the mapping between separate Discovery configurations and individual collections/communities
Default:	All communities, collections and the homepage (key=default) are mapped to defaultConfiguration, also controls the metadata fields that should not be indexed in the search core (item provenance for example).
Class:	DiscoveryConfiguration
Purpose :	Groups configurations for sidebar facets, search filters, search sort options and recent submissions
Default:	There is one configuration by default called defaultConfiguration
Class:	DiscoverySearchFilter
Purpose :	Defines that specific metadata fields should be enabled as a search filter
Default:	dc.title, dc.contributor.author, dc.creator, dc.subject.* and dc.date.issued are defined as search filters
Class:	DiscoverySearchFilterFacet
Purpose :	Defines which metadata fields should be offered as a contextual sidebar browse options, each of these facets has also got to be a search filter
Default:	dc.contributor.author, dc.creator, dc.subject.* and dc.date.issued
Class:	HierarchicalSidebarFacetConfiguration
Purpose :	Defines which metadata fields contain hierarchical data and should be offered as a contextual sidebar option
Class:	DiscoverySortConfiguration
Purpose :	Further specifies the sort options to which a DiscoveryConfiguration refers
Default:	dc.title and dc.date.issued are defined as alternatives for sorting, other than Relevance (hard-coded)

Class:	DiscoveryHitHighlightingConfiguration
Purpose:	Defines which metadata fields can contain hit highlighting & search snippets
Default:	dc.title, dc.contributor.author, dc.subject, dc.description.abstract & full text from text files.
Class:	TagCloudFacetConfiguration
Purpose:	Defines the tag cloud appearance configuration bean and the search filter facets to appear in the tag cloud form. You can have different " TagCloudFacetConfiguration " per community or collection or the home page

Default settings

In addition to the summarized descriptions of the default values, following details help you to better understand these defaults. If you haven't already done so, [download the configuration file and review it together with the following parameters](#).

The file contains one default configuration that defines following sidebar facets, search filters, sort fields and recent submissions display:

- Sidebar facets
 - **searchFilterAuthor**: groups the metadata fields dc.contributor.author & dc.creator with a facet limit of 10, sorted by occurrence count
 - **searchFilterSubject**: groups all subject metadata fields (dc.subject.*) with a facet limit of 10, sorted by occurrence count
 - **searchFilterIssued**: contains the dc.date.issued metadata field, which is identified with the type "date" and sorted by specific date values
- Search filters
 - **searchFilterTitle**: contains the dc.title metadata field
 - **searchFilterAuthor**: contains the dc.contributor.author & dc.creator metadata fields
 - **searchFilterSubject**: contains the dc.subject.* metadata fields
 - **searchFilterIssued**: contains the dc.date.issued metadata field with the type "date"
- Sort fields
 - **sortTitle**: contains the dc.title metadata field
 - **sortDateIssued**: contains the dc.date.issued metadata field, this sort has the type date configured
- defaultFilterQueries
 - The default configuration contains no defaultFilterQueries
 - The default filter queries are disabled by default but there is an example in the default configuration in comments which allows discovery to only return items (as opposed to also communities/collections).
- Recent Submissions
 - The recent submissions are sorted by dc.date.accessioned which is a date and a maximum number of 5 recent submissions are displayed.

- Hit highlighting
 - The fields dc.title, dc.contributor.author & dc.subject can contain hit highlighting.
 - The dc.description.abstract & full text field are used to render search snippets.
- Non indexed metadata fields
 - **Community/Collections:** dc.rights (copyright text)
 - **Items:** dc.description.provenance

Many of the properties contain lists that use references to point to the configuration elements. This way a certain configuration type can be used in multiple discovery configurations so there is no need to duplicate them.

Non indexed metadata fields

The discovery.xml file has configuration to not index certain metadata fields for communities/collections/items. The configuration is handled in the "toIgnoreMetadataFields" property located in the "org.dspace.discovery.configuration.DiscoveryConfigurationService" bean. Below is an example configuration that excludes dc.description.provenance for items & dc.rights for communities/collections:

```
<property name="toIgnoreMetadataFields">
  <map>
    <entry>
      <key><util:constant static-field="org.dspace.core.Constants.COMMUNITY"/></key>
      <list>
        <!--Introduction text-->
        <!--<value>dc.description</value>-->
        <!--Short description-->
        <!--<value>dc.description.abstract</value>-->
        <!--News-->
        <!--<value>dc.description.tableofcontents</value>-->
        <!--Copyright text-->
        <value>dc.rights</value>
        <!--Community name-->
        <!--<value>dc.title</value>-->
      </list>
    </entry>
    <entry>
      <key><util:constant static-field="org.dspace.core.Constants.COLLECTION"/></key>
      <list>
        <!--Introduction text-->
        <!--<value>dc.description</value>-->
        <!--Short description-->
        <!--<value>dc.description.abstract</value>-->
        <!--News-->
        <!--<value>dc.description.tableofcontents</value>-->
        <!--Copyright text-->
        <value>dc.rights</value>
        <!--Collection name-->
        <!--<value>dc.title</value>-->
      </list>
    </entry>
    <entry>
```



```

    <key><util:constant static-field="org.dspace.core.Constants.ITEM"/></key>
    <list>
      <value>dc.description.provenance</value>
    </list>
  </entry>
</map>
</property>

```

By adding additional values to the appropriate lists additional metadata can be excluded from the search core, a reindex is required after altering this file to ensure that the values are removed from the index.

Search filters & sidebar facets Customization

This section explains the properties for search filters & sidebar facets. Each sidebar facet must occur in the reference list of the search filters. Below is an example configuration of a search filter that is not used as a sidebar facet.

```

<bean id="searchFilterTitle" class="org.dspace.discovery.configuration.DiscoverySearchFilter">
  <property name="indexFieldName" value="title"/>
  <property name="metadataFields">
    <list>
      <value>dc.title</value>
    </list>
  </property>
</bean>

```

The id & class attributes are mandatory for this type of bean. The properties that it contains are discussed below

- **indexFieldName** (Required): A unique search filter name, the metadata will be indexed in Solr under this field name.
- **metadataFields** (Required): A list of the metadata fields that need to be included in the facet.

Sidebar facets extend the search filter and add some extra properties to it, below is an example of a search filter that is also used as a sidebar facet.

```

<bean id="searchFilterAuthor" class="org.dspace.discovery.configuration.SidebarFacetConfiguration">
  <property name="indexFieldName" value="author"/>
  <property name="metadataFields">
    <list>
      <value>dc.contributor.author</value>
      <value>dc.creator</value>
    </list>
  </property>
  <property name="facetLimit" value="10"/>
  <property name="sortOrder" value="COUNT"/>
  <property name="type" value="text"/>
</bean>

```

Note that the class has changed from **DiscoverySearchFilter** to **SidebarFacetConfiguration** this is needed to support the extra properties.

- **facetLimit** (optional): The maximum number of values to be shown. This property is optional, if none is specified the default value "10" will be used. If the filter has the type **date**, this property will not be used since dates are automatically grouped together.
- **sortOrder** (optional): The sort order for the sidebar facets, it can either be COUNT or VALUE. The default value is COUNT.
 - **COUNT** Facets will be sorted by the amount of times they appear in the repository
 - **VALUE** Facets will be sorted alphabetically
- **type**(optional): the type of the sidebar facet it can either be "date" or "text", "text" is the default value.
 - **text**: The facets will be treated as is
 - **date**: Only the year will be stored in the Solr index. These years are automatically displayed in ranges that get smaller when you select one.

Hierarchical (taxonomies based) sidebar facets

Discovery supports specialized drill down in hierarchically structured metadata fields. For this drill down to work, the metadata in the field for which you enable this must be composed out of terms, divided by a splitter. For example, you could have a dc.subject.taxonomy field in which you keep metadata like "CARTOGRAPHY::PHOTOGRAMMETRY", in which Cartography and Photogrammetry are both terms, divided by the splitter "::". The sidebar will only display the top level facets, when clicking on view more all the facet options will be displayed.

```
<bean id="searchFilterSubject" class="
org.dspace.discovery.configuration.HierarchicalSidebarFacetConfiguration">
  <property name="indexFieldName" value="subject" />
  <property name="metadataFields">
    <list>
      <value>dc.subject</value>
    </list>
  </property>
  <property name="sortOrder" value="COUNT" />
  <property name="splitter" value="::" />
  <property name="skipFirstNodeLevel" value="false" />
</bean>
```

Note that the class has changed from **SidebarFacetConfiguration** to **HierarchicalSidebarFacetConfiguration** this is needed to support the extra properties.

- **splitter** (required): The splitter used to split up the separate nodes
- **skipFirstNodeLevel** (optional): Whether or not to show the root node level. For some hierarchical data there is a single root node. In most cases it doesn't need to be shown since it isn't relevant. **This property is true by default.**

Sort option customization for search results

This section explains the properties of an individual SortConfiguration, like sortTitle and sortDateIssued from the default configuration. In order to create custom sort options, you can either modify specific properties of those that already exist or create a totally new one from scratch.

Here's what the sortTitle SortConfiguration looks like:

```
<bean id="sortTitle" class="org.dspace.discovery.configuration.DiscoverySortFieldConfiguration">
  <property name="metadataField" value="dc.title"/>
  <property name="type" value="text"/>
</bean>
```

The id & class attributes are mandatory for this type of bean. The properties that it contains are discussed below

- **metadataField** (Required): The metadata field indicating the sort values
- **type** (optional): the type of the sort option can either be date or text, if none is defined text will be used.

DiscoveryConfiguration

The DiscoveryConfiguration Groups configurations for sidebar facets, search filters, search sort options and recent submissions. If you want to show the same sidebar facets, use the same search filters, search options and recent submissions everywhere in your repository, you will only need one DiscoveryConfiguration and you might as well just edit the defaultConfiguration.

The DiscoveryConfiguration makes it very easy to use custom sidebar facets, search filters, ... on specific communities or collection homepage. This is particularly useful if your collections are heterogeneous. For example, in a collection with conference papers, you might want to offer a sidebar facet for conference date, which might be more relevant than the actual issued date of the proceedings. In a collection with papers, you might want to offer a facet for funding bodies or publisher, while these fields are irrelevant for items like learning objects.

A DiscoveryConfiguration consists out of five parts

- The list of applicable sidebarFacets
- The list of applicable searchFilters
- The list of applicable searchSortFields
- Any default filter queries (optional)
- The configuration for the Recent submissions display
- The configuration of the tag cloud facet

Configuring lists of sidebarFacets and searchFilters



After modifying sidebarFacets and searchFilters, don't forget to reindex existing items by running [dspace]/bin/dspace index-discovery -b, otherwise the changes will not appear.

Below is an example of how one of these lists can be configured. It's important that each of the bean references corresponds to the exact name of the earlier defined facets, filters or sort options.

Each sidebar facet must also occur in the list of the search filters.

```
<property name="sidebarFacets">
  <list>
    <ref bean="sidebarFacetAuthor" />
    <ref bean="sidebarFacetSubject" />
    <ref bean="sidebarFacetDateIssued" />
  </list>
</property>
```

Configuring and customizing search sort fields

The search sort field configuration block contains the available sort fields and the possibility to configure a default sort field and sort order.

Below is an example of the sort configuration.

```
<property name="searchSortConfiguration">
  <bean class="org.dspace.discovery.configuration.DiscoverySortConfiguration">
    <!--<property name="defaultSort" ref="sortDateIssued"/>-->
    <!--DefaultSortOrder can either be desc or asc (desc is default)-->
    <property name="defaultSortOrder" value="desc"/>
    <property name="sortFields">
      <list>
        <ref bean="sortTitle" />
        <ref bean="sortDateIssued" />
      </list>
    </property>
  </bean>
</property>
```

The property name & the bean class are mandatory. The property field names are discussed below.

- **defaultSort** (optional): The default field on which the search results will be sorted, this must be a reference to an existing search sort field bean. If none is given relevance will be the default. Sorting according to the internal relevance algorithm is always available, even though it's not explicitly mentioned in the sortFields section.
- **defaultSortOrder** (optional): The default sort order can either be asc or desc.

- **sortFields** (mandatory): The list of available sort options, each element in this list must link to an existing sort field configuration bean.

Adding default filter queries (OPTIONAL)

Default filter queries are applied on all search operations & sidebar facet clicks. One useful application of default filter queries is ensuring that all returned results are items. As a result, subcommunities and collections that are returned as results of the search operation, are filtered out.

Similar to the lists above, the default filter queries are defined as a list. They are optional.

```
<property name="defaultFilterQueries">
  <list>
    <value>query1</value>
    <value>query2</value>
  </list>
</property>
```

This property contains a simple list which in turn contains the queries. Some examples of possible queries:

- search.resourcetype:2
- dc.subject:test
- dc.contributor.author: "Van de Velde, Kevin"
- ...

Access Rights Awareness

By default, when searching and browsing using Discovery, you will only see items that you have access to. So, your search/browse results may differ if you are logged into DSpace. This Access Rights Awareness feature ensures that anonymous users (and search engines) are not able to access information (both files and metadata) about embargoed or private items. It also provides you with more direct control over who can see individual items within your DSpace.

How does Access Rights Awareness work?

Access Rights Awareness checks the "READ" access on the Item.

If the "Anonymous" group has "READ" access on the Item, then anonymous/public users will be able to view that Item's metadata and locate that Item via DSpace's search/browse system. In addition, search engines will also be able to index that Item's metadata. However, even with Anonymous READ set at the Item-level, you may still choose to access-restrict the downloading/viewing of *files* within the Item. To do so, you would restrict "READ" access on individual Bitstream(s) attached to the Item.

If the "Anonymous" group does NOT have "READ" access on the Item, then anonymous users will never see that Item appear within their search/browse results (essentially the Item is "invisible" to them). In addition, that Item will be invisible to search engines, so it will never be indexed by them. However, any users who have been

given READ access will be able to find/locate the item after logging into DSpace. For example, if a "Staff" group was provided "READ" access on the Item, then members of that "Staff" group would be able to locate the item via search/browse after logging into DSpace.

How can I disable Access Rights Awareness?

If you prefer to allow all access-restricted or embargoed Items to be findable within your DSpace, you can choose to turn off Access Rights Awareness. However, please be aware that this means that restricting "READ" access on an Item will not really do anything – the Item metadata will be available to the public no matter what group(s) were given READ access on that Item.

This feature can be switched off by going to the `[dspace.dir]/config/spring/api/discovery.xml` file & commenting out the bean & the alias shown below.

```
<bean class="org.dspace.discovery.SolrServiceResourceRestrictionPlugin" id="
solrServiceResourceIndexPlugin"/>
<alias name="solrServiceResourceIndexPlugin" alias="
org.dspace.discovery.SolrServiceResourceRestrictionPlugin"/>
```



The Browse Engine only supports the "Access Rights Awareness" if the Solr/Discovery backend is enabled (see [Defining the Storage of the Browse Data](#)). However, it is enabled by default for DSpace 3.x and above.

Access Rights Awareness - technical details

The *DSpaceObject* class has an *updateLastModified()* method which will be triggered each time an authorization policy changes. This method is only implemented in the item class where the last_modified timestamp will be updated and a modify event will be fired. By doing this we ensure that the discovery consumer is called and the item is reindexed. Since this feature can be switched off a separate plugin has been created: the *SolrServiceResourceRestrictionPlugin*. Whenever we reindex a DSpace object all the read rights will be stored in the read field. We make a distinction between groups and users by adding a 'g' prefix for groups and the 'e' prefix for epersons.

When searching in discovery all the groups the user belongs to will be added as a filter query as well as the users identifier. If the user is an admin all items will be returned since an admin has read rights on everything.

Customizing the Recent Submissions display



This paragraph only applies to XMLUI. JSPUI relies on the Browse Engine to show "recent submissions". This requires that the Solr/Discovery backend is enabled (see [Defining the Storage of the Browse Data](#)).

The recent submissions configuration element contains all the configuration settings to display the list of recently submitted items on the home page or community/collection page. Because the recent submission configuration is in the discovery configuration block, it is possible to show 10 recently submitted items on the home page but 5 on the community/collection pages.

Below is an example configuration of the recent submissions.

```
<property name="recentSubmissionConfiguration">
  <bean class="org.dspace.discovery.configuration.DiscoveryRecentSubmissionsConfiguration">
    <property name="metadataSortField" value="dc.date.accessioned"/>
    <property name="type" value="date"/>
    <property name="max" value="5"/>
  </bean>
</property>
```

The property name & the bean class are mandatory. The property field names are discussed below.

- **metadataSortField** (mandatory): The metadata field to sort on to retrieve the recent submissions
- **max** (mandatory): The maximum number of results to be displayed as recent submissions
- **type** (optional): the type of the search filter. It can either be date or text, if none is defined text will be used.

Customizing hit highlighting & search snippets

⊖ This paragraph only applies to XMLUI. JSPUI does not currently support "highlighting & search snippets".

The hit highlighting configuration element contains all settings necessary to display search snippets & enable hit highlighting.

⊖ Changes made to the configuration will not automatically be displayed in the user interface. By default, only the following fields are displayed: dc.title, dc.contributor.author, dc.creator, dc.contributor, dc.date.issued, dc.publisher, dc.description.abstract and fulltext.

If additional fields are required, look for the "itemSummaryList" template.

Below is an example configuration of hit highlighting.

```
<property name="hitHighlightingConfiguration">
  <bean class="org.dspace.discovery.configuration.DiscoveryHitHighlightingConfiguration">
    <property name="metadataFields">
      <list>
```

```

    <bean class="
org.dspace.discovery.configuration.DiscoveryHitHighlightFieldConfiguration">
    <property name="field" value="dc.title"/>
    <property name="snippets" value="5"/>
</bean>
    <bean class="
org.dspace.discovery.configuration.DiscoveryHitHighlightFieldConfiguration">
    <property name="field" value="dc.contributor.author"/>
    <property name="snippets" value="5"/>
</bean>
    <bean class="
org.dspace.discovery.configuration.DiscoveryHitHighlightFieldConfiguration">
    <property name="field" value="dc.subject"/>
    <property name="snippets" value="5"/>
</bean>
    <bean class="
org.dspace.discovery.configuration.DiscoveryHitHighlightFieldConfiguration">
    <property name="field" value="dc.description.abstract"/>
    <property name="maxSize" value="250"/>
    <property name="snippets" value="2"/>
</bean>
    <bean class="
org.dspace.discovery.configuration.DiscoveryHitHighlightFieldConfiguration">
    <property name="field" value="fulltext"/>
    <property name="maxSize" value="250"/>
    <property name="snippets" value="2"/>
</bean>
</list>
</property>
</bean>
</property>

```

The property name & the bean class are mandatory. The property field names are:

- **field** (mandatory): The metadata field to be highlighted (can also be * if all the metadata fields should be highlighted).
- **maxSize** (optional): Limit the number of characters displayed to only the relevant part (use metadata field as search snippet).
- **snippets** (optional): The maximum number of snippets that can be found in one metadata field.

Hit highlighting technical details

The *org.dspace.discovery.DiscoveryQuery* object has a setter & getter for the hit highlighting configuration set in Discovery configuration. If this configuration is given the *resolveToSolrQuery* method located in the *org.dspace.discovery.SolrServiceImpl* class will use the standard Solr highlighting feature (<http://wiki.apache.org/solr/HighlightingParameters>). The *org.dspace.discovery.DiscoverResult* class has a method to set the highlighted fields for each object & field.

The rendering of search results is no longer handled by the METS format but uses a special type of list named "TYPE_DSO_LIST". Each metadata field (& fulltext if configured) is added in the DRI and IF the field contains hit

highlighting the Java code will split up the string & add *DRI highlights* to the list. The XSL for the themes also contains special rendering XSL for the DRI; for Mirage, the changes are located in the *discovery.xsl* file. For themes using the old themes based on structural.xml, look for the template matching "*dri:lstf@type='dsolist'*".

"More like this" configuration



This paragraph only apply to XMLUI. The JSPUI does not currently support the "More like this" feature.

The "more like this"-configuration element contains all the settings for displaying related items on an item display page.

Below is an example of the "more like this" configuration.

```
<property name="moreLikeThisConfiguration">
  <bean class="org.dspace.discovery.configuration.DiscoveryMoreLikeThisConfiguration">
    <property name="similarityMetadataFields">
      <list>
        <value>dc.title</value>
        <value>dc.contributor.author</value>
        <value>dc.creator</value>
        <value>dc.subject</value>
      </list>
    </property>
    <!--The minimum number of matching terms across the metadata fields above before an item is
found as related -->
    <property name="minTermFrequency" value="5"/>
    <!--The maximum number of related items displayed-->
    <property name="max" value="3"/>
    <!--The minimum word length below which words will be ignored-->
    <property name="minWordLength" value="5"/>
  </bean>
</property>
```

The property name & the bean class are mandatory. The property field names are discussed below.

- **similarityMetadataFields**: the metadata fields checked for similarity
- **minTermFrequency**: The minimum number of matching terms across the metadata fields above before an item is found as related
- **max**: The maximum number of related items displayed
- **minWordLength**: The minimum word length below which words will be ignored

"More like this" technical details

The *org.dspace.discovery.SearchService* object has received a *getRelatedItems()* method. This method requires an item & the more-like-this configuration bean from above. This method is implemented in the *org.dspace.discovery.SolrServiceImpl* which uses the item as a query & uses the default Solr parameters for more-like-this to pass the bean configuration to solr (<https://cwiki.apache.org/confluence/display/solr/>

[MoreLikeThis](#)). The result will be a list of items or if none found an empty list. The rendering of this list is handled in the `org.dspace.app.xmlui.aspect.discovery.RelatedItems` class.

"Did you mean" spellcheck aid for search configuration

DSpace 4 introduces the use of SOLR's SpellCheckComponent as an aid for search. When a user's search does not return any hits, the user is presented with a suggestion for an alternative search query.



Search: All of DSpace

pannl eror

Did you mean: [panel error](#)

[Add filters](#)

The feature currently only one line of configuration to `discovery.xml`. Changing the value from true to false will disable the feature.

```
<property name="spellCheckEnabled" value="true" />
```

"Did you mean" spellcheck aid for search technical details

Similar to the More like this configuration, SOLR's spell check component is used with default configuration values. Any of these values can be overridden in the `solrconfig.xml` file located in `dspace/solr/search/conf/`. Following links provide more information about the SOLR SpellCheckComponent:

<http://wiki.apache.org/solr/SpellCheckComponent>

<https://cwiki.apache.org/confluence/display/solr/Spell+Checking>

Customizing the "Tag Cloud" facet

This paragraph only applies to JSPUI

```
<!-- Set TagCloud configuration per discovery configuration -->
<property name="tagCloudFacetConfiguration" ref="defaultTagCloudFacetConfiguration"/>
```

*Declare the bean (of class: **TagCloudFacetConfiguration**) that holds the configuration for the tag cloud facet.*

```
<!--TagCloud configuration bean for homepage discovery configuration-->
<bean id="homepageTagCloudFacetConfiguration" class="
org.dspace.discovery.configuration.TagCloudFacetConfiguration">
  <!-- Actual configuration of the tagcloud (colors, sorting, etc.) -->
  <property name="tagCloudConfiguration" ref="tagCloudConfiguration"/>
```

```

<!-- List of tagclouds to appear, one for every search filter, one after the other -->
<property name="tagCloudFacets">
  <list>
    <ref bean="searchFilterSubject" />
  </list>
</property>
</bean>

```

This bean has two properties:

- **tagCloudConfiguration**: is the bean which describes the actual appearance parameters
- **tagCloudFacets**: the search filter facets which will be used for the tag cloud. If you leave the list empty, no tag cloud will appear. If you declare more than one, such number of tag clouds will appear for each search filter, one after the other.

The appearance configuration can have the following properties, as shown in the following bean:

```

<bean id="tagCloudConfiguration" class="org.dspace.discovery.configuration.TagCloudConfiguration">
  <!-- Should display the score of each tag next to it? Default: false -->
  <property name="displayScore" value="true"/>
  <!-- Should display the tag as center aligned in the page or left aligned? Possible
values: true | false. Default: true -->
  <property name="shouldCenter" value="true"/>
  <!-- How many tags will be shown. Value -1 means all of them. Default: -1 -->
  <property name="totalTags" value="-1"/>
  <!-- The letter case of the tags.
Possible values: Case.LOWER | Case.UPPER | Case.CAPITALIZATION |
Case.PRESERVE_CASE | Case.CASE_SENSITIVE
Default: Case.PRESERVE_CASE -->
  <property name="cloudCase" value="Case.PRESERVE_CASE"/>
  <!-- If the 3 CSS classes of the tag cloud should be independent of score (random=yes
) or based on the score. Possible values: true | false . Default: true-->
  <property name="randomColors" value="true"/>
  <!-- The font size (in em) for the tag with the lowest score. Possible values: any
decimal. Default: 1.1 -->
  <property name="fontFrom" value="1.1"/>
  <!-- The font size (in em) for the tag with the lowest score. Possible values: any
decimal. Default: 3.2 -->
  <property name="fontTo" value="3.2"/>
  <!-- The score that tags with lower than that will not appear in the rag cloud.
Possible values: any integer from 1 to infinity. Default: 0 -->
  <property name="cuttingLevel" value="0"/>
  <!-- The distance (in px) between the tags. Default: 5 -->
  <property name="marginRight" value="5"/>
  <!-- The ordering of the tags (based either on the name or the score of the tag)
Possible values: Tag.NameComparatorAsc | Tag.NameComparatorDesc |
Tag.ScoreComparatorAsc | Tag.ScoreComparatorDesc
Default: Tag.NameComparatorAsc -->
  <property name="ordering" value="Tag.NameComparatorAsc"/>
</bean>

```

When tagCloud is rendered there are some CSS classes that you can change in order to change the appearance of the tag cloud.

Class	Note
tagcloud	General class for the whole tagcloud
tagcloud_1	Specific tag class for tag of type 1 (baed on score)
tagcloud_2	Specific tag class for tag of type 2 (baed on score)
tagcloud_3	Specific tag class for tag of type 3 (baed on score)

Discovery Solr Index Maintenance

Command used:	<code>[dspace]/bin/dspace index-discovery [-cbhf[r <item handle>]]</code>
Java class:	<code>org.dspace.discovery.IndexClient</code>
Arguments (short and long forms):	Description
	called without any options, will update/clean an existing index
<code>-b</code>	(re)build index, wiping out current one if it exists
<code>-c</code>	clean existing index removing any documents that no longer exist in the db
<code>-f</code>	if updating existing index, force each handle to be reindexed even if uptodate
<code>-h</code>	print this help message
<code>-o</code>	optimize search core
<code>-r <item handle></code>	remove an Item, Collection or Community from index based on its handle

Routine Discovery Solr Index Maintenance

It is strongly recommended to run maintenance on the Discovery Solr index daily (from crontab or your system's scheduler), to prevent your servlet container from running out of memory:

```
[dspace]/bin/dspace index-discovery -o
```

Advanced Solr Configuration

Discovery is built as an application layer on top of the Solr open source enterprise search server. Therefore, Solr configuration can be applied to the Solr cores that are shipped with DSpace.

The DSpace Solr instance itself now runs two cores. One for collection DSpace Solr based "statistics", the other for Discovery Solr based "search".

```
solr
  search
    conf
      admin-extra.html
      elevate.xml
      protwords.txt
      schema.xml
      scripts.conf
      solrconfig.xml
      spellings.txt
      stopwords.txt
      synonyms.txt
    xslt
      DRI.xsl
      example.xsl
      example_atom.xsl
      example_rss.xsl
      luke.xsl
  conf2
  solr.xml
  statistics
    conf
      admin-extra.html
      elevate.xml
      protwords.txt
      schema.xml
      scripts.conf
      solrconfig.xml
      spellings.txt
      stopwords.txt
      synonyms.txt
    xslt
      example.xsl
      example_atom.xsl
      example_rss.xsl
      luke.xsl
```

4.7.2 Localization L10n

- 1 [Introduction](#)
- 2 [Common areas of localization](#)
 - 2.1 [Enabling additional locales](#)
 - 2.2 [Localization of email messages](#)
 - 2.3 [Metadata localization](#)
- 3 [XMLUI specific localization](#)

- [3.1 Message catalog](#)
- [3.2 Where to find the message catalog](#)
- [3.3 Where to edit](#)
- [3.4 Difference with JSPUI](#)
- 4 [JSPUI specific localization](#)
 - [4.1 Message catalog](#)
 - [4.2 Where to find the message catalog](#)
 - [4.3 Where to edit](#)
 - [4.4 Localization of input-forms.xml and license.default](#)
- 5 [Community translations for input-forms.xml and email messages](#)

Introduction

DSpace ships with a number of interface translations. This page provides information on areas that can be localized by means of configuration or customization. By default, DSpace will look at the user's browser language. If it has a language file in the user's language, it will render the interface in that language. If not, it will default to English or another default that you have configured.

Common areas of localization

Enabling additional locales

Out of the box, DSpace only has English enabled as a supported locale. Additional locales and the default locale are managed through the following parameters in `dspace.cfg`:

`dspace.cfg` configuration parameters

```
webui.supported.locales  
default.locale
```

You can change `default.locale` to a different one than English **after** adding it to `webui.supported.locales`.

Localization of email messages

All email templates used by DSpace can be found in

Path to the DSpace email templates

```
[dspace]/config/emails
```

The contents of the emails can be edited and translated.

Metadata localization

DSpace associates each metadata field value with a language code (though it may be left empty, e.g. for numeric values).

XMLUI specific localization

Message catalog

XMLUI supports multiple languages through the use of internationalization catalogues as defined by the [Cocoon Internationalization Transformer](#). Each catalog contains the translation of all user-displayed strings into a particular language or variant. Each catalog is a single xml file whose name is based upon the language it is designated for, thus:

`messages_language_country_variant.xml`

`messages_language_country.xml`

`messages_language.xml`

`messages.xml`

The interface will automatically determine which file to select based upon the user's browser and system configuration. For example, if the user's browser is set to Australian English then first the system will check if `messages_en_au.xml` is available. If this translation is not available it will fall back to `messages_en.xml`, and finally if that is not available, `messages.xml`.

Where to find the message catalog

The latest **English** message catalog is part of the main DSpace distribution and can be found at:

Location of the XMLUI message catalog in the DSpace source tree

```
[dspace-source]/dspace-xmlui/src/main/webapp/i18n/messages.xml
```

The **different translations** for this message catalog are being managed separately from the DSpace core project, in order to release updates for these files more frequently than the DSpace software itself. Visit the [dspace-xmlui-lang project on Github](#).

Where to edit

In some cases you may want to add additional keys to the message catalog or changing the particular wording of DSpace concepts. For example, you may want to change "Communities" into "Departments". These kind of changes may get automatically overwritten again when you upgrade to the newest version of DSpace. It is therefore advised to keep such changes isolated in the following location:

Recommended location for i18n customizations

```
[dspace-source]/dspace/modules/xmlui/src/main/webapp/i18n/
```

After rebuilding DSpace, any messages files placed in this directory will be automatically included in the XMLUI web application. Files of the same name will override any default files. By default, this full directory path may not exist or may be empty. If it does not exist, you can simply create it. You can place any number of translation catalogues in this directory. To add additional translations, just add another copy of the *messages.xml* file translated into the specific language and country variant you need.

After building and deploying, DSpace will finally read the files from the following location:

Location where your i18n files are being deployed

```
[dspace]/webapps/xmlui/i18n/messages.xml
```

Again, note that you will need to rebuild DSpace for these changes to take effect in your installed XMLUI web application!



Do not customize your messages in the webapps directory

While it seems like a fast option to change your messages straight in the deployed dspace directory, these changes are very volatile. If you rebuild and redeploy DSpace, these changes will get lost.

For more information about the `[dspace-source]/dspace/modules/` directory, and how it may be used to "overlay" (or customize) the default XMLUI interface, classes and files, please see: [Advanced Customisation](#)

Difference with JSPUI

In JSPUI, a wider range of files, including `input-forms.xml` and `default.license` can be localized by adding `_COUNTRY` at the end of the filename. This is currently not supported in XMLUI.

JSPUI specific localization

Message catalog

The [Java Standard Tag Library v1.0](#) is used to specify messages in the JSPs like this:

```
<H1><fmt:message key="jsp.search.results.title" /></H1>
```

This message can 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. Here is an example of a phrase in which two parameters are passed in:

```
jsp.search.results.text = Results {0}-{1} of {2}
```

Multiple *Messages.properties* can be created for different languages. See [ResourceBundle.getBundle](#). 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 by default. The user can change the language by clicking a link in the UI. These links are visible if more than one language is configured in DSpace. The English language file *Messages.properties* (or the default server locale) will be used as a fallback 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 because it is always available as a fallback, regardless of server configuration.

Where to find the message catalog

The latest **English** message catalog is part of the main DSpace distribution and can be found at:

Location of the JSPUI message catalog in the DSpace source tree

```
[dspace-source]/dspace-api/src/main/resources/Messages.properties
```

The **different translations** for this message catalog are being managed separately from the DSpace core project, in order to release updates for these files more frequently than the DSpace software itself. Visit the [dspace-api-lang project on Github](#).

Where to edit

In some cases you may want to add additional keys to the message catalog or changing the particular wording of DSpace concepts. For example, you may want to change "Communities" into "Departments". These kind of changes may get automatically overwritten again when you upgrade to the newest version of DSpace. It is therefore advised to keep such changes isolated in the following location:

Recommended location for i18n customizations

```
[dspace-source]/dspace/modules/jspui/src/main/resources/
```

After rebuilding DSpace, any messages files placed in this directory will be automatically included in the JSPUI web application. Files of the same name will override any default files. By default, this full directory path may not

exist or may be empty. If it does not exist, you can simply create it. You can place any number of translation catalogues in this directory. To add additional translations, just add another copy of the *Messages.properties* file translated into the specific language and country variant you need.

After building and deploying, DSpace will finally read the files from the `dspace-api-4.0.jar` file in your `[tomcat]\webapps\jspui\WEB-INF\lib` directory.

Again, note that you will need to rebuild DSpace for these changes to take effect in your installed JSPUI web application!

For more information about the `[dspace-source]/dspace/modules/` directory, and how it may be used to "overlay" (or customize) the default XMLUI interface, classes and files, please see: [Advanced Customisation](#)

Localization of `input-forms.xml` and `license.default`

The display labels for `input-forms.xml` and the text in the default submission license (`license.default`) are currently not managed in the messages catalogs. To localize these files, you can create versions of these files in the same folders, appending `_COUNTRY` at the end of the filename, before the extension. For example, `input-forms_de.xml` can be used to translate the submission form labels in German.

Community translations for `input-forms.xml` and email messages

Even though they are currently not managed on Github yet, some community translations are available for other files, such as emails and the `input-forms.xml`.

[Click here to access an overview of community translations \(DSpace wiki\)](#)

4.7.3 JSPUI Configuration and Customization

The DSpace digital repository supports two user interfaces: one based on JavaServer Pages (JSP) technologies and one based upon the Apache Cocoon framework (XMLUI). This chapter describes those parameters which are specific to the JPSUI interface.

- 1 [Configuration](#)
- 2 [Customizing the JSP pages](#)

Configuration

The user will need to refer to the extensive [WebUI/JSPUI configurations](#) that are contained in JSP Web Interface Settings.

Customizing the JSP pages

The JSPUI interface is implemented using Java Servlets which handle the business logic, and JavaServer Pages (JSPs) which produce the HTML pages sent to an end-user. Since the JSPs are much closer to HTML than Java code, altering the look and feel of DSpace is relatively easy.

To make it even easier, DSpace allows you to 'override' the JSPs included in the source distribution with modified versions, that are stored in a separate place, so when it comes to updating your site with a new DSpace release, your modified versions will not be overwritten. It should be possible to dramatically change the look of DSpace to suit your organization by just changing the CSS style file and the site 'skin' or 'layout' JSPs in */jsp/layout*, if possible, it is recommended you limit local customizations to these files to make future upgrades easier.

You can also easily edit the text that appears on each JSP page by editing the *Messages.properties* file. However, note that unless you change the entry in all of the different language message files, users of other languages will still see the default text for their language. See [Internationalization](#) in [Application Layer](#).

Note that the data (attributes) passed from an underlying Servlet to the JSP may change between versions, so you may have to modify your customized JSP to deal with the new data.

Thus, if possible, it is recommended you limit your changes to the 'layout' JSPs and the stylesheet.

The JSPs are available in one of two places:

- *[dspace-source]/dspace-jspui/dspace-jspui-webapp/src/main/webapp/* - Only exists if you downloaded the full Source Release of DSpace
- *[dspace-source]/dspace/target/dspace-[version].dir/webapps/dspace-jspui-webapp/* - The location where they are copied after first building DSpace.

If you wish to modify a particular JSP, place your edited version in the *[dspace-source]/dspace/modules/jspui/src/main/webapp/* directory (*this is the replacement for the pre-1.5 /jsp/local directory*), with the same path as the original. If they exist, these will be used in preference to the default JSPs. For example:

DSpace default	Locally-modified version
<i>[jsp.dir]/community-list.jsp</i>	<i>[jsp.custom-dir]/dspace/modules/jspui/src/main/webapp/community-list.jsp</i>
<i>[jsp.dir]/myspace/main.jsp</i>	<i>[jsp.custom-dir]/dspace/modules/jspui/src/main/webapp/myspace/main.jsp</i>

Heavy use is made of a style sheet, *styles.css*. If you make edits, copy the local version to *[jsp.custom-dir]/dspace/modules/jspui/src/main/webapp/styles.css*, 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 `[jsp.custom-dir]/dSPACE/modules/jspui/src/main/webapp/layout`), or define more styles and apply them to pages by using the "style" attribute of the `dSPACE:layout` tag.

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

```
mvn package
```

2. Update all DSpace webapps to `[dSPACE]/webapps` by running the following command from your `[dSPACE-source]/dSPACE/target/dSPACE-installer` directory:

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

3. Deploy the the new webapps:

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

4. Restart Tomcat

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

4.7.4 XMLUI Configuration and Customization

The DSpace digital repository supports two user interfaces: one based on JavaServer Pages (JSP) technologies and one based upon the Apache Cocoon framework (XMLUI). This chapter describes those parameters which are specific to the Manakin (XMLUI) interface based upon the Cocoon framework.

- [1 Overview of XMLUI / Manakin](#)
 - [1.1 Understanding the Flow of an XMLUI Request](#)
- [2 Manakin Configuration Property Keys](#)
- [3 Configuring Themes and Aspects](#)
 - [3.1 Aspects](#)
 - [3.2 Themes](#)
- [4 Multilingual Support](#)
- [5 Creating a New Theme](#)
- [6 Customizing the News Document](#)
- [7 Adding Static Content](#)
- [8 Harvesting Items from XMLUI via OAI-ORE or OAI-PMH](#)
 - [8.1 Automatic Harvesting \(Scheduler\)](#)
- [9 Additional XMLUI Learning Resources](#)

Overview of XMLUI / Manakin

For more information & diagrams

For a more detailed overview of XMLUI/Manakin, see the following resources:

- [Introducing Manakin \(XMLUI\)](#) - Provides an overview of what XMLUI is and how it works.
- [Learning to Use Manakin \(XMLUI\)](#) - Overview of how to use Manakin and how it works. Based on DSpace 1.5, but also valid for current versions
- [Making DSpace XMLUI Your Own](#) - Concentrates on using Maven to build Overlays in the XMLUI (Manakin). Also has very basic examples for JSPUI. Based on DSpace 1.6.x but also valid for current versions.

The XMLUI (aka Manakin) is built on [Apache Cocoon framework](#). The XMLUI uses Cocoon to provide a modular , extendable, tiered interface framework

The XMLUI essentially consists of three main tiers, in increasing order of complexity:

1. **Style Tier** - allows one to use CSS and simple XHTML to stylize an existing XMLUI Theme
2. **Theme Tier** - allows one to use XSLT, XHTML and CSS to create new, more complex XMLUI Theme(s)
3. **Aspect Tier** - allows one to use the Cocoon framework and Java (or XSLT) to create new features (aspects), and generate new content into DRI.

These tiers are very important and powerful because of their modularity. For example, based on your local expertise with these technologies, your institution may decide to only modify the XMLUI at the "Style Tier" (by just modifying CSS & images in an existing theme). As you learn more about themes & aspects, you may decide to slowly venture into the more complex "Theme Tier" and finally into the "Aspect Tier". Other institutions may determine that all they really need to ever do is make "Style Tier" changes.

Digging in a little deeper, there are three main XMLUI components that are unique to the XMLUI and used throughout the system. These main components are:

- [DRI Schema](#)- Digital Repository Interface (DRI) XML schema, which is the "abstract representation of a single repository page". The DRI document is XML that contains all of the information (metadata) available for display on a given page within the XMLUI. This information includes:
 - Metadata elements (described in METS, MODS, DSpace Internal Metadata (DIM), Qualified Dublin Core, etc.)
 - Structural elements (described in TEI light)
 - For more specific information about DRI Schema along with examples, see [DRI Schema Reference](#).

- **#Aspects** - One or more aspects are enabled at a given time. Generally speaking an aspect implements a set of related features within the XMLUI. More specifically, the enabled aspects are what **build** the DRI document. So, Aspects are the only things that can change the structure of the DRI document (or add/remove content to/from DRI)
 - Aspects apply to all pages across your entire DSpace site. Each aspect must take a valid DRI document as its input, and also output a valid DRI document.
 - Aspects usually are written in Java (and controlled by a Cocoon "sitemap.xmap"). However, Aspects can also be written in XSLT (provided that the input and output are both valid DRI documents)
- **#Themes** - One or more themes are enabled at a given time. Themes are in charge of stylizing content into a particular look & feel. More specifically, a theme is what transforms a DRI document into XHTML (and adds any CSS, javascript, images, etc).
 - A single Theme may apply to your entire DSpace site, just a specific Community or Collection (and all members of that Community/Collection), or just a specific page.
 - A Theme may consist of one or more of the following: an XSLT stylesheet, CSS stylesheets, images, other static resources.
 - More information on creating a theme is available at: [#Creating a New Theme](#)
 - Additional Theme Resources include:
 - [Manakin theme tutorial](#)
 - [Manakin Themes and Recipes](#)
 - [Create a new theme \(Manakin\)](#)

Understanding the Flow of an XMLUI Request

One of the harder things to initially grasp in the XMLUI is how a single user's request (e.g. clicking on a link or button) flows through the entire system of enabled Aspects and Themes. Understanding this flow is also very important as you work to build your own Aspects (or complex Themes), as it may allow you to more easily determine what is going on in the system.

Before getting started, it's worth mentioning that this request flow is controlled via a series of Cocoon Sitemap files (named `sitemap.xmap`, `themes.xmap` and `aspects.xmap`). These Sitemap files are Cocoon's way of defining the flow. More information about Cocoon Sitemaps is available at: <http://cocoon.apache.org/2.1/userdocs/concepts/sitemap.html>

The following explanation provides a high level overview of how a request is processed, how a DRI document is generated (via Aspects), and then how it is transformed into XHTML (via Themes). As this is a high level overview, some details are likely left out, but the overarching flow is what is most important.

1. A user visits an XMLUI page (by clicking a link or button, etc)
2. That request begins in the root Cocoon `sitemap.xmap` (located at `[xmlui]/sitemap.xmap`). This is the main entry point for **all requests**
 - a. Within that sitemap, various URL path matching takes place. If the request is to download a document, that document is returned immediately.

- b. However, in many cases, the request is for a page within the XMLUI. In this scenario, the root `sitemap.xmap` will load the `[xmlui]/themes/themes.xmapfile`, which controls all the Themes.
 - i. The `themes.xmap` file will then load all "matching" themes which are configured in your `[dspace]/config/xmlui.xconf` file (see [#Themes](#) below).
 - ii. If more than one theme matches the current URL path, then the **first match wins**
 - iii. Once a matching theme is located, that theme's `sitemap.xmapfile` (located in its theme directory) is loaded and processed.
 1. The theme's `sitemap.xmap` is in charge of actually loading the theme's XSLT, CSS, etc. However, before it does that, you'll notice it makes a call to generate the DRI document for the current page as follows:

```

<map:generate type="file" src="cocoon://DRI/{1}"/>
```

2. This DRI call generates a brand new, internal Cocoon request. This request is then processed back in the **root** `sitemap.xmap` (remember how we said that this `sitemap` is the main entry point for all requests).
3. Back in the root `sitemap`, the "DRI/**" call is matched. This causes the `[xmlui]/aspects/aspects.xmapfile` to be loaded. As the name suggests, this file obviously controls all the Aspects.
 - a. The `aspects.xmap` file will then load all enabled Aspects which are configured in your `[dspace]/config/xmlui.xconf` file (see [#Aspects](#) below).
 - b. Each aspect is loaded in the **order that it appears**. However, **multiple** aspects may be loaded for the same URL path. *Remember, aspects can build upon each other (we call this an "aspect chain") as they work together to generate the final DRI document.*
 - c. When an Aspect is loaded, its `sitemap.xmap` is loaded & processed
 - NOTE: An aspect's `sitemap.xmap` is actually compiled into the `dspace-xmlui-api.jar` file. However, if you have a copy of DSpace source handy, it can be found in: `[dspace-src]/dspace-xmlui/dspace-xmlui-api/src/main/resources/aspects/[name-of-aspect]/`
 - d. Each aspect is processed one-by-one (again in the order they are listed in `xmlui.xconf`). Each aspect may add, remove or change content within the DRI document. After the final aspect is finished processing, the DRI document is complete.
 - HINT: In the XMLUI you can always view the final DRI document by adding "?XML" or "&XML" on to the end of the current URL in your web browser.
4. Once the final DRI document is complete (all aspects are done processing), the flow will return back to your Theme's `sitemap.xmap` (remember, this is the same location that triggered the loading of the Aspects in the first place).
5. At this point, your Theme's `sitemap.xmap` will continue its processing. Generally speaking, most themes will then perform one or more XSLT transformations (to transform the final DRI document into XHTML). They also may load up one or more CSS files to help stylize the final XHTML.
6. Finally, once the Theme has completed its processing (remember, only one theme is ever processed for a single request), the final generated XHTML document is displayed to the user.

Again, the above flow is a slightly simplified version of what is going on underneath the XMLUI. As you can see, [Cocoon Sitemaps](#) are what control most of the XMLUI processing (and the loading of the Aspects and Theme).

Manakin Configuration Property Keys

In an effort to save the programmer/administrator some time, the configuration table below is taken from 5.3.43. *XMLUI Specific Configuration*.

Property:	<i>xmlui.supportedLocales</i>
Example Value:	<i>xmlui.supportedLocales = en, de</i>
Informational Note:	A list of supported locales for Manakin. Manakin will look at a user's browser configuration for the first language that appears in this list to make available to in the interface. This parameter is a comma separated list of Locales. All types of Locales country, country_language, country_language_variant. Note that if the appropriate files are not present (i.e. Messages_XX_XX.xml) then Manakin will fall back through to a more general language.
Property:	<i>xmlui.force.ssl</i>
Example Value:	<i>xmlui.force.ssl = true</i>
Informational Note:	Force all authenticated connections to use SSL, only non-authenticated connections are allowed over plain http. If set to true, then you need to ensure that the ' <i>dspace.hostname</i> ' parameter is set to the correctly.
Property:	<i>xmlui.user.registration</i>
Example Value:	<i>xmlui.user.registration = true</i>
Informational Note:	Determine if new users should be allowed to register. This parameter is useful in conjunction with Shibboleth where you want to disallow registration because Shibboleth will automatically register the user. Default value is true.
Property:	<i>xmlui.user.editmetadata</i>
Example Value:	<i>xmlui.user.editmetadata = true</i>
Informational Note:	Determines if users should be able to edit their own metadata. This parameter is useful in conjunction with Shibboleth where you want to disable the user's ability to edit their metadata because it came from Shibboleth. Default value is true.
Property:	<i>webui.user.assumelogin</i>

Example Value:	<i>webui.user.assumelogin = true</i>
Informational Note:	Determine if super administrators (those whom are in the Administrators group) can login as another user from the "edit eperson" page. This is useful for debugging problems in a running dspace instance, especially in the workflow process. The default value is false, i.e., no one may assume the login of another user.
Property:	<i>xmlui.user.loginredirect</i>
Example Value:	<i>xmlui.user.loginredirect = /profile</i>
Informational Note:	After a user has logged into the system, which url should they be directed? Leave this parameter blank or undefined to direct users to the homepage, or <i>/profile</i> for the user's profile, or another reasonable choice is <i>/submissions</i> to see if the user has any tasks awaiting their attention. The default is the repository home page.
Property:	<i>xmlui.theme.allowoverrides</i>
Example Value:	<i>xmlui.theme.allowoverrides = false</i>
Informational Note:	Allow the user to override which theme is used to display a particular page. When submitting a request add the HTTP parameter "themepath" which corresponds to a particular theme, that specified theme will be used instead of the any other configured theme. Note that this is a potential security hole allowing execution of unintended code on the server, this option is only for development and debugging it should be turned off for any production repository. The default value unless otherwise specified is "false".
Property:	<i>xmlui.bundle.upload</i>
Example Value:	<i>xmlui.bundle.upload = ORIGINAL, METADATA, THUMBNAIL, LICENSE, CC_LICENSE</i>
Informational Note:	Determine which bundles administrators and collection administrators may upload into an existing item through the administrative interface. If the user does not have the appropriate privileges (add and write) on the bundle then that bundle will not be shown to the user as an option.
Property:	<i>xmlui.community-list.render.full</i>
Example Value:	<i>xmlui.community-list.render.full = true</i>
Informational Note:	

	<p>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.</p>
Property:	<i>xmlui.community-list.cache</i>
Example Value:	<i>xmlui.community-list.cache = 12 hours</i>
Informational Note:	<p>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.</p>
Property:	<i>xmlui.bistream.mods</i>
Example Value:	<i>xmlui.bistream.mods = true</i>
Informational Note:	<p>Optionally, you may configure Manakin to take advantage of metadata stored as a bitstream. The MODS metadata file must be inside the "METADATA" bundle and named MODS.xml. If this option is set to 'true' and the bitstream is present then it is made available to the theme for display.</p>
Property:	<i>xmlui.bitstream.mets</i>
Example Value:	<i>xmlui.bitstream.mets = true</i>
Informational Note:	<p>Optionally, you may configure Manakin to take advantage of metadata stored as a bitstream. The METS metadata file must be inside the "METADATA" bundle and named METS.xml. If this option is set to "true" and the bitstream is present then it is made available to the theme for display.</p>
Property:	<i>xmlui.google.analytics.key</i>
Example Value:	<i>xmlui.google.analytics.key = UA-XXXXXX-X</i>
Informational Note:	<p>If you would like to use google analytics to track general website statistics then use the following parameter to provide your analytics key. First sign up for an account at http://analytics.google.com, then create an entry for your repositories website. Google Analytics will give you a snippet of javascript code to place on your site, inside that snip it is your Google Analytics key usually found in the line: <code>_uacct = "UA-XXXXXX-X"</code> Take this key (just the UA-XXXXXX-X part) and place it here in this parameter.</p>

Property:	<i>xmlui.controlpanel.activity.max</i>
Example Value:	<i>xmlui.controlpanel.activity.max = 250</i>
Informational Note:	Assign how many page views will be recorded and displayed in the control panel's activity viewer. The activity tab allows an administrator to debug problems in a running DSpace by understanding who and how their DSpace is currently being used. The default value is 250.
Property:	<i>xmlui.controlpanel.activity.ipheader</i>
Example Value:	<i>xmlui.controlpanel.activity.ipheader = X-Forward-For</i>
Informational Note:	Determine where the control panel's activity viewer receives an events IP address from. If your DSpace is in a load balanced environment or otherwise behind a context-switch then you will need to set the parameter to the HTTP parameter that records the original IP address.

Configuring Themes and Aspects

The Manakin user interface is composed of two distinct components: *aspects* and *themes*. Manakin aspects are like extensions or plugins for Manakin; they are interactive components that modify existing features or provide new features for the digital repository. Manakin themes stylize the look-and-feel of the repository, community, or collection.

The repository administrator is able to define which aspects and themes are installed for the particular repository by editing the `[dspace]/config/xmlui.xconf` configuration file. The `xmlui.xconf` file consists of two major sections: Aspects and Themes.

Aspects

The `<aspects>` section defines the "Aspect Chain", or the linear set of aspects that are installed in the repository. For each aspect that is installed in the repository, the aspect makes available new features to the interface. For example, if the "submission" aspect were to be commented out or removed from the `xmlui.xconf`, then users would not be able to submit new items into the repository (even the links and language prompting users to submit items are removed). Each `<aspect>` element has two attributes, *name* and *path*. The name is used to identify the Aspect, while the path determines the directory where the aspect's code is located. Here is the default aspect configuration:

```

<aspects>
  <aspect name="Displaying Artifacts" path="resource://aspects/ViewArtifacts/" />
  <aspect name="Browsing Artifacts" path="resource://aspects/BrowseArtifacts/" />
  <aspect name="Searching Artifacts" path="resource://aspects/SearchArtifacts/" />
  <aspect name="Administration" path="resource://aspects/Administrative/" />
  <aspect name="E-Person" path="resource://aspects/EPerson/" />
  <aspect name="Submission and Workflow" path="resource://aspects/Submission/" />
  <aspect name="Statistics" path="resource://aspects/Statistics/" />

```

```
<aspect name="Original Workflow" path="resource://aspects/Workflow/" />
</aspects>
```

A standard distribution of Manakin/DSpace includes eight "core" aspects:

- **ViewArtifacts** The ViewArtifacts Aspect is responsible for displaying individual item metadata.
- **BrowseArtifacts** The BrowseArtifacts Aspect is responsible for displaying different browse options
- **SearchArtifacts** The SearchArtifacts Aspect displays the different search boxes. **Shouldn't be activated together with DSpace Discovery.**
- **Administrative** The Administrative Aspect is responsible for administrating DSpace, such as creating, modifying and removing all communities, collections, e-persons, groups, registries and authorizations.
- **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.
- **Statistics** The Statistics Aspect is responsible for displaying statistics information.
- **Workflow** The Original Workflow Aspect is responsible for displaying workflow tasks. **Shouldn't be activated with the new workflow called XMLWorkflow**

Following Aspects are optional

- **XMLWorkflow** This Aspect was added in DSpace 1.8 and is responsible for the new configurable workflow system. **Shouldn't be activated together with the Workflow aspect.**
- **Discovery** The Discovery Aspect replaces the standard search with faceted searching. It also takes care of the faceted browse options. **Shouldn't be activated together with SearchArtifacts.**
- **SwordClient** The SwordClient aspect displays options that allow you to "push" DSpace content to another SWORD-server enabled system.
- **XMLTest** An aspect to assist developers in creating themes, as it displays different debugging options.

Following Aspects are deprecated and shouldn't be used anymore at all

- **ArtifactBrowser** This aspect has been split up into ViewArtifacts, BrowseArtifacts and SearchArtifacts in DSpace 1.7.0

Themes

The *<themes>* section defines a set of "rules" that determine where themes are installed in the repository. Each rule is processed in the order that it appears, and the first rule that matches determines the theme that is applied (so order is important). Each rule consists of a *<theme>* element with several possible attributes:

- **name** (*always required*) The name attribute is used to document the theme's name.
- **path** (*always required*) The path attribute determines where the theme is located relative to the *themes/* directory and must either contain a trailing slash or point directly to the theme's *sitemap.xml* file.
- **regex** (*either regex and/or handle is required*) The regex attribute determines which URLs the theme should apply to.

- **handle** (*either regex and/or handle is required*) The handle attribute determines which community, collection, or item the theme should apply to.

If you use the "handle" attribute, the effect is cascading, meaning if a rule is established for a community then all collections and items within that community will also have this theme apply to them as well. Here is an example configuration:

```
<themes>
  <theme name="Theme 1" handle="123456789/23" path="theme1/" />
  <theme name="Theme 2" regex="community-list" path="theme2/" />
  <theme name="Reference Theme" regex=".*" path="Reference/" />
</themes>
```

In the example above three themes are configured: "Theme 1", "Theme 2", and the "Reference Theme". The first rule specifies that "Theme 1" will apply to all communities, collections, or items that are contained under the parent community "123456789/23". The next rule specifies any URL containing the string "community-list" will get "Theme 2". The final rule, using the regular expression ".*", **will match anything**, so all pages which have not matched one of the preceding rules will be matched to the Reference Theme.

Multilingual Support

The XMLUI user interface supports multiple languages through the use of internationalization catalogues as defined by the [Cocoon Internationalization Transformer](#). Each catalog contains the translation of all user-displayed strings into a particular language or variant. Each catalog is a single xml file whose name is based upon the language it is designated for, thus:

`messages_ language_ country_ variant.xml`

`messages_ language_ country.xml`

`messages_ language.xml`

`messages.xml`

The interface will automatically determine which file to select based upon the user's browser and system configuration. For example, if the user's browser is set to Australian English then first the system will check if `messages_en_au.xml` is available. If this translation is not available it will fall back to `messages_en.xml`, and finally if that is not available, `messages.xml`.

DSpace XMLUI supplies an English only translation of the interface, which can be found in the XMLUI web application (`[dspace]/webapps/xmlui/i18n/messages.xml`), after you first build DSpace.

If you wish to add other translations to the system, or make customizations to the existing `messages.xml` file, you can place them in the following directory:

```
[dspace-source]/dspace/modules/xmlui/src/main/webapp/i18n/
```

After rebuilding DSpace, any messages files placed in this directory will be automatically included in the XMLUI web application (and files of the same name will override any default files). By default this full directory path may not exist (if not, just create it) or may be empty. You can place any number of translation catalogues in this directory. To add additional translations, just 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, you can simply rename the translation catalogue for the new default language to "*messages.xml*".

Again, note that you will need to rebuild DSpace for these changes to take effect in your installed XMLUI web application!

For more information about the `[dspace-source]/dspace/modules/` directory, and how it may be used to "overlay" (or customize) the default XMLUI interface, classes and files, please see: [Advanced Customisation](#)

Creating a New Theme

Manakin themes stylize the look-and-feel of the repository, community, or collection and are distributed as self-contained packages. A Manakin/DSpace installation may have multiple themes installed and available to be used in different parts of the repository. The central component of a theme is the *sitemap.xmap*, which defines what resources are available to the theme such as XSL stylesheets, CSS stylesheets, images, or multimedia files.

1) Create theme skeleton

Most theme developers do not create a new theme from scratch; instead they start from the standard theme template, which defines a skeleton structure for a theme. The template is located at: `[dspace-source]/dspace-xmlui/dspace-xmlui-webbapp/src/main/webbapp/themes/template`. To start your new theme simply copy the theme template into your locally defined modules directory, `[dspace-source]/dspace/modules/xmlui/src/main/webbapp/themes/[your theme's directory]`.

2) Modify theme variables

The next step is to modify the theme's parameters so that the theme knows where it is located. Open the `[your theme's directory]/sitemap.xmap` and look for `<global-variables>`

```
<global-variables>
  <theme-path>[your theme's    directory]</theme-path>
  <theme-name>[your theme's name]</theme-name>
</global-variables>
```

Update both the theme's path to the directory name you created in step one. The theme's name is used only for documentation.

3) Add your CSS stylesheets

The base theme template will produce a repository interface without any style - just plain XHTML with no color or formatting. To make your theme useful you will need to supply a CSS Stylesheet that creates your desired look-and-feel. Add your new CSS stylesheets:

[your theme's directory]/lib/style.css (The base style sheet used for all browsers)

[your theme's directory]/lib/style-ie.css (Specific stylesheet used for internet explorer)

4) Install theme and rebuild DSpace

Next rebuild and deploy DSpace (replace <version> with the your current release):

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

```
mvn package
```

2. Update all DSpace webapps to *[dspace]/webapps* by running the following command from your *[dspace-source]/dspace/target/dspace-[version]-build.dir* directory:

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

3. Deploy the the new webapps:

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

4. Restart Tomcat

This will ensure the theme has been installed as described in the previous section "Configuring Themes and Aspects".

Customizing the News Document

The XMLUI "news" document is only shown on the root page of your repository. It was intended to provide the title and introductory message, but you may use it for anything.

The news document is located at *[dspace]/dspace/config/news-xmlui.xml*. There is only one version; it is localized by inserting "i18n" callouts into the text areas. It must be a complete and valid XML DRI document (see Chapter 15).

Its (the News document) exact rendering in the XHTML UI depends, of course, on the theme. The default content is designed to operate with the reference themes, so when you modify it, be sure to preserve the tag structure and e.g. the exact attributes of the first DIV tag. Also note that the text is DRI, not HTML, so you must use only DRI tags, such as the XREF tag to construct a link.

Example 1: a single language:

```

<document>
  <body>
    <div id="file.news.div.news" n="news" rend="primary">
      <head> TITLE OF YOUR REPOSITORY HERE </head>
      <p>
        INTRO MESSAGE HERE
        Welcome to my wonderful repository etc etc ...
        A service of <xref target="http://myuni.edu/">My University</xref>
      </p>
    </div>
  </body>
</options/>
<meta>
  <userMeta/>
  <pageMeta/>
  <repositoryMeta/>
</meta>
</document>

```

Example 2: all text replaced by references to localizable message keys:

```

<document>
  <body>
    <div id="file.news.div.news" n="news" rend="primary">
      <head><i18n:text>myuni.repo.title</i18n:text></head>
      <p>
        <i18n:text>myuni.repo.intro</i18n:text>
        <i18n:text>myuni.repo.a.service.of</i18n:text>
        <xref target="http://myuni.edu/"><i18n:text>myuni.name</i18n:text></xref>
      </p>
    </div>
  </body>
</options/>
<meta>
  <userMeta/>
  <pageMeta/>
  <repositoryMeta/>
</meta>
</document>

```

Adding Static Content

The XMLUI user interface supports the addition of globally static content (as well as static content within individual themes).

Globally static content can be placed in the `[dspace-source]/dspace/modules/xmlui/src/main/webapp/static/` directory. By default this directory only contains the default `robots.txt` file, which provides helpful site information to web spiders/crawlers. However, you may also add static HTML (`*.html`) content to this directory, as needed for your installation.

Any static HTML content you add to this directory may also reference static content (e.g. CSS, Javascript, Images, etc.) from the same `[dspace-source]/dspace/modules/xmlui/src/main/webapp/static/` directory. You may reference other static content from your static HTML files similar to the following:

```
<link href="./static/mystyle.css" rel="stylesheet" type="text/css"/>


```

Harvesting Items from XMLUI via OAI-ORE or OAI-PMH

This feature allows you to harvest Items (both metadata and bitstreams) from one DSpace to another DSpace or from one OAI-PMH/OAI-ORE server to a DSpace instance.

This section will give the necessary steps to set up the OAI-ORE/OAI-PMH Harvester from the XMLUI (Manakin). This feature is currently not available in the JSPUI.

Setting up a Harvesting Collection:

1. Login to XMLUI and create a new collection.
2. Go to the tab named "Content Source" that appears next to "Edit Metadata" and "Assign Roles " in the collection edit screens.
3. The two "Content Source" options are "standard DSpace collection" (selected by default) and "collection harvests its content from an external source". Select "harvests from an external source" option and click Save.
4. A new set of menus appear to configure the harvesting settings:
 - "OAI Provider" is in the URL of the OAI-PMH provider that the content from this collection should be harvested from. The OAI-PMH provider deployed with DSpace typically has the format: `http://dspace.url/oai/request` For example, you could use the Demo DSpace OAI-PMH provider: `"http://demo.dspace.org/oai/request"`
 - "OAI Set Id" is the [OAI-PMH setSpec](#) of the collection you wish to harvest from. For DSpace, this Set ID has the format: `hdl_<handle-prefix>_<handle-suffix>`. For example `"hdl_10673_2"` would refer to the Collection whose handle is "10673/2" (on the DSpace Demo Server, this is the [Collection of Sample Items](#)). If the target instance is using OAI 2.0 (DSpace 3.0 or the OAI 2.0 addon for DSpace 1.8.2), replace `"hdl_"` with `"col_"` if the set is a collection or with `"com_"` if it's a community.
 - "Metadata format" determines the format that the descriptive metadata will be harvested. The OAI-PMH server of the source DSpace instance may only support certain metadata formats. Select "DSpace Intermediate Metadata" if available (as this provides the richest metadata transfer) and "Simple Dublin Core" otherwise

- To determine which metadata formats an OAI-PMH server supports, you can send a `ListMetadataFormats` request to that OAI-PMH server. Typically this has the format: `http://dspace.url/oai/request?verb=ListMetadataFormats` For example, you can see which metadata formats are supported by the DSpace Demo Server by visiting: <http://demo.dspace.org/oai/request?verb=ListMetadataFormats>
 - Click the "Test Settings" button to verify the settings supplied in the previous steps. This will usually let you know if anything is missing or does not validate correctly. If you receive an error, you will need to fix the settings before continuing
5. The list of radio buttons labeled "Content being harvested" allows you to select the level of harvest. These harvesting options include:
- *Harvest Metadata Only* - will only harvest item metadata from the source DSpace (or any OAI-PMH source)
 - *Harvest metadata and references to bitstreams (requires ORE support)* - will harvest item metadata and create links to files/bitstreams (stored remotely) from the source DSpace (requires OAI-ORE)
 - *Harvest metadata and bitstreams (requires ORE support)* - performs a full local replication. Harvests both item metadata and files/bitstreams (requires OAI-ORE).
6. Select the appropriate option based on your needs, and click Save

At this point the settings are saved and the menu changes to provide three options:

- *Change Settings*: takes you back to the edit screen (see above instructions)
- *Import Now*: performs a single harvest from the remote collection into the local one. Success, notes, and errors encountered in the process will be reflected in the "Last Harvest Result" entry. More detailed information is available in the DSpace log.

"Import Now" May Timeout for Large Harvests

Note that the whole harvest cycle is executed within a single HTTP request and will time out for large collections. For this reason, it is advisable to use the [automatic harvest scheduler](#) set up either in XMLUI or from the command line. If the scheduler is running, "Import Now" will handle the harvest task as a separate thread.

- *Reset and Reimport Collection*: will perform the same function as "Import Now", but will clear the collection of all existing items before doing so.

Automatic Harvesting (Scheduler)

Setting up automatic harvesting in the Control Panel Screen.

- Login as an Administrative user in XMLUI
- Visit the "Harvesting" tab under "Administrative > Control Panel"
- The panel offers the following information:

- Available actions:
 - *Start Harvester*: starts the scheduler. From this point on, all properly configured collections (listed on the next line) will be harvested at regular intervals. This interval can be changed in the `dspace.cfg` using the `harvester.harvestFrequency` parameter.
 - *Pause*: the "nice" stop; waits for the active harvests to finish, saves the state/progress and pauses execution. Can be either resumed or stopped.
 - *Stop*: the "full stop"; waits for the current item to finish harvesting, and aborts further execution.
 - *Reset Harvest Status*: since stopping in the middle of a harvest is likely to result in collections getting "stuck" in the queue, the button is available to clear all states.

Additional XMLUI Learning Resources

Useful links with further information into XMLUI Development

- [Making DSpace XMLUI Your Own](#) - Concentrates on using Maven to build Overlays in the XMLUI (Manakin). Also has very basic examples for JSPUI. Based on DSpace 1.6.x.
- [Learning to Use Manakin \(XMLUI\)](#) - Overview of how to use Manakin and how it works. Based on DSpace 1.5, but also valid for 1.6.
- [Introducing Manakin \(XMLUI\)](#)

Mirage 2 Configuration and Customization

```
/*<![CDATA[* / div.rbtoc1424986283028 {padding: 0px;} div.rbtoc1424986283028 ul {list-style: none;margin-left: 0px;} div.rbtoc1424986283028 li {margin-left: 0px;padding-left: 0px;} /*]]>*/
```

- 1 [Introduction](#)
- 2 [Responsive design](#)
- 3 [The build process and enabling Mirage 2](#)
 - 3.1 [Common Build Issues](#)
- 4 [Configuration options](#)
- 5 [Customizing Mirage 2](#)
 - 5.1 [The Mirage 2 color scheme](#)
 - 5.2 [Simple styling customization](#)
 - 5.3 [Advanced styling customizations](#)
- 6 [Automatically retrieving the latest versions of Mirage 2 dependencies](#)
- 7 [Additional Developer documentation](#)

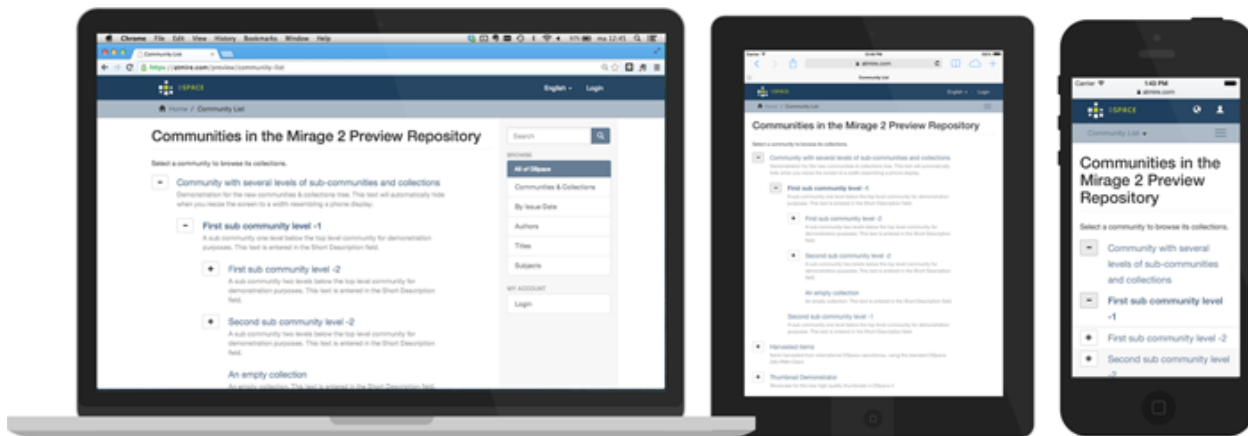
Introduction

Mirage has been the default XMLUI theme since DSpace 1.7 and has been used as base point for most custom themes. DSpace 5 includes Mirage 2, also developed by [@mire](#), an upgrade to Mirage built on modern web

technologies. The predominant improvement for the end user is the responsive design. Repository admins and developers will also benefit because of the tools available to make both simple and advanced customizations.

Responsive design

A responsive website is a website that rearranges its content to fit in different screen sizes. The main focus is to provide a better overall user experience whether you're browsing on a mobile phone, a tablet or desktop computer. As opposed to using a separate mobile theme, there's only one version of the website that will work everywhere. A simple way to find out what the differences are between a narrow screen and a larger screen in Mirage 2, go to any webpage and resize the browser window. You will notice the sidebar is only shown when the window is large enough, otherwise a menu button is displayed to get to the sidebar options. The theme provides a distinct look for each of the 3 different categories of screen sizes: mobile, tablet and desktop.



The build process and enabling Mirage 2

The modern web technologies that power Mirage 2 include a precompiler ([Compass](#)), a package manager ([Bower](#)) and a task runner ([Grunt](#)). These tools can only be installed when some prerequisites are present on the system. DSpace's Maven build process is capable of making a temporary installation of these dependencies just so the theme can be built. However the overall build time will be significantly shortened if these dependencies are manually installed on the system (see below for more info).

All of the Mirage 2 builds require git. Make sure to install git before starting any of the Mirage 2 builds.

- By default, DSpace does *not* build the Mirage 2 theme (as it lengthens the normal build process). However, you can easily tell DSpace to build Mirage 2 by running the following from your [`dspace-source`] directory:

```
mvn package -Dmirage2.on=true
```

- If you wish to speed up the Mirage 2 build process, you can do so by pre-installing all of the Mirage 2 dependencies on your system (by default they will be downloaded each time you rebuild Mirage 2). This will *significantly shorten* the build process for Mirage 2. More information on installing these prerequisites can be found in the [Developer Documentation for Mirage 2](#). Once these

prerequisites have been installed on your local server, you can then build Mirage 2 more rapidly by running:

```
# WARNING: This command will only work if you've manually installed *all* the
prerequisites for Mirage 2
mvn package -Dmirage2.on=true -Dmirage2.deps.included=false
```

- After building Mirage 2, you can install this theme into your DSpace by simply re-running Ant from [dspace-source]/dspace/target/dspace-installer/ :

```
ant update
```

- To enable Mirage 2, add the following to the <themes> section of your xmlui.xconf, replacing the currently active theme:

```
<theme name="Mirage 2" regex=".*" path="Mirage2/" />
```

- Finally, restart your Tomcat or servlet container, and you should see the Mirage 2 theme.

Common Build Issues

- Running the Mirage 2 build (mvn package -Dmirage.on=true) as the "root" user (or via sudo) will result in the following error from "Bower". This will result in a broken Mirage 2 build. The fix is to ensure you are building DSpace as a **non-root** user account. For more information on this Bower error, see: <http://serverfault.com/questions/548537/cant-get-bower-working-bower-esudo-cannot-be-run-with-sudo>

```
bower ESUDO  Cannot be run with sudo
```

```
Additional error details:
```

```
Since bower is a user command, there is no need to execute it with superuser permissions.
```

- The Mirage 2 build requires git. Ensure that git is installed before you launch the Mirage 2 build.
- The Mirage 2 build process will attempt to retrieve some dependencies from GitHub via the "git" protocol. This requires outgoing access to github.com, port 9418. If the machine on which you're running the build has access restrictions in place for that port but outgoing access via HTTPS (port 443) is allowed, you can substitute the https protocol by running (with the same user account that will run the maven step):

```
git config --global url."https://github.com/".insteadOf git://github.com/
```

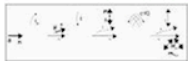
For more information on this issue, see [DS-2428-Mirage 2 build problem - timeout errorClosed](#)

Configuration options

Mirage 2 adds two configuration options to `dspace.cfg` that affect the rendering of bitstream labels on item pages

:

Abbildung.wm



View/Open

PDF with a black and white vector diagram (47.12Kb)

Date

2011-10-05

Author

Landgraf, Werner

Metadata

Show full item record

As representações estáticas e dinâmicas das dimensões e relações entre elas como aspecto geométrico e físico dum desdobramento cada vez a uma nova direção não representável pelos já ocorridos no conjunto causal inicial : Eventos e suas Ações (discreto); Tempo e Energia; Estensão cinemática e Impulso; Massa gravitacional ou raios da curvatura e Crescimento do Plano Tangencial ou do superfície

URI

<http://hdl.handle.net/123456789/1646>

<https://commons.wikimedia.org/wiki/File:Abbildung.wm.pdf>

Collections

PDF Collection

As an administrator, you can choose between displaying the file name (title) or the description (label). Because bitstream description is an optional value, you can also define a fallback value. The default configuration will use the label as the first choice, and fall back to the title field.

```
### Settings for the Item page in Mirage2 theme ###
# Whether the title or the label of a file should be used to display it on the item page
mirage2.item-view.bitstream.href.label.1 = label
# Whether the title or the label of a file should be used as a fallback to display it on the item page
mirage2.item-view.bitstream.href.label.2 = title
```

There are other configuration properties that affect the theme. These aren't new but we mention them here for the sake of completeness.

- When METSRIGHT is included in `plugin.named.org.dspace.containt.crosswalk.DisseminationCrosswalk` the item page will display those rights.
- The property `xmlui.theme.mirage.item-list.emphasis` defines the style of the item lists. When the value is 'file' another style is used.
- The property `webui.browse.render-scientific-formulas` includes a javascript library to render scientific formulas.
- The properties `thumbnail.maxheight` and `thumbnail.maxwidth` define the outer bounds for the dimensions of the item thumbnails in the item lists.

Customizing Mirage 2



Do not attempt the following

Do not attempt to manage local customizations to Mirage 2 in:

- `[dspace-source]/dspace/modules/xmlui/src/main/webapp/themes`
 - This is where you would put standard XMLUI Themes or customizations. However, because of the Mirage 2 build process, this won't work.
- `[dspace-source]/dspace-xmlui-mirage2`
 - This is the place where the community, committers and contributors manage the STANDARD version of Mirage 2. You could change files there if your intention is to create a contribution that would benefit everyone. But in this case, we are not talking about a local customization.



Recommended approach

Manage your local Mirage 2 customizations or derived themes in:

```
[dspace-source]/dspace/modules/xmlui-mirage2
```

Managing your local customizations in this folder comes with the advantage that you ONLY need to keep files you have changed, compared to the standard Mirage 2 folder. To get you started, the contributors have added a `_styles.scss` file where you can make local scss customizations:

```
dspace/modules/xmlui-mirage2/src/main/webapp/themes/Mirage2/styles
```

The Mirage 2 color scheme

The style sheet of Mirage 2 is written in sass and relies on the bootstrap framework. A big advantages of this is the ease of changing the color scheme. By default Mirage 2 has the colors that are familiar from the classic Mirage theme, but another color scheme with only the standard bootstrap colors is also ready and available. In fact this color scheme can be activated by building DSpace using one extra maven profile, `mirage2_bootstrap_color_scheme`.

The classic mirage theme is a customization of the bootstrap theme. Thanks to the sass variables, a complete color scheme can be conceived by modifying one or two variables. These variables are set in the theme's `/styles/classic_mirage_color_scheme/_bootstrap_variables.scss`. Copy this file into `[dspace-source]/dspace/modules/xmlui-mirage2/src/main/webapp/themes/Mirage2` and see what happens when you change `$brand-primary`. More detailed information on how to customize this file can be found [in the Mirage 2 readme](#).

How to reuse an existing bootstrap theme is also explained in that section.

Simple styling customization

Simple customizations imply that they only require custom css, e.g. changing the font or the logo. The theme's `_styles.scss` file is the right place for this. All the lines of css in this file will be included in the theme's style sheet. Even though it's a file with the scss extension, the usual lines of css will work just as well.

Advanced styling customizations

For guidelines on how to include javascript, please read the [the Additional Developer documentation](#).

Automatically retrieving the latest versions of Mirage 2 dependencies

Mirage 2 dependencies are automatically pulled in during the Bower step of the build process. For official DSpace releases, the committers lock the dependencies on a specific version in order to make the behaviour of the theme predictable.

For development purposes however, it is recommended that set the dependencies to "latest" so you can benefit from the most recent updates and bugfixes in Mirage 2's dependencies.

You can make these changes in the bower.json file: [bower.json](#)

As mentioned in the previous section, make sure you manage this file and any changes you make to it in `[dspace-source]/dspace/modules` (e.g. `[dspace-source]/dspace/modules/xmlui-mirage2/src/main/webapp/themes/Mirage2`). It is not recommended to update the officially distributed bower.json file directly in `[dspace-source]/dspace-xmlui-mirage2`

Additional Developer documentation

Specific guidelines and technical details about Mirage 2 are part of the [Readme.MD file in the Mirage 2 sourcetree](#).

Mirage Configuration and Customization

- [1 Introduction](#)
- [2 Configuration Parameters](#)
- [3 Technical Features](#)
 - [3.1 Look & Feel](#)
 - [3.2 Structural enhancements for easier customization.](#)
 - [3.3 Enhanced Performance](#)
- [4 Troubleshooting](#)
 - [4.1 Errors using HTTPS](#)

Introduction

Mirage is a new XMLUI theme, added in DSpace 1.7 by [@mire](#). The code was mainly developed by Art Lowel. The main benefits of Mirage are:

- Clean new look and feel.
- Increased browser compatibility. The whole theme renders perfectly in today's modern browsers (Internet Explorer 7 and higher, Firefox, Safari, Chrome, ...)
- Easier to customize.
- Enhanced Performance

Configuration Parameters

Property:	xmlui.theme.mirage.item-list.emphasis
Example Value:	xmlui.theme.mirage.item-list.emphasis = metadata
Informational Note:	<p>Determines which style should be used to display item lists. Allowed values:</p> <ul style="list-style-type: none"> • metadata: includes item abstracts in the listing and is suited for scientific articles. • file: immediately shows you whether files are attached to the items, by displaying a large thumbnail icon for each of the items. metadata is the default value.
Property:	xmlui.theme.enableConcatenation
Example Value:	xmlui.theme.enableConcatenation = false
Informational Note:	<p>Allows to enable concatenation for .js and .css files. Enhances performance when enabled by lowering the number of files that needs to be sent to the client per page request (as multiple files will be concatenated together and sent as one file). Value can be true or false. False by default.</p>
Property:	xmlui.theme.enableMinification
Example Value:	xmlui.theme.enableMinification = false
Informational Note:	<p>Allows to enable minification for .js and .css files. Enhances performance when enabled by removing unnecessary whitespaces and other characters, thus reducing the size of files to be sent. Value can be true or false. False by default.</p>

Technical Features

Look & Feel

- The Simple Item Display underwent a full redesign to provide visitors with a clearer overview of available metadata and associated files.

- **Item list views** can now be displayed in two distinct different styles. Switching between these styles is possible with the new `dspace.cfg` parameter `'xmlui.theme.mirage.item-list.emphasis'`
 - The **'metadata'** list style includes item abstracts in the listing and is suited for scientific articles.
 - The **'file'** list style immediately shows you whether files are attached to the items, by displaying a large thumbnail icon for each of the items.

Structural enhancements for easier customization.

- **Based on the new restructured dri2xhtml base templates.** Templates in the theme, overriding the new base templates, are located in the same folder hierarchy to ensure full transparency.
- **Automated browser feature detection** for improved browser compatibility.
 - In other themes, user agent detection is used to identify which browser version your user is using. Based on the result of this detection, the theme would use a different cascaded style sheet (CSS) to render a compatible page for the visitor. This approach has 2 major issues:
 - User agent detection isn't very reliable
 - Maintaining these different CSS files is a maintenance nightmare for developers, especially when using features from newer browsers.
 - Mirage applies two novel techniques to resolve these issues
 - For compatibility with older Internet Explorer browsers, [conditional comments give the body tag a class corresponding to the version of IE](#)
 - [modernizr](#) is used to detect which css features are available in the user's browser. This way you can target all browsers that support a certain feature using css classes, and rules affecting the same element can be put together in the same place for all browsers.
- **CSS files are now split up according to function** instead of browser. **style.css** will now fit most needs for customization. Following additional CSS files are included, but will rarely need to be changed:
 - **reset.css** ensures that browser-specific initializations are being reset.
 - **base.css** contains a few base styles
 - **helper.css** contains helper classes to deal with specific functionality.
 - **handheld.css and print.css** enable you to define styles for handheld devices and printing of pages.
- **jQuery and jQueryUI are included by default.** To avoid conflicts the authority control javascript has been rewritten to use jQuery instead of Prototype and Script.aculo.us.

Enhanced Performance

- **Concatenation and Minification techniques** for css and js files.
 - The `IncludePageMeta` has been extended to generate URL's to the concatenated version of all css files using the same media tag.
 - The `ConcatenationReader` has been created to return concatenated and minified versions of the css and js files.
 - Once js and css files have been minified and concatenated, they are being properly cached. As a result, the minification and concatenation operations only need to happen once, and do not include performance overhead.

- Caution: when minification is enabled, all code-comments will be removed. This could be a problem for comments containing copyright notices, so for files with those comments you should disable minification by adding '?nominify' after the url e.g.
`<map:parameter name="javascript" value="lib/js/jquery-ui-1.8.5.custom.min.js?nominify"/>`
 - Disabled by default, these features need to be enabled in the configuration using the properties 'xmlui.theme.enableConcatenation' and 'xmlui.theme.enableMinification'
 - These features can be enabled for other themes as well, but will require an alteration of the theme's sitemap.
- Javascript references are included at the bottom of the page instead of the top. This optimizes page load times in general.

Troubleshooting

Errors using HTTPS

DSpace 1.7.0 ships with a hardcoded `http://` link for JQuery, causing problems for users running 1.7.0 Mirage on HTTPS. While awaiting the implementation of this fix in an upcoming release, you can solve in the following file: **lib/core/page-structure.xsl**, **addJavascript** template. In this file, you will need to **replace**

```
<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js">&#160;</script>
```

with

```
<script type="text/javascript">
  <xsl:text disable-output-escaping="yes">var JsHost = (("https:" ==
document.location.protocol) ? "https://" : "http://");
  document.write(unescape("%3Cscript src='" + JsHost + "ajax.googleapis.com/ajax/libs/
jquery/1.4.2/jquery.min.js' type='text/javascript'%3E%3C/script%3E"));</xsl:text>
</script>
```

Thanks Peter Dietz for providing this fix. Note: This issue is resolved in 1.7.1

XMLUI Base Theme Templates (dri2xhtml)



Two options for base templates to use

There are two main base templates you can use when creating an XMLUI Theme:

- [dri2xhtml](#) - used in the generation of default Reference, Classic and Kubrick themes
- [dri2xhtml-alt](#) - used in the generation of default Mirage theme

You only should use **one** of these two templates, based on which seems easier to you.

- 1 [dri2xhtml](#)
 - 1.1 [Template Structure](#)
- 2 [dri2xhtml-alt](#)
 - 2.1 [Configuration and Installation](#)
 - 2.2 [Features](#)
 - 2.3 [Template Structure](#)

dri2xhtml

The dri2xhtml base template is the original template for creating XMLUI themes. It attempts to provide generic XSLT templates which are then applied across the entire DSpace site, thus making it easier to make site-wide changes.

The dri2xhtml base template is used in the following Themes:

- Reference - the default XMLUI theme
- Classic - an XMLUI theme which looks similar to JSPUI
- Kubrick

Template Structure

The dri2xhtml base template consists of five main XSLTs:

- `dri2xhtml/structural.xsl` - this XSLT is in charge of creating the main layout/page structure of every page within DSpace
- `dri2xhtml/General-Handler.xsl` - this XSLT is in charge of displaying File download links throughout DSpace (it matches the METS `<fileSec>` element).
- `dri2xhtml/DIM-Handler.xsl` - this XSLT is in charge of displaying all DIM (DSpace Intermediate Metadata) metadata throughout DSpace (it matches any DIM metadata in the METS). By default, this is the template used to display all metadata.
- `dri2xhtml/MODS-Handler.xsl` - this XSLT is in charge of displaying all MODS metadata throughout DSpace (it matches any MODS metadata in the METS). By default, this template is not used, as MODS metadata is not generated by XMLUI by default.
- `dri2xhtml/QDC-Handler.xsl` - this XSLT is in charge of displaying all Qualified Dublin Core (QDC) metadata throughout DSpace (it matches any QDC metadata in the METS). By default, this template is not used, as QDC metadata is not generated by XMLUI by default.

dri2xhtml-alt

The dri2xhtml-alt base template is an alternative template for creating XMLUI themes. It contains the same XSLT templates from dri2xhtml, but they are divided into multiple files and folders. Each file attempts to group XSLT templates together based on their function, in order to make it easier to find the templates related to the feature you're trying to modify.

The dri2xhtml-alt base template is used in the following Themes:

- [Mirage](#)

Configuration and Installation

The alternative basic templates is called "dri2xhtml-alt".

Any of the existing themes can be updated to reference this new set of templates by replacing in your theme.xsl:

```
<xsl:stylesheet xmlns:i18n="http://apache.org/cocoon/i18n/2.1"
  xmlns:dri="http://di.tamu.edu/DRI/1.0/"
  xmlns:mets="http://www.loc.gov/METS/"
  xmlns:xlink="http://www.w3.org/TR/xlink/"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
  xmlns:dim="http://www.dspace.org/xmlns/dspace/dim"
  xmlns:xhtml="http://www.w3.org/1999/xhtml"
  xmlns:mods="http://www.loc.gov/mods/v3"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns="http://www.w3.org/1999/xhtml"
  exclude-result-prefixes="i18n dri mets xlink xsl dim xhtml mods dc">
  <!--
    comment out original dri2xhtml
    <xsl:import href="../dri2xhtml.xsl"/>
    and enable dri2xhtml-alt
  -->
  <xsl:import href="../dri2xhtml-alt/dri2xhtml.xsl"/>
  <xsl:output indent="yes"/>
```

Because the contents of dri2xhtml-alt is identical to the current dri2xhtml.xsl and its derivatives, updating any of the existing themes to reference the new dri2xhtml-alt should not impose any changes in the rendering of the pages.

Features

- No changes to existing templates found in legacy dri2xhtml
- Drops inclusion of Handlers other than DIM and Default
- Templates divided out into files so they can be more easily located, divided by Aspect, Page and Functionality

Template Structure

```
/dspace-xmlui/dspace-xmlui-webapp/src/main/webapp/themes/dri2xhtml-alt/
aspect
  administrative
    harvesting.xsl
  artifactbrowser
    COinS.xsl
    ORE.xsl
    artifactbrowser.xsl
  collection-list.xsl
  collection-view.xsl
```

```
common.xml
community-list.xml
community-view.xml
item-list.xml
item-view.xml
general
  choice-authority-control.xml
core
  attribute-handlers.xml
  elements.xml
  forms.xml
  global-variables.xml
  navigation.xml
  page-structure.xml
  utils.xml
dri2xhtml.xml
```

DRI Schema Reference

Digital Repository Interface (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 reference document explains the purpose of DRI, provides a broad architectural overview, and explains common design patterns. The appendix includes a complete reference for elements used in the DRI Schema, a graphical representation of the element hierarchy, and a quick reference table of elements and attributes.

Table of Contents:

- 1 [Introduction](#)
 - 1.1 [The Purpose of DRI](#)
 - 1.2 [The Development of DRI](#)
- 2 [DRI in Manakin](#)
 - 2.1 [Themes](#)
 - 2.2 [Aspect Chains](#)
- 3 [Common Design Patterns](#)
 - 3.1 [Localization and Internationalization](#)
 - 3.2 [Standard attribute triplet](#)
 - 3.3 [Structure-oriented markup](#)
- 4 [Schema Overview](#)
- 5 [Merging of DRI Documents](#)
- 6 [Version Changes](#)
 - 6.1 [Changes from 1.0 to 1.1](#)
- 7 [Element Reference](#)
 - 7.1 [BODY](#)
 - 7.2 [cell](#)
 - 7.3 [div](#)

- [7.4 DOCUMENT](#)
- [7.5 field](#)
- [7.6 figure](#)
- [7.7 head](#)
- [7.8 help](#)
- [7.9 hi](#)
- [7.10 instance](#)
- [7.11 item](#)
- [7.12 label](#)
- [7.13 list](#)
- [7.14 META](#)
- [7.15 metadata](#)
- [7.16 OPTIONS](#)
- [7.17 p](#)
- [7.18 pageMeta](#)
- [7.19 params](#)
- [7.20 reference](#)
- [7.21 referenceSet](#)
- [7.22 repository](#)
- [7.23 repositoryMeta](#)
- [7.24 row](#)
- [7.25 table](#)
- [7.26 trail](#)
- [7.27 userMeta](#)
- [7.28 value](#)
- [7.29 xref](#)

Introduction

This manual describes the Digital Repository Interface (DRI) as it applies to the DSpace digital repository and XMLUI Manakin based interface. DSpace XML UI is a comprehensive user interface system. It is centralized and generic, allowing it to be applied to all DSpace pages, effectively replacing the JSP-based interface system. Its ability to apply specific styles to arbitrarily large sets of DSpace pages significantly eases the task of adapting the DSpace look and feel to that of the adopting institution. This also allows for several levels of branding, lending institutional credibility to the repository and collections.

Manakin, the second version of DSpace XML UI, consists of several components, written using Java, XML, and XSL, and is implemented in [Cocoon](#). Central to the interface is the XML Document, which is a semantic representation of a DSpace page. In Manakin, the XML Document adheres to a schema called the Digital Repository Interface (DRI) Schema, which was developed in conjunction with Manakin and is the subject of this guide. For the remainder of this guide, the terms XML Document, DRI Document, and Document will be used interchangeably.

This reference document explains the purpose of DRI, provides a broad architectural overview, and explains common design patterns. The appendix includes a complete reference for elements used in the DRI Schema, a graphical representation of the element hierarchy, and a quick reference table of elements and attributes.

The Purpose of DRI

DRI is a schema that governs the structure of the XML Document. It determines the elements that can be present in the Document and the relationship of those elements to each other. Since all Manakin components produce XML Documents that adhere to the DRI schema, The XML Document serves as the abstraction layer. Two such components, Themes and Aspects, are essential to the workings of Manakin and are described briefly in this manual.

The Development of DRI

The DRI schema was developed for use in Manakin. The choice to develop our own schema rather than adapt an existing one came after a careful analysis of the schema's purpose as well as the lessons learned from earlier attempts at customizing the DSpace interface. Since every DSpace page in Manakin exists as an XML Document at some point in the process, the schema describing that Document had to be able to structurally represent all content, metadata and relationships between different parts of a DSpace page. It had to be precise enough to avoid losing any structural information, and yet generic enough to allow Themes a certain degree of freedom in expressing that information in a readable format.

Popular schemas such as XHTML suffer from the problem of not relating elements together explicitly. For example, if a heading precedes a paragraph, the heading is related to the paragraph not because it is encoded as such but because it happens to precede it. When these structures are attempted to be translated into formats where these types of relationships are explicit, the translation becomes tedious, and potentially problematic. More structured schemas, like TEI or DocBook, are domain specific (much like DRI itself) and therefore not suitable for our purposes.

We also decided that the schema should natively support a metadata standard for encoding artifacts. Rather than encoding artifact metadata in structural elements, like tables or lists, the schema would include artifacts as objects encoded in a particular standard. The inclusion of metadata in native format would enable the Theme to choose the best method to render the artifact for display without being tied to a particular structure.

Ultimately, we chose to develop our own schema. We have constructed the DRI schema by incorporating other standards when appropriate, such as [Cocoon's i18n schema](#) for internationalization, [DCMI's Dublin Core](#), and the [Library of Congress's METS schema](#). The design of structural elements was derived primarily from [TEI](#), 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 [XMLUI Configuration and Customization](#) section for more information on installing and configuring themes). Themes may be configured to apply to all pages of specific type, like browse-by-title, to all pages of a one particular community or collection or sets of communities and collections, and to any mix of the two. They can also be configured to apply to a single arbitrary page or handle.

Aspect Chains

Manakin Aspects are arrangements of Cocoon components (transformers, actions, matchers, etc) that implement a new set of coupled features for the system. These Aspects are chained together to form all the features of Manakin. Five Aspects exist in the default installation of Manakin, each handling a particular set of features of DSpace, and more can be added to implement extra features. All Aspects take a DRI Document as input and generate one as output. This allows Aspects to be linked together to form an Aspect chain. Each Aspect in the chain takes a DRI Document as input, adds its own functionality, and passes the modified Document to the next Aspect in the chain.

Common Design Patterns

There are several design patterns used consistently within the DRI schema. This section identifies the need for and describes the implementation of these patterns. Three patterns are discussed: language and internationalization issues, standard attribute triplet (*id*, *n*, and *rend*), and the use of structure-oriented markup.

Localization and Internationalization

Internationalization is a very important component of the DRI system. It allows content to be offered in other languages based on user's locale and conditioned upon availability of translations, as well as present dates and currency in a localized manner. There are two types of translated content: content stored and displayed by DSpace itself, and content introduced by the DRI styling process in the XSL transformations. Both types are handled by Cocoon's *i18n* transformer without regard to their origin.

When the Content Generation process produces a DRI Document, some of the textual content may be marked up with *i18n* elements to signify that translations are available for that content. During the Style Application

process, the Theme can also introduce new textual content, marking it up with *i18n* tags. As a result, after the Theme's XSL templates are applied to the DRI Document, the final output consists of a DSpace page marked up in the chosen display format (like XHTML) with *i18n* elements from both DSpace and XSL content. This final document is sent through Cocoon's i18n transformer that translates the marked up text.

Standard attribute triplet

Many elements in the DRI system (all top-level containers, character classes, and many others) contain one or several of the three standard attributes: *id*, *n*, and *rend*. The *id* and *n* attributes can be required or optional based on the element's purpose, while the *rend* attribute is always optional. The first two are used for identification purposes, while the third is used as a display hint issued to the styling step.

Identification is important because it allows elements to be separated from their peers for sorting, special case rendering, and other tasks. The first attribute, *id*, is the global identifier and it is unique to the entire document. Any element that contains an *id* attribute can thus be uniquely referenced by it. The *id* attribute of an element can be either assigned explicitly, or generated from the Java Class Path of the originating object if no name is given. While all elements that can be uniquely identified can carry the *id* attribute, only those that are independent on their context are required to do so. For example, tables are required to have an *id* since they retain meaning regardless of their location in the document, while table rows and cells can omit the attribute since their meaning depends on the parent element.

The name attribute *n* is simply the name assigned to the element, and it is used to distinguish an element from its immediate peers. In the example of a particular list, all items in that list will have different names to distinguish them from each other. Other lists in the document, however, can also contain items whose names will be different from each other, but identical to those in the first list. The *n* attribute of an element is therefore unique only in the scope of that element's parent and is used mostly for sorting purposes and special rendering of a certain class of elements, like, for example, all first items in lists, or all items named "browse". The *n* attribute follows the same rules as *id* when determining whether or not it is required for a given element.

The last attribute in the standard triplet is *rend*. Unlike *id* and *n*, the *rend* attribute can consist of several space delimited values and is optional for all elements that can contain it. Its purpose is to provide a rendering hint from the middle layer component to the styling theme. How that hint is interpreted and whether it is used at all when provided, is completely up to the theme. There are several cases, however, where the content of the *rend* attribute is outlined in detail and its use is encouraged. Those cases are the emphasis element *h1*, the division element *div*, and the *list* element. Please refer to the Element Reference for more detail on these elements.

Structure-oriented markup

The final design pattern is the use of structure-oriented markup for content carried by the XML Document. Once generated by Cocoon, the Document contains two major types of information: metadata about the repository and its contents, and the actual content of the page to be displayed. A complete overview of metadata and content markup and their relationship to each other is given in the next section. An important thing to note here, however, is that the markup of the content is oriented towards explicitly stating structural relationships between the elements rather than focusing on the presentational aspects. This makes the markup used by the Document more similar to TEI or Docbook rather than HTML. For this reason, XSL templates are used by the themes to convert structural DRI markup to XHTML. Even then, an attempt is made to create XHTML as structural as

possible, leaving presentation entirely to CSS. This allows the XML Document to be generic enough to represent any DSpace page without dictating how it should be rendered.

Schema Overview

The DRI XML Document consists of the root element `document` and three top-level elements that contain two major types of elements. The three top-level containers are *meta*, *body*, and *options*. The two types of elements they contain are metadata and content, carrying metadata about the page and the contents of the page, respectively. Figure 1 depicts the relationship between these six components.

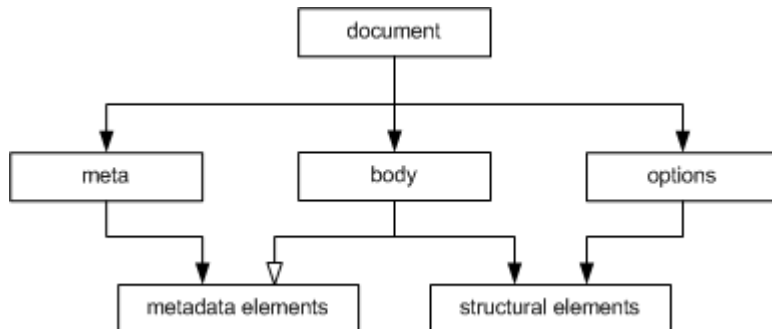


Figure 1: The two content types across three major divisions of a DRI page.

The *document* element is the root for all DRI pages and contains all other elements. It bears only one attribute, *version*, that contains the version number of the DRI system and the schema used to validate the produced document. At the time of writing the working version number is "1.1".

The *meta* element is the top-level element under *document* and contains all metadata information about the page, the user that requested it, and the repository it is used with. It contains no structural elements, instead being the only container of metadata elements in a DRI Document. The metadata stored by the *meta* element is broken up into three major groups: *userMeta*, *pageMeta*, and *objectMeta*, each storing metadata information about their respective component. Please refer to the reference entries for more information about these elements.

The *options* element is another top-level element that contains all navigation and action options available to the user. The options are stored as items in list elements, broken up by the type of action they perform. The five types of actions are: browsing, search, language selection, actions that are always available, and actions that are context dependent. The two action types also contain sub-lists that contain actions available to users of varying degrees of access to the system. The *options* element contains no metadata elements and can only make use of a small set of structural elements, namely the *list* element and its children.

The last major top-level element is the *body* element. It contains all structural elements in a DRI Document, including the lists used by the *options* element. Structural elements are used to build a generic representation of a DSpace page. Any DSpace page can be represented with a combination of the structural elements, which will in turn be transformed by the XSL templates into another format. This is the core mechanism that allows DSpace XML UI to apply uniform templates and styling rules to all DSpace pages and is the fundamental difference from the JSP approach currently used by DSpace.

The *body* element directly contains only one type of element: *div*. The *div* element serves as a major division of content and any number of them can be contained by the *body*. Additionally, divisions are recursive, allowing *divs* to contain other *divs*. It is within these elements that all other structural elements are contained. Those elements include tables, paragraph elements *p*, and lists, as well as their various children elements. At the lower levels of this hierarchy lie the character container elements. These elements, namely paragraphs *p*, table *cells*, lists *items*, and the emphasis element *hi*, contain the textual content of a DSpace page, optionally modified with links, figures, and emphasis. If the division within which the character class is contained is tagged as interactive (via the *interactive* attribute), those elements can also contain interactive form fields. Divisions tagged as interactive must also provide *method* and *action* attributes for its fields to use.

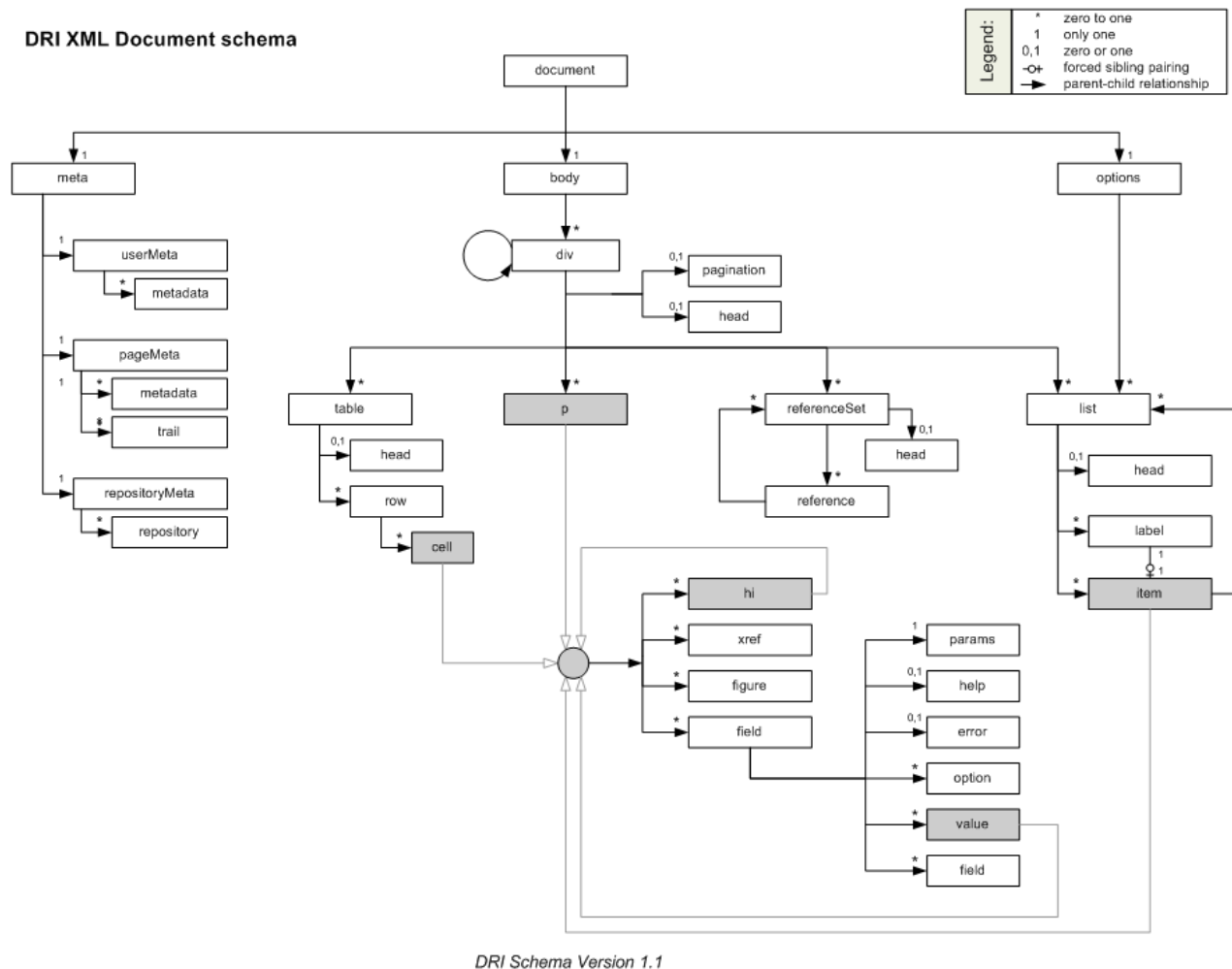


Figure 2: All the elements in the DRI schema (version 1.1).

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*. Originally all metadata for objects were included in-line with the DRI document, this proved to have several problems and has been removed in version 1.1 of the DRI schema. Instead of including metadata in-line, external references to the metadata is included. Thus, a *reference* element has been added along with *referenceSet*. These new elements operate like their counterparts in the previous version except referencing metadata contained on the *objectMeta* element they reference metadata in external files. The *repository* and *repositoryMeta* elements were also modified in a similar manner removing in-line metadata and referencing external metadata documents.

Element Reference

Element	Attributes	Required?
BODY		
cell		
	cols	
	id	
	n	
	rend	
	role	
	rows	
div		
	action	required for interactive behavior
	behaviorSensitivFields	
	currentPage	
	firstItemIndex	
	id	required
	interactive	
	itemsTotal	
	lastItemIndex	
	method	required for interactive
	n	required
	nextPage	
	pagesTotal	
	pageURLMask	
	pagination	
	previousPage	

Element	Attributes	Required?
	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		

Element	Attributes	Required?
	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	

Element	Attributes	Required?
	size	
reference		
	url	required
	repositoryID	required
	type	
referenceSet		
	id	required
	n	required
	orderBy	
	rend	
	type	required
repository		
	repositoryID	required
	url	required
repositoryMeta		
row		
	id	
	n	
	rend	
	role	required
table		
	cols	required
	id	required
	n	required
	rend	

Element	Attributes	Required?
	rows	required
trail		
	rend	
	target	
userMeta	authenticated	required
value		
	optionSelected	
	optionValue	
	type	required
xref	target	required

BODY

Top-Level Container

The *body* element is the main container for all content displayed to the user. It contains any number of *div* elements that group content into interactive and display blocks.

Parent

- document

Children

- div (any)

Attributes

- None

```

<document version=1.0>
  <meta> ... </meta>
  <body>
    <div n="division-example1"
      id="XMLExample.div.division-example1">
      ...
    </div>
    <div n="division-example2" id="XMLExample.div.division-example2"
      interactive="yes" action="www.DRItest.com"
      method="post">

```

```
...
</div>
...
</body>
<options> ... </options>
</document>
```

cell

Rich Text Container

Structural Element

The *cell* element contained in a *row* of a *table* carries content for that table. It is a character container, just like *p*, *item*, and *hi*, and its primary purpose is to display textual data, possibly enhanced with hyperlinks, emphasized blocks of text, images and form fields. Every *cell* can be annotated with a *role* (the most common being "header" and "data") and can stretch across any number of rows and columns. Since cells cannot exist outside their container, *row*, their *id* attribute is optional.

Parent

- row

Children

- hi (any)
- xref (any)
- figure (any)
- field (any)

Attributes

- **cols**: (optional) The number of columns the cell spans.
- **id**: (optional) A unique identifier of the element.
- **n**: (optional) A local identifier used to differentiate the element from its siblings.
- **rend**: (optional) A rendering hint used to override the default display of the element.
- **role**: (optional) An optional attribute to override the containing row's role settings.
- **rows**: (optional) The number of rows the cell spans.

```
<table n="table-example" id="XMLExample.table.table-example" rows="2"
  cols="3">
  <row role="head">
    <cell cols="2">Data Label One and Two</cell> <cell>Data Label
    Three</cell>
    ...
  </row>
  <row>
    <cell> Value One </cell> <cell> Value Two </cell> <cell> Value
```

```
    Three </cell>
    ...
</row>
...
</table>
```

div

Structural Element

The *div* element represents a major section of content and can contain a wide variety of structural elements to present that content to the user. It can contain paragraphs, tables, and lists, as well as references to artifact information stored in *artifactMeta*, *repositoryMeta*, *collections*, and *communities*. The *div* element is also recursive, allowing it to be further divided into other divs. Divs can be of two types: interactive and static. The two types are set by the use of the *interactive* attribute and differ in their ability to contain interactive content. Children elements of divs tagged as interactive can contain form fields, with the *action* and *method* attributes of the *div* serving to resolve those fields.

Parent

- body
- div

Children

- head (zero or one)
- pagination (zero or one)
- table (any)
- p (any)
- referenceSet (any)
- list (any)
- div (any)

Attributes

- **action**: (required for interactive) The form action attribute determines where the form information should be sent for processing.
- **behavior**: (optional for interactive) The acceptable behavior options that may be used on this form. The only possible value defined at this time is "ajax" which means that the form may be submitted multiple times for each individual field in this form. Note that if the form is submitted multiple times it is best for the behaviorSensitiveFields to be updated as well.
- **behaviorSensitiveFields**: (optional for interactive) A space separated list of field names that are sensitive to behavior. These fields must be updated each time a form is submitted with out a complete refresh of the page (i.e. ajax).
- **currentPage**: (optional) For paginated divs, the currentPage attribute indicates the index of the page currently displayed for this div.

- **firstItemIndex**: (optional) For paginated divs, the firstItemIndex attribute indicates the index of the first item included in this div.
- **id**: (required) A unique identifier of the element.
- **interactive**: (optional) Accepted values are "yes", "no". This attribute determines whether the div is interactive or static. Interactive divs must provide action and method and can contain field elements.
- **itemsTotal**: (optional) For paginated divs, the itemsTotal attribute indicates how many items exist across all paginated divs.
- **lastItemIndex**: (optional) For paginated divs, the lastItemIndex attribute indicates the index of the last item included in this div.
- **method**: (required for interactive) Accepted values are "get", "post", and "multipart". Determines the method used to pass gathered field values to the handler specified by the action attribute. The multipart method should be used for uploading files.
- **n**: (required) A local identifier used to differentiate the element from its siblings.
- **nextPage**: (optional) For paginated divs the nextPage attribute points to the URL of the next page of the div, if it exists.
- **pagesTotal**: (optional) For paginated divs, the pagesTotal attribute indicates how many pages the paginated divs spans.
- **pageURLMask**: (optional) For paginated divs, the pageURLMask attribute contains the mask of a url to a particular page within the paginated set. The destination page's number should replace the {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 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" or "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 possess multiple combined values.

```
<p>
  <hi> ... </hi>
  <xref> ... </xref>
  <figure> ... </figure>
  ...
  <field id="XMLExample.field.name" n="name" type="text"
    required="yes">
    <params size="16" maxlength="32"/>
    <help>Some help text with <i18n>localized
    content</i18n>.</help>
    <value type="raw">Default value goes
    here</value>
  </field>
</p>
```

figure

Text Container

Structural Element

The *figure* element is used to embed a reference to an image or a graphic element. It can be mixed freely with text, and any text within the tag itself will be used as an alternative descriptor or a caption.

Parent

- cell
- p
- hi
- item

Children

- none

Attributes

- **rend**: (optional) A rendering hint used to override the default display of the element.

- **source:** (optional) The source for the image, using either a URL or a pre-defined XML entity.
- **target:** (optional) A target for an image used as a link, using either a URL or an id of an existing element as a destination.

```
<p>
  <hi> ... </hi>
  ...
  <xref> ... </xref>
  ...
  <field> ... </field>
  ...
  <figure source="www.example.com/fig1"> This is a static image.
</figure> <figure source="www.example.com/fig1"
target="www.example.net">
  This image is also a link.
  </figure>
  ...
</p>
```

head

Text Container

Structural Element

The *head* element is primarily used as a label associated with its parent element. The rendering is determined by its parent tag, but can be overridden by the *rend* attribute. Since there can only be one *head* element associated with a particular tag, the *n* attribute is not needed, and the *id* attribute is optional.

Parent

- div
- table
- list
- referenceSet

Children

- none

Attributes

- **id:** (optional) A unique identifier of the element
- **n:** (optional) A local identifier used to differentiate the element from its siblings
- **rend:** (optional) A rendering hint used to override the default display of the element.

```
<div ...>
  <head> This is a simple header associated with its div element.
```

```

</head>
<div ...>
  <head rend="green"> This header will be green.
</head>
  <p>
    <head> A header with <i18n>localized content</i18n>.
  </head>
  ...
</p>
</div>
<table ...>
  <head> ...
</head>
  ...
</table>
<list ...>
  <head> ...
</head>
  ...
</list>
  ...
</body>

```

help

Text Container

Structural Element

The optional *help* element is used to supply help instructions in plain text and is normally contained by the *field* element. The method used to render the help text in the target markup is up to the theme.

Parent

- field

Children

- none

Attributes

- None

```

<p>
  <hi> ... </hi>
  ...
  <xref> ... </xref>
  ...
  <figure> ... </figure>
  ...

```

```
<field id="XMLExample.field.name" n="name" type="text"
required="yes">
  <params size="16" maxlength="32" />
  <help>Some help text with <i18n>localized
content</i18n>.</help>
</field>
...
</p>
```

hi

Rich Text Container

Structural Element

The *hi* element is used for emphasis of text and occurs inside character containers like *p* and *listitem*. It can be mixed freely with text, and any text within the tag itself will be emphasized in a manner specified by the required *rend* attribute. Additionally, *hi* element is the only text container component that is a rich text container itself, meaning it can contain other tags in addition to plain text. This allows it to contain other text containers, including other *hi* tags.

Parent

- cell
- p
- item
- hi

Children

- hi (any)
- xref (any)
- figure (any)
- field (any)

Attributes

- **rend:** (required) A required attribute used to specify the exact type of emphasis to apply to the contained text. Common values include but are not limited to "bold", "italic", "underline", and "emph".

```
<p>
This text is normal, while <hi rend="bold">this text is bold and
this text is <hi rend="italic">bold and
italic.</hi></hi>
</p>
```

instance

Structural Element

The *instance* element contains the value associated with a form field's multiple instances. Fields encoded as an instance should also include the values of each instance as a hidden field. The hidden field should be appended with the index number for the instance. Thus if the field is "firstName" each instance would be named "firstName_1", "firstName_2", "firstName_3", etc...

Parent

- field

Children

- value

Attributes

- None listed yet.

Example needed.

item

Rich Text Container

Structural Element

The *item* element is a rich text container used to display textual data in a list. As a rich text container it can contain hyperlinks, emphasized blocks of text, images and form fields in addition to plain text.

The *item* element can be associated with a label that directly precedes it. The Schema requires that if one *item* in a *list* has an associated *label*, then all other items must have one as well. This mitigates the problem of loose connections between elements that is commonly encountered in XHTML, since every item in particular list has the same structure.

Parent

- list

Children

- hi (any)
- xref (any)
- figure (any)
- field (any)

- list (any)

Attributes

- **id**: (optional) A unique identifier of the element
- **n**: (optional) A non-unique local identifier used to differentiate the element from its siblings
- **rend**: (optional) A rendering hint used to override the default display of the element.

```

<list n="list-example"
  id="XMLExample.list.list-example">
  <head> Example List </head>
  <item> This is the first item
  </item> <item> This is the second item with <hi ...>highlighted text</hi>,
  <xref ...> a link</xref> and an <figure
  ...>image</figure>.</item>
  ...
<list n="list-example2"
  id="XMLExample.list.list-example2">
  <head> Example List </head>
  <label>ITEM ONE:</label>
  <item> This is the first item
  </item>
  <label>ITEM TWO:</label>
  <item> This is the second item with <hi ...>highlighted
  text</hi>, <xref ...> a link</xref> and an <figure
  ...>image</figure>.</item>
  <label>ITEM THREE:</label>
  <item> This is the third item with a <field ...> ... </field>
  </item>
  ...
</list>
  <item> This is the third item in the list
  </item>
  ...
</list>

```

label

Text Container

Structural Element

The *label* element is associated with an item and annotates that item with a number, a textual description of some sort, or a simple bullet.

Parent

- item

Children

- none

Attributes

- **id**: (optional) A unique identifier of the element
- **n**: (optional) A local identifier used to differentiate the element from its siblings
- **rend**: (optional) An optional rend attribute provides a hint on how the label should be rendered, independent of its type.

```

<list n="list-example"
  id="XMLExample.list.list-example">
  <head>Example List</head>
  <label>1</label>
  <item> This is the first item </item>
  <label>2</label>
  <item> This is the second item with <hi ...>highlighted text</hi>,
    <xref ...> a link</xref> and an <figure
    ...>image</figure>.</item>
  ...
<list n="list-example2"
  id="XMLExample.list.list-example2">
  <head>Example Sublist</head>
  <label>ITEM
  ONE:</label>
  <item> This is the first item </item>
  <label>ITEM
  TWO:</label>
  <item> This is the second item with <hi ...>highlighted
  text</hi>, <xref ...> a link</xref> and an <figure
  ...>image</figure>.</item>
  <label>ITEM
  THREE:</label>
  <item> This is the third item with a <field ...> ... </field>
  </item>
  ...
</list>
<item> This is the third item in the list </item>
  ...
</list>

```

list

Structural Element

The *list* element is used to display sets of sequential data. It contains an optional *head* element, as well as any number of *item* and *list* elements. *Items* contain textual information, while sublists contain other *item* or *list* elements. An *item* can also be associated with a *label* element that annotates an item with a number, a textual description of some sort, or a simple bullet. The list type (ordered, bulleted, gloss, etc.) is then determined either by the content of *labels* on *items* or by an explicit value of the *type* attribute. Note that if *labels* are used in

conjunction with any *items* in a list, all of the *items* in that list must have a *label*. It is also recommended to avoid mixing *label*/styles unless an explicit type is specified.

Parent

- div
- list

Children

- head (zero or one)
- label (any)
- item (any)
- list (any)

Attributes

- **id**: (required) A unique identifier of the element
- **n**: (required) A local identifier used to differentiate the element from its siblings
- **rend**: (optional) An optional rend attribute provides a hint on how the list should be rendered, independent of its type. Common values are but not limited to:
 - **alphabet**: The list should be rendered as an alphabetical index
 - **columns**: The list should be rendered in equal length columns as determined by the theme.
 - **columns2**: The list should be rendered in two equal columns.
 - **columns3**: The list should be rendered in three equal columns.
 - **horizontal**: The list should be rendered horizontally.
 - **numeric**: The list should be rendered as a numeric index.
 - **vertical**: The list should be rendered vertically.
- **type**: (optional) An optional attribute to explicitly specify the type of list. In the absence of this attribute, the type of a list will be inferred from the presence and content of labels on its items. Accepted values are :
 - **form**: Used for form lists that consist of a series of fields.
 - **bulleted**: Used for lists with bullet-marked items.
 - **gloss**: Used for lists consisting of a set of technical terms, each marked with a *label* element and accompanied by the definition marked as an *item* element.
 - **ordered**: Used for lists with numbered or lettered items.
 - **progress**: Used for lists consisting of a set of steps currently being performed to accomplish a task . For this type to apply, each *item* in the list should represent a step and be accompanied by a *label* that contains the displayable name for the step. The *item* contains an *xref* that references the step. Also the *rend* attribute on the *item* element should be: "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 of an attribute-value pair. The type of information it contains is determined by two attributes: *element*, which specifies the general type of metadata stored, and an optional *qualifier* attribute that narrows the type down. The standard representation for this pairing is *element.qualifier*. The actual metadata is contained in the text of the tag itself. Additionally, a *language* attribute can be used to specify the language used for the metadata entry.

Parent

- userMeta
- pageMeta

Children

- none

Attributes

- **element**: (required) The name of a metadata field.
- **language**: (optional) An optional attribute to specify the language used in the metadata tag.
- **qualifier**: (optional) An optional postfix to the field name used to further differentiate the names.

```
<meta>
  <userMeta>
    <metadata element="identifier" qualifier="firstName"> Bob
  </metadata> <metadata element="identifier" qualifier="lastName"> Jones
  </metadata> <metadata ...> ...
  </metadata>
  ...
</userMeta>
<pageMeta>
  <metadata element="rights"
  qualifier="accessRights">user</metadata> <metadata ...> ...
  </metadata>
  ...
</pageMeta>
</meta>
```

OPTIONS

Top-Level Container

The *options* element is the main container for all actions and navigation options available to the user. It consists of any number of *list* elements whose items contain navigation information and actions. While any list of navigational options may be contained in this element, it is suggested that at least the following 5 lists be included.

Parent

- document

Children

- list (any)

Attributes

- None

```
<document version=1.0>
  <meta> ... </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">Example DRI
  page</metadata>
    <metadata
  element="contextPath">/xmlui/</metadata>
    <metadata ...> ... </metadata>
    ...
    <trail source="123456789/6"> A bread crumb item
  </trail>
    <trail ...> ... </trail>
    ...
  </pageMeta>
</meta>

```

params

Structural Component

The *params* element identifies extra parameters used to build a form field. There are several attributes that may be available for this element depending on the field type.

Parent

- field

Children

- none

Attributes

- **cols:** (optional) The default number of columns that the text area should span. This applies only to textarea field types.
- **maxlength:** (optional) The maximum length that the theme should accept for form input. This applies to text and password field types.
- **multiple:** (optional) yes/no value. Determine if the field can accept multiple values for the field. This applies only to select lists.
- **operations:** (optional) The possible operations that may be preformed on this field. The possible values are "add" and/or "delete". If both operations are possible then they should be provided as a space separated list. The "add" operations indicates that there may be multiple values for this field and the user may add to the set one at a time. The front-end should render a button that enables the user to add more fields to the set. The button must be named the field name appended with the string "_add", thus if the field's name is "firstName" the button must be called "firstName_add".The "delete" operation indicates that there may be multiple values for this field each of which may be removed from the set. The front-end should render a checkbox by each field value, except for the first, The checkbox must be named the field name appended with the string "_selected", thus if the field's name is "firstName" the checkbox must be called "firstName_selected" and the value of each successive checkbox should be the field name. The front-end must also render a delete button. The delete button name must be the field's name appended with the string "_delete".
- **rows:** (optional) The default number of rows that the text area should span. This applies only to textarea field types.
- **size:** (optional) The default size for a field. This applies to text, password, and select field types.

```

<p>
<field id="XMLExample.field.name" n="name" type="text"
required="yes">
  <params size="16"
maxlength="32"/>
  <help>Some help text with <i18n>localized
content</i18n>.</help>
  <default>Default value goes here</default>
</field>
</p>

```

reference

Metadata Reference Element

reference is a reference element used to access information stored in an external metadata file. The *url* attribute is used to locate the external metadata file. The *type* attribute provides a short limited description of the referenced object's type.

reference elements can be both contained by *includeSet* elements and contain *includeSets* themselves, making the structure recursive.

Parent

- referenceSet

Children

- referenceSet (zero or more)

Attributes

- **url**: (required) A url to the external metadata file.
- **repositoryIdentifier**: (required) A reference to the repositoryIdentifier of the repository.
- **type**: (optional) Description of the reference object's type.

```

    <includeSet n="browse-list"
    id="XMLTest.includeSet.browse-list">
      <reference url="/metadata/handle/123/4/mets.xml"
    repositoryID="123" type="DSpace
    Item"/> <reference url="/metadata/handle/123/5/mets.xml"
    repositoryID="123" />
      ...
    </includeSet>
  
```

referenceSet

Metadata Reference Element

The *referenceSet* element is a container of artifact or repository references.

Parent

- div
- reference

Children

- head (zero or one)
- reference (any)

Attributes

- **id:** (required) A unique identifier of the element
- **n:** (required) Local identifier used to differentiate the element from its siblings
- **orderBy:** (optional) A reference to the metadata field that determines the ordering of artifacts or repository objects within the set. When the Dublin Core metadata scheme is used this attribute should be the element.qualified value that the set is sorted by. As an example, for a browse by title list, the value should be sortedBy=title, while for browse by date list it should be sortedBy=date.created
- **rend:** (optional) A rendering hint used to override the default display of the element.
- **type:** (required) Determines the level of detail for the given metadata. Accepted values are:
 - **summaryList:** Indicates that the metadata from referenced artifacts or repository objects should be used to build a list representation that is suitable for quick scanning.
 - **summaryView:** Indicates that the metadata from referenced artifacts or repository objects should be used to build a partial view of the referenced object or objects.
 - **detailList:** Indicates that the metadata from referenced artifacts or repository objects should be used to build a list representation that provides a complete, or near complete, view of the referenced objects. Whether such a view is possible or different from summaryView depends largely on the repository at hand and the implementing theme.
 - **detailView:** Indicates that the metadata from referenced artifacts or repository objects should be used to display complete information about the referenced object. Rendering of several references included under this type is up to the theme.

```

<div ...>
<head> Example Division </head>
<p> ... </p>
<table> ... </table>
<list>
...
</list>
  <referenceSet n="browse-list"
id="XMLTest.referenceSet.browse-list" type="summaryView"
informationModel="DSpace">
  <head>A header for the includeset</head>
  <reference
url="/metadata/handle/123/34/mets.xml"/>
  <reference
url=" "metadata/handle/123/34/mets.xml"/>
  </referenceSet>
...
</p>

```

repository

Metadata Element

The *repository* element is used to describe the repository. Its principal component is a set of structural metadata that carrier information on how the repository's objects under *objectMeta* are related to each other. The principal method of encoding these relationships at the time of this writing is a METS document, although other formats, like RDF, may be employed in the future.

Parent

- repositoryMeta

Children

- none

Attributes

- **repositoryID**: requiredA unique identifier assigned to a repository. It is referenced by the *object* element to signify the repository that assigned its identifier.
- **url**: requiredA 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">Example 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 subelements: the raw element which stores the unprocessed value directly from the user or other source, and the interpreted element which stores the value in a format appropriate for display to the user, possibly including rich text markup.

Parent

- field

Children

- hi (any)
- xref (any)
- figure (any)

Attributes

- **optionSelected:** (optional) An optional attribute for select, checkbox, and radio fields to determine if the value is to be selected or not.
- **optionValue:** (optional) An optional attribute for select, checkbox, and radio fields to determine the value that should be returned when this value is selected.
- **type:** (required) A required attribute to specify the type of value. Accepted types are:
 - **raw:** The raw type stores the unprocessed value directly from the user or other source.
 - **interpreted:** The interpreted type stores the value in a format appropriate for display to the user, possibly including rich text markup.
 - **default:** The default type stores a value supplied by the system, used when no other values are provided.

```

<p>
  <hi> ... </hi>
  <xref> ... </xref>
  <figure> ... </figure>
  <field id="XMLExample.field.name" n="name" type="text"
    required="yes">
    <params size="16" maxlength="32"/>
    <help>Some help text with <i18n>localized
    content</i18n>.</help>
    <value type="default">Author,
    John</value>
  </field>
</p>

```

xref

Text Container

Structural Element

The *xref* element is a reference to an external document. It can be mixed freely with text, and any text within the tag itself will be used as part of the link's visual body.

Parent

- cell
- p
- item
- hi

Children

- none

Attributes

- **target:** (required) A target for the reference, using either a URL or an id of an existing element as a destination for the *xref*.

```
<p>  
  <xref target="/url/link/target">This text is shown as a link.</xref>  
</p>
```

5 System Administration

This top level node intends to hold all system administration aspects of DSpace including but not limited to:

- Installation
- Upgrading
- Troubleshooting system errors
- Managing Dependencies

In this context System administration is defined as all technical tasks required to get DSpace in a state in which it operates properly so its behaviour is predictable and can be used according to all the guidelines under "Using DSpace".

5.1 Introduction to DSpace System Administration

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

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

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

DSpace Command Launcher

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

- [AIP Backup and Restore](#)
 - [DSpace AIP Format](#)
- [Ant targets and options](#)
- [Command Line Operations](#)

- [Executing streams of commands](#)
- [Database Utilities](#)
- [Legacy methods for re-indexing content](#)
- [Mediafilters for Transforming DSpace Content](#)
 - [ImageMagick Media Filters](#)
- [Performance Tuning DSpace](#)
- [Scheduled Tasks via Cron](#)
- [Search Engine Optimization](#)
 - [Google Scholar Metadata Mappings](#)
- [Troubleshooting Information](#)
- [Validating CheckSums of Bitstreams](#)

5.2 AIP Backup and Restore


- [1 Background & Overview](#)
 - [1.1 How does this differ from traditional DSpace Backups? Which Backup route is better?](#)
 - [1.2 How does this help backup your DSpace to remote storage or cloud services \(like DuraCloud\)?](#)
 - [1.3 AIPs are Archival Information Packages](#)
 - [1.4 AIP Structure / Format](#)
- [2 Running the Code](#)
 - [2.1 Exporting AIPs](#)
 - [2.1.1 Export Modes & Options](#)
 - [2.1.2 Exporting just a single AIP](#)
 - [2.1.3 Exporting AIP Hierarchy](#)
 - [2.1.3.1 Exporting Entire Site](#)
 - [2.2 Ingesting / Restoring AIPs](#)
 - [2.2.1 Ingestion Modes & Options](#)
 - [2.2.1.1 The difference between "Submit" and "Restore/Replace" modes](#)
 - [2.2.2 Submitting AIP\(s\) to create a new object](#)
 - [2.2.2.1 Submitting a Single AIP](#)
 - [2.2.2.2 Submitting an AIP Hierarchy](#)
 - [2.2.2.3 Submitting AIP\(s\) while skipping any Collection Approval Workflows](#)
 - [2.2.3 Restoring/Replacing using AIP\(s\)](#)
 - [2.2.3.1 Default Restore Mode](#)
 - [2.2.3.2 Restore, Keep Existing Mode](#)
 - [2.2.3.3 Force Replace Mode](#)
 - [2.2.3.4 Restoring Entire Site](#)
 - [2.3 Performance considerations](#)
 - [2.4 Disable User Interaction for Cron](#)
- [3 Command Line Reference](#)
 - [3.1 Additional Packager Options](#)
 - [3.1.1 How to use additional options](#)

- 4 [Configuration in 'dspace.cfg'](#)
 - 4.1 [AIP Metadata Dissemination Configurations](#)
 - 4.2 [AIP Ingestion Metadata Crosswalk Configurations](#)
 - 4.3 [AIP Ingestion EPerson Configurations](#)
 - 4.4 [AIP Configurations To Improve Ingestion Speed while Validating](#)
- 5 [Common Issues or Error Messages](#)

5.2.1 Background & Overview

AIP Backup & Restore functionality only works with the Latest Version of Items

If you are using the new XMLUI-only [Item Level Versioning](#) functionality (disabled by default), you must be aware that this "Item Level Versioning" feature is **not yet compatible** with AIP Backup & Restore. **Using them together may result in accidental data loss.** Currently the AIPs that DSpace generates only store the *latest version* of an Item. Therefore, past versions of Items will always be lost when you perform a restore / replace using AIP tools.

 Additional background information available in the Open Repositories 2010 Presentation entitled [Improving DSpace Backups, Restores & Migrations](#)

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

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

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

Benefits for the DSpace community:

- Allows one to more easily move entire Communities or Collections between DSpace instances.
- Allows for a potentially more consistent backup of this hierarchy (e.g. to DuraCloud, or just to your own local backup system), rather than relying on synchronizing a backup of your Database (stores metadata/relationships) and assetstore (stores files/bitstreams).
- Provides a way for people to more easily get their data out of DSpace (whatever the purpose may be).

- Provides a relatively standard format for people to migrate entire hierarchies (Communities/Collections) from one DSpace to another (or from another system into DSpace).

How does this differ from traditional DSpace Backups? Which Backup route is better?

Traditionally, it has always been recommended to backup and restore DSpace's database and files (also known as the "assetstore") separately. This is described in more detail in the [Storage Layer](#) section of the DSpace System Documentation. The traditional backup and restore route is still a recommended and supported option.

However, the new AIP Backup & Restore option seeks to try and resolve many of the complexities of a traditional backup and restore. The below table details some of the differences between these two valid Backup and Restore options.

	Traditional Backup & Restore (Database and Files)	AIP Backup & Restore
Supported Backup/Restore Types		
Can Backup & Restore all DSpace Content easily	Yes (Requires two backups/restores – one for Database and one for Files)	Yes (Though, will not backup/restore items which are not officially "in archive")
Can Backup & Restore a Single Community/Collection/Item easily	No (It is possible, but requires a strong understanding of DSpace database structure & folder organization in order to only backup & restore metadata/files belonging to that single object)	Yes
Backups can be used to move one or more Community/Collection/Items to another DSpace system easily.	No (Again, it is possible, but requires a strong understanding of DSpace database structure & folder organization in order to only move metadata/files belonging to that object)	Yes

	Traditional Backup & Restore (Database and Files)	AIP Backup & Restore
Can Backup & Restore Item Versions	Yes (Requires two backups/restores – one for Database and one for Files)	No (Currently Item Level Versioning is not fully compatible with AIP Backup & Restore. AIP Backup & Restore can only backup/restore the <i>latest version</i> of an Item)
Supported Object Types During Backup & Restore		
Supports backup/restore of all Communities/Collections/Items (including metadata, files, logos, etc.)	Yes	Yes
Supports backup/restore of all People/Groups/Permissions	Yes	Yes
Supports backup/restore of all Collection-specific Item Templates	Yes	Yes
Supports backup/restore of all Collection Harvesting settings (only for Collections which pull in all Items via OAI-PMH or OAI-ORE)	Yes	No (This is a known issue. All previously harvested Items will be restored, but the OAI-PMH/OAI-ORE harvesting settings will be lost during the restore process .)
Supports backup/restore of all Withdrawn (but not deleted) Items	Yes	Yes
	Yes	

	Traditional Backup & Restore (Database and Files)	AIP Backup & Restore
Supports backup/restore of Item Mappings between Collections		Yes (During restore, the AIP Ingestor may throw a false "Could not find a parent DSpaceObject" error (see Common Issues or Error Messages), if it tries to restore an Item Mapping to a Collection that it hasn't yet restored. But this error can be safely bypassed using the 'skipIfParentMissing' flag (see Additional Packager Options for more details).
Supports backup/restore of all in-process, uncompleted Submissions (or those currently in an approval workflow)	Yes	No (AIPs are only generated for objects which are completed and considered "in archive")
Supports backup/restore of Items using custom Metadata Schemas & Fields	Yes	Yes (Custom Metadata Fields will be automatically recreated. Custom Metadata Schemas must be manually created first, in order for DSpace to be able to recreate custom fields belonging to that schema. See Common Issues or Error Messages for more details.)
Supports backup/restore of all local DSpace Configurations and Customizations	Yes (if you backup your entire DSpace directory as part of backing up your files)	Not by default (unless you also backup parts of your DSpace directory – note, you wouldn't need to backup the '[dspace]/assetstore' folder again, as those files are already included in AIPs)

Based on your local institutions needs, you will want to choose the backup & restore process which is most appropriate to you. You may also find it beneficial to use both types of backups on different time schedules, in order to keep to a minimum the likelihood of losing your DSpace installation settings or its contents. For example , you may choose to perform a Traditional Backup once per week (to backup your local system configurations and customizations) and an AIP Backup on a daily basis. Alternatively, you may choose to perform daily Traditional Backups and only use the AIP Backup as a "permanent archives" option (perhaps performed on a weekly or monthly basis).



Don't Forget to Backup your Configurations and Customizations

If you choose to use the AIP Backup and Restore option, do not forget to also backup your local DSpace configurations and customizations. Depending on how you manage your own local DSpace, these configurations and customizations are likely in one or more of the following locations:

- [`dspace`] - The DSpace installation directory (Please note, if you also use the AIP Backup & Restore option, you do **not** need to backup your [`dspace`]/`assetstore` directory, as those files already exist in your AIPs).
- [`dspace-source`] - The DSpace source directory

How does this help backup your DSpace to remote storage or cloud services (like DuraCloud)?

While AIP Backup and Restore is primarily a way to export your DSpace content objects to a local filesystem (or mounted drive), it can also be used as the basis for ensuring your content is safely backed up in a remote location (e.g. [DuraCloud](#) or other cloud backup services).

Simply put, these AIPs can be generated and then replicated off to remote storage or a cloud backup service for safe keeping. You can then pull them down either as an entire set, or individually, in order to restore one or more objects into your DSpace instance. While you could simply backup your entire DSpace database and "assetstore" to a cloud service, you'd have to download the **entire** database backup again in order to restore any content. With AIPs, you can instead just download the individual AIP files you need (which can decrease your I/O costs, if any exist) for that restoration.

This upload/download of your AIPs to a backup location can be managed in a manual fashion (e.g. via your own custom code or shell scripts), or you can use a DSpace [Replication Task Suite](#) add-on to help ease this process

The Replication Task Suite add-on for DSpace allows you the ability to backup and restore DSpace contents to/from AIPs via the DSpace Administrative Web Interface. It also includes "connectors" to the [DuraCloud](#) API, so you can configure it to automatically backup/retrieve your AIPs to/from DuraCloud. Installing this add-on means you can now easily backup and restore DSpace to DuraCloud (or other systems) simply via the DSpace Administrative Web Interface. More information on installing and configuring this add-on can be found on the [Replication Task Suite](#) page.

Makeup and Definition of AIPs

AIPs are Archival Information Packages

- AIP is a package describing **one archival object** in DSpace.
 - The **archival object** may be a single **Item**, **Collection**, **Community**, or **Site** (Site AIPs contain site-wide information). Bitstreams are included in an Item's AIP.
 - Each AIP is logically self-contained, can be restored without rest of the archive. (So you could restore a single Item, Collection or Community)

- Collection or Community AIPs do **not** include all child objects (e.g. Items in those Collections or Communities), as each AIP only describes **one** object. However, these container AIPs do contain references (links) to all child objects. These references can be used by DSpace to automatically restore all referenced AIPs when restoring a Collection or Community.
- AIPs are only generated for objects which are currently in the "in archive" state in DSpace. This means that in-progress, uncompleted submissions are not described in AIPs and cannot be restored after a disaster. Permanently removed objects will also no longer be exported as AIPs after their removal. However, withdrawn objects will continue to be exported as AIPs, since they are still considered under the "in archive" status.
- AIPs with identical contents will always have identical [checksums](#). This provides a basic means of validating whether the contents within an AIP have changed. For example, if a Collection's AIP has the same checksum at two different points in time, it means that Collection has not changed during that time period.
- AIP profile favors completeness and accuracy rather than presenting the semantics of an object in a standard format. It conforms to the quirks of DSpace's internal object model rather than attempting to produce a universally understandable representation of the object. When possible, an AIP tries to use common standards to express objects.
- An AIP *can* serve as a DIP (Dissemination Information Package) or SIP (Submission Information Package), especially when transferring custody of objects to another DSpace implementation.
- In contrast to SIP or DIP, the AIP should include all available DSpace structural and administrative metadata, and basic provenance information. AIPs also describe some basic system level information (e.g. Groups and People).

AIP Structure / Format

Generally speaking, an AIP is an Zip file containing a METS manifest and all related content bitstreams.

For more specific details of AIP format / structure, along with examples, please see [DSpace AIP Format](#).

5.2.2 Running the Code

Exporting AIPs

Export Modes & Options

All AIP Exports are done by using the Dissemination Mode (`-d` option) of the `packager` command.

There are two types of AIP Dissemination you can perform:

- [Single AIP](#) (default, using `-d` option) - Exports just an AIP describing a single DSpace object. So, if you ran it in this default mode for a Collection, you'd just end up with a single Collection AIP (which would not include AIPs for all its child Items)

- **Hierarchy of AIPs** (using the `-d --all` or `-d -a` option) - Exports the requested AIP describing an object, plus the AIP for all child objects. Some examples follow:
 - For a Site - this would export **all** Communities, Collections & Items within the site into AIP files (in a provided directory)
 - For a Community - this would export that Community and all SubCommunities, Collections and Items into AIP files (in a provided directory)
 - For a Collection - this would export that Collection and all contained Items into AIP files (in a provided directory)
 - For an Item – this just exports the Item into an AIP as normal (as it already contains its Bitstreams/ Bundles by default)

Exporting just a single AIP

To export in single AIP mode (default), use this "packager" command template:

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

for example:

```
[dspace]/bin/dspace packager -d -t AIP -e admin@myu.edu -i 4321/4567 aip4567.zip
```

The above code will export the object of the given handle (4321/4567) into an AIP file named "aip4567.zip". This will **not** include any child objects for Communities or Collections.

Exporting AIP Hierarchy

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

For example, use this 'packager' command template:

```
[dspace]/bin/dspace packager -d -a -t AIP -e <eperson> -i <handle> <file-path>
```

for example:

```
[dspace]/bin/dspace packager -d -a -t AIP -e admin@myu.edu -i 4321/4567 aip4567.zip
```

The above code will export the object of the given handle (4321/4567) into an AIP file named "aip4567.zip". In addition it would export all children objects to the same directory as the "aip4567.zip" file. The child AIP files are all named using the following format:

- File Name Format: <Obj-Type>@<Handle-with-dashes>.zip
 - e.g. COMMUNITY@123456789-1.zip, COLLECTION@123456789-2.zip, ITEM@123456789-200.zip

- This general file naming convention ensures that you can easily locate an object to restore by its name (assuming you know its Object Type and Handle).
- Alternatively, if object doesn't have a Handle, it uses this File Name Format: <Obj-Type>@internal-id-<DSpace-ID>.zip (e.g. ITEM@internal-id-234.zip)

AIPs are only generated for objects which are currently in the "in archive" state in DSpace. This means that in-progress, uncompleted submissions are not described in AIPs and cannot be restored after a disaster.

Exporting Entire Site

To export an entire DSpace Site, pass the packager the Handle <site-handle-prefix>/0. For example, if your site prefix is "4321", you'd run a command similar to the following:

```
[dspace]/bin/dspace packager -d -a -t AIP -e admin@myu.edu -i 4321/0 sitewide-aip.zip
```

Again, this would export the DSpace Site AIP into the file "sitewide-aip.zip", and export AIPs for **all** Communities, Collections and Items into the same directory as the Site AIP.

Ingesting / Restoring AIPs

Ingestion Modes & Options

Ingestion of AIPs is a bit more complex than Dissemination, as there are several different "modes" available:

1. **Submit/Ingest Mode** (-s option, default) – submit AIP(s) to DSpace in order to create a new object(s) (i.e. AIP is treated like a SIP – Submission Information Package)
2. **Restore Mode** (-r option) – restore pre-existing object(s) in DSpace based on AIP(s). This also attempts to restore all handles and relationships (parent/child objects). This is a specialized type of "submit", where the object is created with a known Handle and known relationships.
3. **Replace Mode** (-r -f option) – replace existing object(s) in DSpace based on AIP(s). This also attempts to restore all handles and relationships (parent/child objects). This is a specialized type of "restore" where the contents of existing object(s) is replaced by the contents in the AIP(s). By default, if a normal "restore" finds the object already exists, it will back out (i.e. rollback all changes) and report which object already exists.


Again, like export, there are two types of AIP Ingestion you can perform (using any of the above modes):

- **Single AIP** (default) - Ingests just an AIP describing a single DSpace object. So, if you ran it in this default mode for a Collection AIP, you'd just create a DSpace Collection from the AIP (but not ingest any of its child objects)
- **Hierarchy of AIPs** (by including the --all or -aoption after the mode) - Ingests the requested AIP describing an object, plus the AIP for all child objects. Some examples follow:
 - For a Site - this would ingest **all** Communities, Collections & Items based on the located AIP files

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

The difference between "Submit" and "Restore/Replace" modes

It's worth understanding the primary differences between a Submission (specified by `-s` parameter) and a Restore (specified by `-r` parameter).

- **Submission Mode** (`-s` mode) - creates a new object (AIP is treated like a SIP)
 - By default, a new Handle is always assigned
 - However, you can force it to use the handle specified in the AIP by specifying `-o ignoreHandle=false` as one of your parameters
 - By default, a new Parent object **must** be specified (using the `-p` parameter). This is the location where the new object will be created.
 - However, you can force it to use the parent object specified in the AIP by specifying `-o ignoreParent=false` as one of your parameters
 - By default, will respect a Collection's Workflow process when you submit an Item to a Collection
 - However, you can specifically *skip* any workflow approval processes by specifying `-w` parameter.
 - **Always** adds a new Deposit License to Items
 - **Always** adds new DSpace System metadata to Items (includes new "dc.date.accessioned", "dc.date.available", "dc.date.issued" and "dc.description.provenance" entries)
 - **WARNING:** Submission mode may not be able to maintain Item Mappings between Collections. Because these mappings are recorded via the Collection Handles, mappings may be restored improperly if the Collection handle has changed when moving content from one DSpace instance to another.
- **Restore / Replace Mode** (`-r` mode) - restores a previously existing object (as if from a backup)
 - By default, the Handle specified in the AIP is restored
 - However, for restores, you can force a new handle to be generated by specifying `-o ignoreHandle=true` as one of your parameters. (NOTE: Doesn't work for *replace* mode as the new object always retains the handle of the replaced object)
 -  Although a Restore/Replace does restore Handles, it will not necessarily restore the same internal IDs in your Database.
 - By default, the object is restored under the Parent specified in the AIP
 - However, for restores, you can force it to restore under a different parent object by using the `-p` parameter. (NOTE: Doesn't work for *replace* mode, as the new object always retains the parent of the replaced object)
 - **Always** skips any Collection workflow approval processes when restoring/replacing an Item in a Collection

- **Never** adds a new Deposit License to Items (rather it restores the previous deposit license, as long as it is stored in the AIP)
- **Never** adds new DSpace System metadata to Items (rather it just restores the metadata as specified in the AIP)

Changing Submission/Restore Behavior

It is possible to change some of the default behaviors of both the Submission and Restore/Replace Modes. Please see the [Additional Packager Options](#) section below for a listing of command-line options that allow you to override some of the default settings described above.

Submitting AIP(s) to create a new object

The Submission mode (`-s`) always creates a new object with a newly assigned handle. In addition by default it respects all existing Collection approval workflows (so items may require approval unless the workflow is skipped by using the `-w` option). For information about how the "Submission Mode" differs from the "Replace / Restore mode", see [The difference between "Submit" and "Restore/Replace" modes](#) above.

Submitting a Single AIP

AIPs treated as SIPs

This option allows you to essentially use an AIP as a SIP (Submission Information Package). The default settings will create a new DSpace object (with a new handle and a new parent object, if specified) from your AIP.

To ingest a single AIP and create a new DSpace object under a parent of your choice, specify the `-p` (or `--parent`) package parameter to the command. Also, note that you are running the `packager` in `-s` (submit) mode.

NOTE: This only ingests the single AIP specified. It does **not** ingest all children objects.

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

If you leave out the `-p` parameter, the AIP package ingester will attempt to install the AIP under the same parent it had before. As you are also specifying the `-s` (submit) parameter, the `packager` will assume you want a new Handle to be assigned (as you are effectively specifying that you are submitting a **new** object). If you want the object to retain the Handle specified in the AIP, you can specify the `-o ignoreHandle=false` option to force the packager to *not* ignore the Handle specified in the AIP.

Submitting an AIP Hierarchy



AIPs treated as SIPs

This option allows you to essentially use a set of AIPs as SIPs (Submission Information Packages). The default settings will create a new DSpace object (with a new handle and a new parent object, if specified) from each AIP

To ingest an AIP hierarchy from a directory of AIPs, use the `-a` (or `--all`) package parameter.

For example, use this 'packager' command template:

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

for example:

```
[dspace]/bin/dspace packager -s -a -t AIP -e admin@myu.edu -p 4321/12 aip4567.zip
```

The above command will ingest the package named "aip4567.zip" as a child of the specified Parent Object (handle="4321/12"). The resulting object is assigned a new Handle (since `-s` is specified). In addition, any child AIPs referenced by "aip4567.zip" are also recursively ingested (a new Handle is also assigned for each child AIP).

Another example – **Ingesting a Top-Level Community** (by using the Site Handle, `<site-handle-prefix>/0`):

```
[dspace]/bin/dspace packager -s -a -t AIP -e admin@myu.edu -p 4321/0 community-aip.zip
```

The above command will ingest the package named "community-aip.zip" as a **top-level community** (i.e. the specified parent is "4321/0" which is a Site Handle). Again, the resulting object is assigned a new Handle. In addition, any child AIPs referenced by "community-aip.zip" are also recursively ingested (a new Handle is also assigned for each child AIP).



May want to skip Collection Approvals Workflows

Please note: If you are submitting a larger amount of content (e.g. multiple Communities/Collections) to your DSpace, you may want to tell the 'packager' command to skip over any existing Collection approval workflows by using the `-w` flag. By default, all Collection approval workflows will be respected. This means if the content you are submitting includes a Collection with an enabled workflow, you may see the following occur:

1. First, the Collection will be created & its workflow enabled
2. Second, each Item belonging to that Collection will be created & placed into the workflow approval process

Therefore, if this content has already received some level of approval, you may want to submit it using the `-w` flag, which will skip any workflow approval processes. For more information, see [Submitting AIP\(s\) while skipping any Collection Approval Workflows](#).

⊖ Item Mappings may not be maintained when submitting an AIP hierarchy

When an Item is mapped to one or more Collections, this mapping is recorded in the AIP using the mapped Collection's handle. Unfortunately, since the submission mode (`-s`) assigns **new handles** to all objects in the hierarchy, this may mean that the mapped Collection's handle will have changed (or even that a different Collection will be available at the original mapped Collection's handle). DSpace does not have a way to uniquely identify Collections other than by handle, which means that item mappings are only able to be retained when the Collection handle is *also retained*.

If you encounter this issue, there are a few possible workarounds:

1. Use the restore/replace mode (`-r`) instead, as it will retain existing Collection Handles. Unfortunately though, this may not work if the content is being moved from a Test DSpace to a Production DSpace, as these existing handles may not be valid.
2. OR, use the submission mode with the `--o ignoreHandle=false`. This will also retain existing Collection Handles. Unfortunately though, this may not work if the content is being moved from a Test DSpace to a Production DSpace, as these existing handles may not be valid.
3. OR, remove all existing Item Mappings and re-export AIPs (without Item Mappings). Then, import the hierarchy into the new DSpace instance (again without Item Mappings). Finally, recreate the necessary Item Mappings using a different tool, e.g. the [Batch Metadata Editing](#) tool supports bulk editing of Collection memberships/mappings.

⊖ Missing Groups or EPeople cannot be created when submitting an individual Community or Collection AIP

Please note, if you are using AIPs to move an entire Community or Collection from one DSpace to another, there is a known issue (see [DS-1105](#)) that the new DSpace instance will be unable to (re-) create any DSpace Groups or EPeople which are referenced by a Community or Collection AIP. The reason is that the Community or Collection AIP itself doesn't contain enough information to create those Groups or EPeople (rather that info is stored in the SITE AIP, for usage during [Full Site Restores](#)).

However, there are two possible ways to get around this known issue:

- **EITHER**, you can manually recreate all referenced Groups/EPeople in the new DSpace that you are submitting the Community or Collection AIP into.
 - Note that if you are using Groups named with DSpace Database IDs (e.g. COMMUNITY_1_ADMIN, COLLECTION_2_SUBMIT), you may first need to rename those groups to no longer include Database IDs (e.g. MY_SUBMITTERS). The reason is that Database IDs will likely change when you move a Community or Collection to a new DSpace installation.
- **OR**, you can temporarily disable the import of Group/EPeople information when submitting the Community or Collection AIP to the new DSpace. This would mean that after you submit the AIP to the new DSpace, you'd have to manually go in and add in any special permissions (as needed). To disable the import of Group/EPeople information, add these settings to your `dspace.cfg` file, and re-run the submission of the AIP with these settings in place:

```
mets.dspaceAIP.ingest.crosswalk.METSRIGHTS = NIL
mets.dspaceAIP.ingest.crosswalk.DSPACE-ROLES = NIL
```

- Don't forget to remove these settings after you import your Community or Collection AIP. *Leaving them in place will mean that every time you import an AIP, all of its Group/EPeople/Permissions would be ignored.*

Submitting AIP(s) while skipping any Collection Approval Workflows

By default, the Submission mode (`-s`) always respects existing Collection approval workflows. So, if a Collection has a workflow, then a newly submitted Item will be placed into that workflow process (rather than immediately appearing in DSpace).

However, if you'd like to skip all workflow approval processes you can use the `-w` flag to do so. For example, the following command will skip any Collection approval workflows and immediately add the Item to a Collection.

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

This `-w` flag may also be used when [Submitting an AIP Hierarchy](#). For example, if you are migrating one or more Collections/Communities from one DSpace to another, you may choose to submit those AIPs with the `-w` option enabled. This will ensure that, if a Collection has a workflow approval process enabled, all its Items are available immediately rather than being all placed into the workflow approval process.

Restoring/Replacing using AIP(s)

Restoring is slightly different than just **submitting**. When restoring, we make every attempt to restore the object as it **used to be** (including its handle, parent object, etc.). For more information about how the "Replace/Restore Mode" differs from the "Submit mode", see [The difference between "Submit" and "Restore/Replace" modes](#) above.

There are currently three restore modes:

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

Default Restore Mode

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

Restore a Single AIP: Use this 'packager' command template to restore a single object from an AIP (not including any child objects):

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

Restore a Hierarchy of AIPs: Use this 'packager' command template to restore an object from an AIP along with all child objects (from their AIPs):

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

For example:

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

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

In the above example, the package "aip4567.zip" is restored to the DSpace installation with the Handle provided within the package itself (and added as a child of the parent object specified within the package itself). In addition, any child AIPs referenced by "aip4567.zip" are also recursively ingested (the `-a` option specifies to also restore all child AIPs). They are also restored with the Handles & Parent Objects provided with their package. If any object is found to already exist, all changes are rolled back (i.e. nothing is restored to DSpace)



Highly Recommended to Update Database Sequences after a Large Restore

In some cases, when you restore a large amount of content to your DSpace, the internal database counts (called "sequences") may get out of sync with the Handles of the content you just restored. As a best practice, it is **highly recommended to always** re-run the "update-sequences.sql" script on your DSpace database after a larger scale restore. This database script should be run while DSpace is stopped (you may either stop Tomcat or just the DSpace webapps). PostgreSQL/Oracle must be running. The script can be found in the following locations for PostgreSQL and Oracle, respectively:

```
[dspace]/etc/postgres/update-sequences.sql
```

```
[dspace]/etc/oracle/update-sequences.sql
```



More Information on using Default Restore Mode with Community/Collection AIPs

- Using the Default Restore Mode without the `-a` option, will only restore the **metadata** for that specific Community or Collection. No child objects will be restored.
- Using the Default Restore Mode with the `-a` option, will only successfully restore a Community or Collection if that object along with any child objects (Sub-Communities, Collections or Items) do not already exist. In other words, if any objects belonging to that Community or Collection already exist in DSpace, the Default Restore Mode will report an error that those object(s) could not be recreated. If you encounter this situation, you will need to perform the restore using either the [Restore, Keep Existing Mode](#) or the [Force Replace Mode](#) (depending on whether you want to keep or replace those existing child objects).

Restore, Keep Existing Mode

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

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

Restore a Hierarchy of AIPs: Use this 'packager' command template to restore an object from an AIP along with all child objects (from their AIPs):

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

For example:

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

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

Force Replace Mode

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



May also be useful in some specific restoration scenarios

This mode may also be used to restore missing objects which refer to existing objects. For example, if you are restoring a missing Collection which had existing Items linked to it, you can use this mode to auto-restore the Collection and update those existing Items so that they again link back to the newly restored Collection.



Potential for Data Loss

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

Replace using a Single AIP: Use this 'packager' command template to replace a single object from an AIP (not including any child objects):

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

Replace using a Hierarchy of AIPs: Use this 'packager' command template to replace an object from an AIP along with all child objects (from their AIPs):


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

For example:

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

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

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

Restoring Entire Site

In order to restore an entire Site from a set of AIPs, you must do the following:

1. Install a completely "fresh" version of DSpace by following the [Installation instructions in the DSpace Manual](#)
 - At this point, you should have a completely empty, but fully-functional DSpace installation. You will need to create an initial Administrator user in order to perform this restore (as a full-restore can only be performed by a DSpace Administrator).
2. Once DSpace is installed, run the following command to restore all its contents from AIPs

```
[dspace]/bin/dspace packager -r -a -f -t AIP -e <eperson> -i <site-handle-prefix>/0 -o skipIfParentMissing=true /full/path/to/your/site-aip.zip
```

- a. While the "-o skipIfParentMissing=true" flag is optional, it is often necessary whenever you are performing a large hierarchical site restoration. Please see the [Additional Packager Options](#) section below.

Please note the following about the above restore command:

- Notice that you are running this command in "Force Replace" mode (-r -f). This is necessary as your empty DSpace install will already include a few default groups (Administrators and Anonymous) and your initial administrative user. You need to replace these groups in order to restore your prior DSpace contents completely.
- <eperson> should be replaced with the Email Address of the initial Administrator (who you created when you reinstalled DSpace).
- <site-handle-prefix> should be replaced with your DSpace site's assigned Handle Prefix. This is equivalent to the handle.prefix setting in your dspace.cfg

- `/full/path/to/your/site-aip.zip` is the full path to the AIP file which represents your DSpace SITE. This file will be named whatever you named it when you actually [exported your entire site](#). All other AIPs are assumed to be referenced from this SITE AIP (in most cases, they should be in the same directory as that SITE AIP).



Highly Recommended to Update Database Sequences after a Large Restore

In some cases, when you restore a large amount of content to your DSpace, the internal database counts (called "sequences") may get out of sync with the Handles of the content you just restored. As a best practice, it is **highly recommended to always** re-run the "update-sequences.sql" script on your DSpace database after a larger scale restore. This database script should be run while DSpace is stopped (you may either stop Tomcat or just the DSpace webapps). PostgreSQL/Oracle must be running. The script can be found in the following locations for PostgreSQL and Oracle, respectively:

```
[dspace]/etc/postgres/update-sequences.sql  
[dspace]/etc/oracle/update-sequences.sql
```

Performance considerations

When importing large structures like the whole site or a large collection/community, keep in mind that this can require a lot of memory, more than the default amount of heap allocated to the command-line launcher (256 Mb: `JAVA_OPTS="-Xmx256m -Dfile.encoding=UTF-8"`). This memory must be allocated in addition to the normal amount of memory allocated to Tomcat. For example, a site of 2500 fulltext items (2 Gb altogether) requires 5 Gb of maximum heap space and takes around 1 hour, including import and indexing.

You can raise the limit for a single run of the packager command by specifying memory options in the `JAVA_OPTS` environment variable, e.g.:

```
JAVA_OPTS="-Xmx4096m -Dfile.encoding=UTF-8" /dspace/bin/dspace packager -u -r -a -f -t AIP -e  
dspace@example.com -i 123456789/0 sitewide-aip.zip
```

If the importer runs out of heap memory, it will crash either with "java.lang.OutOfMemoryError: GC overhead limit exceeded", which can be suppressed by adding "-XX:-UseGCOverheadLimit" to `JAVA_OPTS`, or with "java.lang.OutOfMemoryError: Java heap space". You can increase the allocated heap memory and try again, but keep in mind that although no changes were made in the database, the unsuccessfully imported files are still left in the assetstore (see [DS-2227](#)).

Disable User Interaction for Cron

If you wish to run any of the following commands from a cron job (or similar), then you may wish to **disable all user interaction** using the `-u` (`--no-user-interaction`) flag. For example, supposing you wanted to perform a full Site Backup (see [Exporting Entire Site](#) above) via a cronjob, you could simply run that command passing it the "-u" flage like this:

```
# Perform a full site backup to AIPs(with user interaction disabled) every Sunday at 1:00AM
# NOTE: Make sure to replace "123456789" with your actual Handle Prefix, and "admin@myu.edu" with
your Administrator account email.
0 1 * * * [dSPACE]/bin/dSPACE packager -u -d -a -t AIP -e admin@myu.edu -i 123456789/0 [
full-path-to-backup-folder]/sitewide-aip.zip
```

5.2.3 Command Line Reference

The following flags are valid to pass to the `[dSPACE]/bin/dSPACE packager` command:

Flag	Ingest or Export	Description / Usage
<code>-a (--all)</code>	both ingest and export	<i>For Ingest:</i> recursively ingest all child AIPs (referenced from this AIP) . <i>For Export:</i> recursively export all child objects (referenced from this parent object)
<code>-d (--disseminate)</code>	export-only	This flag simply triggers the export of AIPs from the system. See Exporting AIPs
<code>-e (-eperson) [email-address]</code>	ingest-only	The email address of the EPerson who is ingesting the AIPs. Oftentimes this should be an Administrative account.
<code>-f (--force-replace)</code>	ingest-only	Ingest the AIPs in " Force Replace Mode " (must be specified in conjunction with <code>-r</code> flag), where existing objects will be replaced by the contents of the AIP.
<code>-h (--help)</code>	both ingest and export	Return help information. You should specify with <code>-t</code> for additional package specific help information
<code>-i (--identifier) [handle]</code>	both ingest and export	<i>For Ingest:</i> Only valid in " Force Replace Mode ". In that mode this is the identifier of the object to replace. <i>For Export:</i> The identifier of the object to export to an AIP
<code>-k (--keep-existing)</code>	ingest-only	Specifies to use " Restore, Keep Existing Mode " during ingest (must be specified in conjunction with <code>-r</code> flag). In this mode, existing objects in DSpace will NOT be replaced by their AIPs, but missing objects will be restored from AIPs.
<code>-o (--option) [setting]=[value]</code>	both ingest and export	This flag is used to pass Additional Packager Options to the Packager command. Each type of packager may define its own

Flag	Ingest or Export	Description / Usage
		custom Additional Options. For AIPs, the valid options are documented in the Additional Packager Options section below. This is repeatable (e.g. <code>-o [setting1]=[value] -o [setting2]=value</code>)
<code>-p (--parent) [handle]</code>	ingest only	Handle(s) of the parent Community or Collection to into which an AIP should be ingested. This may be repeatable.
<code>-r (--restore)</code>	ingest only	Specifies that this ingest is either " Restore Mode " (when standalone), " Restore, Keep Existing Mode " (when used with <code>-k</code> flag) or " Force Replace Mode " (when used with <code>-f</code> flag)
<code>-s (--submit)</code>	ingest only	Specifies that this ingest is in " Submit Mode " where an AIP is treated as a new object and assigned a new Handle/Identifier, etc.
<code>-t (--type) [package-type]</code>	both ingest and export	Specifies the type of package which is being ingested or exported. This controls which Ingestor or Disseminator class is called. For AIPs, this is always set to " <code>-t AIP</code> "
<code>-u (--no-user-interaction)</code>	both ingest and export	Skips over all user interaction (e.g. question prompts). This flag can be used when running the packager from a script or cron job to bypass all user interaction. See also Disable User Interaction for Cron

Additional Packager Options

In addition to the various "modes" settings described under "[Running the Code](#)" above, the AIP Packager supports the following packager options. These options allow you to better tweak how your AIPs are processed (especially during ingests/restores/replaces).

Option	Ingest or Export	Default Value	Description
<code>createMetadataFields=[value]</code>	ingest-only	true	Tells the AIP ingestor to automatically create any metadata fields which are found to be missing from the DSpace Metadata Registry. When 'true', this means as each AIP is ingested, new fields may be added to the DSpace Metadata Registry if they don't already exist. When 'false', an AIP ingest will fail if it encounters a metadata field that doesn't exist in the DSpace Metadata Registry. (NOTE: This will not

Option	Ingest or Export	Default Value	Description
			create missing DSpace Metadata <i>Schemas</i> . If a schema is found to be missing, the ingest will always fail.)
filterBundles=[value]	export-only	defaults to exporting all Bundles	<p>This option can be used to limit the Bundles which are exported to AIPs for each DSpace Item. By default, all file Bundles will be exported into Item AIPs. You could use this option to limit the size of AIPs by only exporting certain Bundles. <i>WARNING: any bundles not included in AIPs will obviously be unable to be restored.</i> This option can be run in two ways:</p> <ul style="list-style-type: none"> • Exclude Bundles: By default, you can provide a comma-separated list of bundles to be excluded from AIPs (e.g. "TEXT, THUMBNAIL") • Include Bundles: If you prepend the list with the "+" symbol, then the list specifies the bundles to be included in AIPs (e.g. "+ ORIGINAL,LICENSE" would only include those two bundles). This second option is identical to using "includeBundles" option described below. <p>(NOTE: If you choose to no longer export LICENSE or CC_LICENSE bundles, you will also need to disable the License Dissemination Crosswalks in the <code>aip.disseminate.rightsMD</code> configuration for the changes to take affect)</p>
ignoreHandle=[value]]	ingest-only	Restore/ Replace Mode defaults to 'false', Submit Mode defaults to 'true'	<p>If 'true', the AIP ingester will ignore any Handle specified in the AIP itself, and instead create a new Handle during the ingest process (this is the default when running in Submit mode, using the <code>-s</code> flag). If 'false', the AIP ingester attempts to restore the Handles specified in the AIP (this is the default when running in Restore/replace mode, using the <code>-r</code> flag).</p>

Option	Ingest or Export	Default Value	Description
ignoreParent=[value]	ingest-only	Restore/Replace Mode defaults to 'false', Submit Mode defaults to 'true'	If 'true', the AIP ingester will ignore any Parent object specified in the AIP itself, and instead ingest under a new Parent object (this is the default when running in Submit mode, using the <code>-s</code> flag). The new Parent object must be specified via the <code>-p</code> flag (run <code>dspace packager -h</code> for more help). If 'false', the AIP ingester attempts to restore the object directly under its old Parent (this is the default when running in Restore/replace mode, using the <code>-r</code> flag).
includeBundles=[value]	export-only	defaults to "all"	<p>This option can be used to limit the Bundles which are exported to AIPs for each DSpace Item. By default, all file Bundles will be exported into Item AIPs. You could use this option to limit the size of AIPs by only exporting certain Bundles. <i>WARNING: any bundles not included in AIPs will obviously be unable to be restored.</i> This option expects a comma separated list of bundle names (e.g. "ORIGINAL,LICENSE,CC_LICENSE,METADATA"), or "all" if all bundles should be included.</p> <p>(See "filterBundles" option above if you wish to exclude particular Bundles. However, this "includeBundles" option cannot be used at the same time as "filterBundles".)</p> <p>(NOTE: If you choose to no longer export LICENSE or CC_LICENSE bundles, you will also need to disable the License Dissemination Crosswalks in the <code>aip.disseminate.rightsMD</code> configuration for the changes to take affect)</p>
manifestOnly=[value]	both ingest and export	false	If 'true', the AIP Disseminator will only import/export a METS Manifest XML file (i.e. result will be an unzipped 'mets.xml' file), instead of a full AIP. This METS Manifest contains URI references to all content files, but does <i>not</i> contain any content files. This option is experimental and is meant for debugging purposes only. It should never be set to 'true' if you want to be able to restore content files. Again,

Option	Ingest or Export	Default Value	Description
			please note that when you use this option, the final result will be an XML file, NOT the normal ZIP-based AIP format.
passwords=[value]	export-only	false	If 'true' (and the 'DSpace-ROLES' crosswalk is enabled, see #AIP Metadata Dissemination Configurations), then the AIP Disseminator will export user password hashes (i.e. encrypted passwords) into Site AIP's METS Manifest. This would allow you to restore user's passwords from Site AIP. If 'false', then user password hashes are not stored in Site AIP, and passwords cannot be restored at a later time.
skipIfParentMissing=[value]	ingest-only	false	If 'true', ingestion will skip over any "Could not find a parent DSpaceObject" errors that are encountered during the ingestion process (Note: those errors will still be logged as "warning" messages in your DSpace log file). If you are performing a full site restore (or a restore of a larger Community/Collection hierarchy), you may encounter these errors if you have a larger number of Item mappings between Collections (i.e. Items which are mapped into several collections at once). When you are performing a recursive ingest, skipping these errors should not cause any problems. Once the missing parent object is ingested it will automatically restore the Item mapping that caused the error. For more information on this "Could not find a parent DSpaceObject" error see Common Issues or Error Messages .
unauthorized=[value]	export-only	<i>unspecified</i>	If 'skip', the AIP Disseminator will skip over any unauthorized Bundle or Bitstream encountered (i.e. it will not be added to the AIP). If 'zero', the AIP Disseminator will add a Zero-length "placeholder" file to the AIP when it encounters an unauthorized Bitstream. If unspecified (the default value), the AIP Disseminator will throw an error if an unauthorized Bundle or Bitstream is encountered.
updatedAfter=[value]	export-only	<i>unspecified</i>	This option works as a basic form of "incremental backup". This option requires that an ISO-8601 date is

Option	Ingest or Export	Default Value	Description
			specified. When specified, the AIP Disseminator will only export Item AIPs which have a last-modified date after the specified ISO-8601 date. This option has no affect on the export of Site, Community or Collection AIPs as DSpace does not record a last-modified date for Sites, Communities or Collections. For example, when this option is specified during a full-site export, the AIP Disseminator will export the Site AIP, all Community AIPs, all Collection AIPs, and only Item AIPs modified after that date and time.
validate=[value]	both ingest and export	Export defaults to 'true', Ingest defaults to 'false'	If 'true', every METS file in AIP will be validated before ingesting or exporting. By default, DSpace will validate everything on export, but will skip validation during import. Validation on export will ensure that all exported AIPs properly conform to the METS profile (and will throw errors if any do not). Validation on import will ensure every METS file in every AIP is first validated before importing into DSpace (this will cause the ingestion processing to take longer, but tips on speeding it up can be found in the " AIP Configurations To Improve Ingestion Speed while Validating " section below). <i>DSpace recommends minimally validating AIPs on export. Ideally, you should validate both on export and import, but import validation is disabled by default in order to increase the speed of AIP restores.</i>

How to use additional options

These options can be passed in two main ways:

From the Command Line

From the command-line, you can add the option to your command by using the `-o` or `--option` parameter.

```
[dspace]/bin/dspace packager -r -a -t AIP -o [option1]=[value] -o [option2]=[value] -e admin@myu.edu aip4567.zip
```

For example:


```
[dspace]/bin/dspace packager -r -a -t AIP -o ignoreParent=false -o createMetadataFields=false -e  
admin@myu.edu aip4567.zip
```

Via the Java API call

If you are programmatically calling the `org.dspace.content.packager.DSpaceAIPIngester` from your own custom script, you can specify these options via the `org.dspace.content.packager.PackageParameters` class.

As a basic example:

```
PackageParameters params = new PackageParameters;  
params.addProperty("createMetadataFields", "false");  
params.addProperty("ignoreParent", "true");
```

5.2.4 Configuration in 'dspace.cfg'

The following new configurations relate to AIPs:

AIP Metadata Dissemination Configurations

The following configurations allow you to specify what metadata is stored within each METS-based AIP. In 'dspace.cfg', the general format for each of these settings is:

- `aip.disseminate.<setting> = <mdType>:<DSpace-crosswalk-name> [, ...]`
 - `<setting>` is the setting name (see below for the full list of valid settings)
 - `<mdType>` is optional. It allows you to specify the value of the `@MDTYPE` or `@OTHERMDTYPE` attribute in the corresponding METS element.
 - `<DSpace-crosswalk-name>` is required. It specifies the name of the DSpace Crosswalk which should be used to generate this metadata.
 - Zero or more `<label-for-METS>:<DSpace-crosswalk-name>` may be specified for each setting



AIP Metadata Recommendations

It is recommended to **minimally** use the default settings when generating AIPs. DSpace can only restore information that is included within an AIP. Therefore, if you choose to no longer include some information in an AIP, DSpace will no longer be able to restore that information from an AIP backup

The default settings in 'dspace.cfg' are:

- `aip.disseminate.techMD` - Lists the DSpace Crosswalks (by name) which should be called to populate the `<techMD>` section of the METS file within the AIP (Default: `PREMIS`, `DSPACE-ROLES`)
 - The `PREMIS` crosswalk generates PREMIS metadata for the object specified by the AIP
 - The `DSPACE-ROLES` crosswalk exports DSpace Group / EPerson information into AIPs in a DSpace-specific XML format. Using this crosswalk means that AIPs can be used to recreated Groups & People within the system. (NOTE: The `DSPACE-ROLES` crosswalk should be used alongside the `METSRights` crosswalk if you also wish to restore the *permissions* that Groups/ People have within the System. See below for more info on the `METSRights` crosswalk.)
- `aip.disseminate.sourceMD` - Lists the DSpace Crosswalks (by name) which should be called to populate the `<sourceMD>` section of the METS file within the AIP (Default: `AIP-TECHMD`)
 - The `AIP-TECHMD` Crosswalk generates technical metadata (in DIM format) for the object specified by the AIP
- `aip.disseminate.digiprovmD` - Lists the DSpace Crosswalks (by name) which should be called to populate the `<digiprovmD>` section of the METS file within the AIP (Default: *None*)
- `aip.disseminate.rightsMD` - Lists the DSpace Crosswalks (by name) which should be called to populate the `<rightsMD>` section of the METS file within the AIP (Default: `DSpaceDepositLicense`: `DSPACE_DEPLICENSE`, `CreativeCommonsRDF:DSPACE_CCRDF`, `CreativeCommonsText`: `DSPACE_CCTEXT`, `METSRights`)
 - The `DSPACE_DEPLICENSE` crosswalk ensures the DSpace Deposit License is referenced/stored in AIP
 - The `DSPACE_CCRDF` crosswalk ensures any Creative Commons RDF Licenses are reference/ stored in AIP
 - The `DSPACE_CCTEXT` crosswalk ensures any Creative Commons Textual Licenses are referenced /stored in AIP
 - The `METSRights` crosswalk ensures that Permissions/Rights on DSpace Objects (Communities, Collections, Items or Bitstreams) are referenced/stored in AIP. Using this crosswalk means that AIPs can be used to restore permissions that a particular Group or Person had on a DSpace Object. (NOTE: The `METSRights` crosswalk should always be used in conjunction with the `DSPACE-ROLES` crosswalk (see above) or a similar crosswalk. The `METSRights` crosswalk can only restore permissions, and cannot re-create Groups or EPeople in the system. The `DSPACE-ROLES` can actually re-create the Groups or EPeople as needed.)
- `aip.disseminate.dmd` - Lists the DSpace Crosswalks (by name) which should be called to populate the `<dmdSec>` section of the METS file within the AIP (Default: `MODS`, `DIM`)
 - The `MODS` crosswalk translates the DSpace descriptive metadata (for this object) into MODS. As MODS is a relatively "standard" metadata schema, it may be useful to include a copy of MODS metadata in your AIPs if you should ever want to import them into another (non-DSpace) system.
 - The `DIM` crosswalk just translates the DSpace internal descriptive metadata into an XML format. This XML format is proprietary to DSpace, but stores the metadata in a format similar to Qualified Dublin Core.

AIP Ingestion Metadata Crosswalk Configurations

The following configurations allow you to specify what DSpace Crosswalks are used during the ingestion/restoration of AIPs. These configurations also allow you to ignore areas of the METS file (in the AIP) if you do not want that area to be restored.

In `dspace.cfg`, the general format for each of these settings is:

- `mets.dspaceAIP.ingest.crosswalk.<mdType> = <DSpace-crosswalk-name>`
 - `<mdType>` is the type of metadata as specified in the METS file. This corresponds to the value of the `@MDTYPE` attribute (of that metadata section in the METS). When the `@MDTYPE` attribute is "OTHER", then the `<mdType>` corresponds to the `@OTHERMDTYPE` attribute value.
 - `<DSpace-crosswalk-name>` specifies the name of the DSpace Crosswalk which should be used to ingest this metadata into DSpace. You can specify the "NULLSTREAM" crosswalk if you specifically want this metadata to be ignored (and skipped over during ingestion).

By default, the settings in `dspace.cfg` are:

```
mets.dspaceAIP.ingest.crosswalk.DSpaceDepositLicense = NULLSTREAM
mets.dspaceAIP.ingest.crosswalk.CreativeCommonsRDF = NULLSTREAM
mets.dspaceAIP.ingest.crosswalk.CreativeCommonsText = NULLSTREAM
```

The above settings tell the ingester to **ignore** any metadata sections which reference DSpace Deposit Licenses or Creative Commons Licenses. These metadata sections can be safely ignored as long as the "LICENSE" and "CC_LICENSE" bundles are included in AIPs (which is the default setting). As the Licenses are included in those Bundles, they will already be restored when restoring the bundle contents.

More Info on Default Crosswalks used

If unspecified in the above settings, the AIP ingester will automatically use the Crosswalk which is named the same as the `@MDTYPE` or `@OTHERMDTYPE` attribute for the metadata section. For example, a metadata section with an `@MDTYPE="PREMIS"` will be processed by the DSpace Crosswalk named "PREMIS".

AIP Ingestion EPerson Configurations

The following setting determines whether the AIP Ingester should create an EPerson (if necessary) when attempting to restore or ingest an Item whose Submitter cannot be located in the system. By default it is set to "false", as for AIPs the creation of EPeople (and Groups) is generally handled by the `DSPACE-ROLES` crosswalk (see [#AIP Metadata Dissemination Configurations](#) for more info on `DSPACE-ROLES` crosswalk.)

- `mets.dspaceAIP.ingest.createSubmitter = false`

AIP Configurations To Improve Ingestion Speed while Validating

It is recommended to validate all AIPs on ingestion (when possible). But validation can be extremely slow, as each validation request first must download all referenced Schema documents from various locations on the web (sometimes as many as 10 schemas may be necessary to download in order to validate a single METS file). To make matters worse, the same schema will be re-downloaded each time it is used (i.e. it is not cached locally). So, if you are validating just 20 METS files which each reference 10 schemas, that results in 200 download requests.

In order to perform validations in a speedy fashion, you can pull down a local copy of **all** schemas. Validation will then use this local cache, which can sometimes increase the speed up to 10 x.

To use a local cache of XML schemas when validating, use the following settings in 'dSPACE.cfg'. The general format is:

- `mets.xsd.<abbreviation> = <namespace> <local-file-name>`
 - `<abbreviation>` is a unique abbreviation (of your choice) for this schema
 - `<namespace>` is the Schema namespace
 - `<local-file-name>` the full name of the cached schema file (which should reside in your `[dSPACE]/config/schemas/` directory, by default this directory does not exist – you will need to create it)

The default settings are all commented out. But, they provide a full listing of all schemas currently used during validation of AIPs. In order to utilize them, uncomment the settings, download the appropriate schema file, and save it to your `[dSPACE]/config/schemas/` directory (by default this directory does not exist – you will need to create it) using the specified file name:

```
#mets.xsd.mets = http://www.loc.gov/METS/ mets.xsd
#mets.xsd.xlink = http://www.w3.org/1999/xlink xlink.xsd
#mets.xsd.mods = http://www.loc.gov/mods/v3 mods.xsd
#mets.xsd.xml = http://www.w3.org/XML/1998/namespace xml.xsd
#mets.xsd.dc = http://purl.org/dc/elements/1.1/ dc.xsd
#mets.xsd.dcterms = http://purl.org/dc/terms/ dcterms.xsd
#mets.xsd.premis = http://www.loc.gov/standards/premis PREMIS.xsd
#mets.xsd.premisObject = http://www.loc.gov/standards/premis PREMIS-Object.xsd
#mets.xsd.premisEvent = http://www.loc.gov/standards/premis PREMIS-Event.xsd
#mets.xsd.premisAgent = http://www.loc.gov/standards/premis PREMIS-Agent.xsd
#mets.xsd.premisRights = http://www.loc.gov/standards/premis PREMIS-Rights.xsd
```

5.2.5 Common Issues or Error Messages

The below table lists common fixes to issues you may encounter when backing up or restoring objects using AIP Backup and Restore.

Issue / Error Message	How to Fix this Problem
Ingest/Restore Error: "Group Administrator already exists"	If you receive this problem, you are likely attempting to Restore an Entire Site , but are not running the command in Force Replace Mode (<code>-r -f</code>). Please see the section on Restoring an Entire Site for more details on the flags you should be using.
Ingest/Restore Error: "Unknown Metadata Schema encountered (mycustomschema)"	If you receive this problem, one or more of your Items is using a custom metadata schema which DSpace is currently not aware of (in the example, the schema is named " mycustomschema"). Because DSpace AIPs do not contain enough details to recreate the missing Metadata Schema, you must create it manually via the DSpace Admin UI. Please note that you only need to create the Schema. You do not need to manually create all the fields belonging to that schema, as DSpace will do that for you as it restores each AIP. Once the schema is created in DSpace, re-run your restore command. DSpace will automatically re-create all fields belonging to that custom metadata schema as it restores each Item that uses that schema.
Ingest Error: " Could not find a parent DSpaceObject referenced as 'xxx/xxx'"	When you encounter this error message it means that an object could not be ingested/restored as it belongs to a parent object which doesn't currently exist in your DSpace instance. During a full restore process, this error can be skipped over and treated as a warning by specifying the <code>'-o skipIfParentMissing=true'</code> option (see Additional Packager Options). If you have a larger number of Items which are mapped to multiple Collections, the AIP Ingester will sometimes attempt to restore an item mapping before the Collection itself has been restored (thus throwing this error). Luckily, this is not anything to be concerned about. As soon as the Collection is restored, the Item Mapping which caused the error will also be automatically restored. So, if you encounter this error during a full restore, it is safe to bypass this error message using the <code>'-o skipIfParentMissing=true'</code> option. All your Item Mappings should still be restored correctly.
Submit Error: PSQLError: ERROR: duplicate key value violates unique constraint " handle_handle_key "	This error means that while submitting one or more AIPs, DSpace encountered a Handle conflict. This is a general error that may occur in DSpace if your Handle sequence has somehow become out-of-date. However, it's easy to fix. Just run the <code>[dspace]/etc/postgres/update-sequences.sql</code> script (or if you are using Oracle, run: <code>[dspace]/etc/oracle/update-sequences.sql</code>).

5.2.6 DSpace AIP Format

- 1 [Makeup and Definition of AIPs](#)

- 1.1 AIPs are Archival Information Packages.
- 1.2 General AIP Structure / Examples
 - 1.2.1 Customizing What Is Stored in Your AIPs
- 2 AIP Details: METS Structure
- 3 Metadata in METS
 - 3.1 DIM (DSpace Intermediate Metadata) Schema
 - 3.1.1 DIM Descriptive Elements for Item objects
 - 3.1.2 DIM Descriptive Elements for Collection objects
 - 3.1.3 DIM Descriptive Elements for Community objects
 - 3.1.4 DIM Descriptive Elements for Site objects
 - 3.2 MODS Schema
 - 3.3 AIP Technical Metadata Schema (AIP-TECHMD)
 - 3.3.1 AIP Technical Metadata for Item
 - 3.3.2 AIP Technical Metadata for Bitstream
 - 3.3.3 AIP Technical Metadata for Collection
 - 3.3.4 AIP Technical Metadata for Community
 - 3.3.5 AIP Technical Metadata for Site
 - 3.4 PREMIS Schema
 - 3.4.1 PREMIS Metadata for Bitstream
 - 3.5 DSPACE-ROLES Schema
 - 3.5.1 Example of DSPACE-ROLES Schema for a SITE AIP
 - 3.5.2 Example of DSPACE-ROLES Schema for a Community or Collection
 - 3.6 METSRights Schema
 - 3.6.1 Example of METSRights Schema for an Item
 - 3.6.2 Example of METSRights Schema for a Collection
 - 3.6.3 Example of METSRights Schema for a Community

Makeup and Definition of AIPs

AIPs only store the Latest Version of Items

If you are using the new XMLUI-only [Item Level Versioning](#) functionality (disabled by default), you must be aware that this "Item Level Versioning" feature is **not yet compatible** with AIP Backup & Restore.

Using them together may result in accidental data loss. Currently the AIPs that DSpace generates only store the *latest version* of an Item. Therefore, past versions of Items will always be lost when you perform a restore / replace using AIP tools.

AIPs are Archival Information Packages.

- AIP is a package describing **one archival object** in DSpace.

- The **archival object** may be a single **Item, Collection, Community, or Site** (Site AIPs contain site-wide information). Bitstreams are included in an Item's AIP.
- Each AIP is logically self-contained, can be restored without rest of the archive. (So you could restore a single Item, Collection or Community)
- Collection or Community AIPs do **not** include all child objects (e.g. Items in those Collections or Communities), as each AIP only describes **one** object. However, these container AIPs do contain references (links) to all child objects. These references can be used by DSpace to automatically restore all referenced AIPs when restoring a Collection or Community.
- AIPs are only generated for objects which are currently in the "in archive" state in DSpace. This means that in-progress, uncompleted submissions are not described in AIPs and cannot be restored after a disaster. Permanently removed objects will also no longer be exported as AIPs after their removal. However, withdrawn objects will continue to be exported as AIPs, since they are still considered under the "in archive" status.
- AIPs with identical contents will always have identical [checksums](#). This provides a basic means of validating whether the contents within an AIP have changed. For example, if a Collection's AIP has the same checksum at two different points in time, it means that Collection has not changed during that time period.
- AIP profile favors completeness and accuracy rather than presenting the semantics of an object in a standard format. It conforms to the quirks of DSpace's internal object model rather than attempting to produce a universally understandable representation of the object. When possible, an AIP tries to use common standards to express objects.
- An AIP *can* serve as a DIP (Dissemination Information Package) or SIP (Submission Information Package), especially when transferring custody of objects to another DSpace implementation.
- In contrast to SIP or DIP, the AIP should include all available DSpace structural and administrative metadata, and basic provenance information. AIPs also describe some basic system level information (e.g. Groups and People).

General AIP Structure / Examples

Generally speaking, an AIP is an Zip file containing a METS manifest and all related content bitstreams, license files and any other associated files.

Some examples include:

- Site AIP (Sample: [SITE-example.zip](#))
 - METS contains basic metadata about DSpace Site and persistent IDs referencing all Top Level Communities
 - METS also contains a list of all Groups and EPeople information defined in the DSpace system. (NOTE: By default, user passwords are not stored in AIPs, unless you specify the 'passwords' flag. See [Additional Packager Options](#).)
- Community AIP (Sample: [COMMUNITY@123456789-1.zip](#))
 - METS contains all metadata for Community and persistent IDs referencing all members (SubCommunities or Collections). Package may also include a Logo file, if one exists.

- METS contains any Group information for Community-specific groups (e.g. `COMMUNITY_<ID>_ADMIN` group).
- METS contains all Community permissions/policies (translated into [METSRights schema](#))
- Collection AIP (Sample: [COLLECTION@123456789-2.zip](#))
 - METS contains all metadata for Collection and persistent IDs referencing all members (Items). Package may also include a Logo file, if one exists.
 - METS contains any Group information for Collection-specific groups (e.g. `COLLECTION_<ID>_ADMIN`, `COLLECTION_<ID>_SUBMIT`, etc.).
 - METS contains all Collection permissions/policies (translated into [METSRights schema](#))
 - If the Collection has an Item Template, the METS will also contain all the metadata for that Item Template.
- Item AIP (Sample: [ITEM@123456789-8.zip](#))
 - METS contains all metadata for Item and references to all Bundles and Bitstreams. Package also includes all Bitstream files.
 - METS contains all Item/Bundle/Bitstream permissions/policies (translated into [METSRights schema](#))

Notes:

- Bitstreams and Bundles are second-class archival objects; they are recorded in the context of an Item.
- BitstreamFormats are not even second-class; they are described implicitly within Item technical metadata, and reconstructed from that during restoration
- EPeople are only defined in Site AIP, but may be referenced from Community or Collection AIPs
- Groups may be defined in Site AIP, Community AIP or Collection AIP. Where they are defined depends on whether the Group relates specifically to a single Community or Collection, or is just a general site-wide group.

What is NOT in AIPs

- DSpace Site configurations (`[dspace]/config/` directory) or customizations (themes, stylesheets, etc) are not described in AIPs
- DSpace Database model (or customizations therein) is not described in AIPs
- Any objects which are not currently in the "In Archive" state are not described in AIPs. This means that in-progress, unfinished submissions are never included in AIPs.

Customizing What Is Stored in Your AIPs

If you choose, you can customize exactly what information is stored in your AIPs. However, you should be aware that you can only restore information which is stored within your AIPs. If you choose to remove information from your AIPs, you will be unable to restore it later on (unless you are also backing up your entire DSpace database and assetstore folder).




AIP Recommendations

It is recommended to minimally use the default settings when generating AIPs. DSpace can only restore information that is included within an AIP. Therefore, if you choose to no longer include some information in an AIP, DSpace will no longer be able to restore that information from an AIP backup

There are two ways to go about customizing your AIP format:

1. You can [customize your `dspace.cfg` settings pertaining to AIP generation](#). These configurations will allow you to specify exactly which DSpace Crosswalks will be called when generating the AIP METS manifest.
2. You can export your AIPs using one of the [special options/flags](#).

AIP Details: METS Structure

 This METS Structure is based on the structure decided for the original [AipPrototype](#), developed as part of the MIT & UCSD PLEDGE project.

- `mets` element
 - `@PROFILE` fixed value="http://www.dspace.org/schema/aip/1.0/mets.xsd" (this is how we identify an AIP manifest)
 - `@OBJID` URN-format persistent identifier (i.e. Handle) if available, or else a unique identifier. (e.g. "hdl:123456789/1")
 - `@LABEL` title if available
 - `@TYPE` DSpace object type, one of "DSpace ITEM", "DSpace COLLECTION", "DSpace COMMUNITY" or "DSpace SITE".
 - `@ID` is a globally unique identifier, built using the Handle and the Object type (e.g. `dspace-COLLECTION-hdl:123456789/3`).
- `mets/metsHdr` element
 - `@LASTMODDATE` last-modified date for a DSpace Item, or nothing for other objects.
 - `agent` element:
 - `@ROLE` = "CUSTODIAN",
 - `@TYPE` = "OTHER",
 - `@OTHERTYPE` = "DSpace Archive",
 - `name` = *Site handle*. (Note: The Site Handle is of the format `[handle_prefix]/0`, e.g. "123456789/0")
 - `agent` element:
 - `@ROLE` = "CREATOR",
 - `@TYPE` = "OTHER",
 - `@OTHERTYPE` = "DSpace Software",

- name = "DSpace [version]" (Where "[version]" is the specific version of DSpace software which created this AIP, e.g. "1.7.0")
- mets/dmdSec element(s)
 - By default, two dmdSec elements are included for all AIPs:
 1. object's descriptive metadata crosswalked to MODS (specified by mets/dmdSec/mdWrap@MDTYPE="MODS"). See [#MODS Schema](#) section below for more information.
 2. object's descriptive metadata in DSpace native DIM intermediate format, to serve as a complete and precise record for restoration or ingestion into another DSpace. Specified by mets/dmdSec/mdWrap@MDTYPE="OTHER", @OTHERMDTYPE="DIM". See [#DIM \(DSpace Intermediate Metadata\) Schema](#) section below for more information.
 - For Collection AIPs, additional dmdSec elements may exist which describe the Item Template for that Collection. Since an Item template is not an actual Item (i.e. it only includes metadata), it is stored within the Collection AIP. The Item Template's dmdSec elements will be referenced by a div @TYPE="DSpace ITEM Template" in the METS structMap.
 - When the mdWrap @TYPE value is OTHER, the element *MUST* include a value for the @OTHERMDTYPE attribute which names the crosswalk that produced (or interprets) that metadata, e.g. DIM.
- mets/amdSec element(s)
 - One or more amdSec elements are include for all AIPs. The first amdSec element contains administrative metadata (technical, source, rights, and provenance) for the entire archival object. Additional amdSec elements may exist to describe parts of the archival object (e.g. Bitstreams or Bundles in an Item).
 - techMD elements. By default, two types of techMD elements may be included:
 - PREMIS metadata about an object may be included here (*currently only specified for Bitstreams (files)*). Specified by mdWrap@MDTYPE="PREMIS". See [#PREMIS Schema](#) section below for more information.
 - DSPACE-ROLES metadata may appear here to describe the Groups or EPeople related to this object (*_currently only specified for Site, Community and Collection*). Specified by mdWrap@MDTYPE="OTHER", @OTHERMDTYPE="DSPACE-ROLES". See [#DSPACE-ROLES Schema](#) section below for more information.
 - rightsMD elements. By default, there are four possible types of rightsMD elements which may be included:
 - METSRights metadata may appear here to describe the permissions on this object. Specified by mdWrap@MDTYPE="OTHER", @OTHERMDTYPE="METSRIGHTS". See [#METSRights Schema](#) section below for more information.
 - DSpaceDepositLicense if the object is an Item and it has a deposit license, it is contained here. Specified by mdWrap@MDTYPE="OTHER", @OTHERMDTYPE="DSpaceDepositLicense".
 - CreativeCommonsRDF If the object is an Item with a Creative Commons license expressed in RDF, it is included here. Specified by mdWrap@MDTYPE="OTHER", @OTHERMDTYPE="CreativeCommonsRDF".

- `CreativeCommonsText` If the object is an Item with a Creative Commons license in plain text, it is included here. Specified by `mdWrap@MDTYPE="OTHER",@OTHERMDTYPE="CreativeCommonsText"`.
 - `sourceMD` element. By default, there is only one type of `sourceMD` element which may appear:
 - `AIP-TECHMD` metadata may appear here. This stores basic technical/source metadata about in object in a DSpace native format. Specified by `mdWrap@MDTYPE="OTHER",@OTHERMDTYPE="AIP-TECHMD"`. See [#AIP Technical Metadata Schema \(AIP-TECHMD\)](#) section below for more information.
 - `digiprovdMD` element.
 - *Not used at this time.*
- `mets/fileSec` element
 - For ITEM objects:
 - Each distinct Bundle in an Item goes into a `fileGrp`. The `fileGrp` has a `@USE` attribute which corresponds to the Bundle name.
 - Bitstreams in bundles become `file` elements under `fileGrp`.
 - `mets/fileSec/fileGrp/fileelements`
 - Set `@SIZE` to length of the bitstream. There is a redundant value in the `<techMD>` but it is more accessible here.
 - Set `@MIMETYPE`, `@CHECKSUM`, `@CHECKSUMTYPE` to corresponding bitstream values. There is redundant info in the `<techMD>`. (For DSpace, the `@CHECKSUMTYPE="MD5"` at all times)
 - SET `@SEQ` to bitstream's SequenceID if it has one.
 - SET `@ADMID` to the list of `<amdSec>` element(s) which describe this bitstream.
 - For COLLECTION and COMMUNITY objects:
 - *Only* if the object has a *logo bitstream*, there is a `fileSec` with one `fileGrp` child of `@USE="LOGO"`.
 - The `fileGrp` contains one `file` element, representing the logo Bitstream. It has the same `@MIMETYPE`, `@CHECKSUM`, `@CHECKSUMTYPE` attributes as the Item content bitstreams, but does NOT include metadata section references (e.g. `@ADMID`) or a `@SEQ` attribute.
 - See the main `structMap` for the `fptr` reference to this logo file.
- `mets/structMap` - Primary structure map, `@LABEL="DSpace Object",@TYPE="LOGICAL"`
 - For ITEM objects:
 1. Top-Level `div` with `@TYPE="DSpace Object Contents"`.
 - For every Bitstream in Item it contains a `div` with `@TYPE="DSpace BITSTREAM"`. Each Bitstream `div` has a single `fptr` element which references the bitstream location.
 - If Item has primary bitstream, put it in `structMap/div/fptr` (i.e. directly under the `div` with `@TYPE="DSpace Object Contents"`)
 - For COLLECTION objects:
 1. Top-Level `div` with `@TYPE="DSpace Object Contents"`.

- For every Item in the Collection, it contains a `div` with `@TYPE="DSpace ITEM"`. Each Item `div` has up to two child `mptrelements`:
 - a. One linking to the Handle of that Item. Its `@LOCTYPE="HANDLE"`, and `@xlink:href` value is the raw Handle.
 - b. (Optional) one linking to the location of the local AIP for that Item (if known). Its `@LOCTYPE="URL"`, and `@xlink:href` value is a relative link to the AIP file on the local filesystem.
- If Collection has a Logo bitstream, there is an `fptr` reference to it in the very first `div`.
- If the Collection includes an Item Template, there will be a `div` with `@TYPE="DSpace ITEM Template"` within the very first `div`. This `div` `@TYPE="DSpace ITEM Template"` must have a `@DMDID` specified, which links to the `dmdSec` element(s) that contain the metadata for the Item Template.
- For COMMUNITY objects:
 1. Top-Level `div` with `@TYPE="DSpace Object Contents"`.
 - For every Sub-Community in the Community it contains a `div` with `@TYPE="DSpace COMMUNITY"`. Each Community `div` has up to two `mptrelements`:
 - a. One linking to the Handle of that Community. Its `@LOCTYPE="HANDLE"`, and `@xlink:href` value is the raw Handle.
 - b. (Optional) one linking to the location of the local AIP file for that Community (if known). Its `@LOCTYPE="URL"`, and `@xlink:href` value is a relative link to the AIP file on the local filesystem.
 - For every Collection in the Community there is a `div` with `@TYPE="DSpace COLLECTION"`. Each Collection `div` has up to two `mptrelements`:
 - a. One linking to the Handle of that Collection. Its `@LOCTYPE="HANDLE"`, and `@xlink:href` value is the raw Handle.
 - b. (Optional) one linking to the location of the local AIP file for that Collection (if known). Its `@LOCTYPE="URL"`, and `@xlink:href` value is a relative link to the AIP file on the local filesystem.
 - If Community has a Logo bitstream, there is an `fptr` reference to it in the very first `div`.
- For SITE objects:
 1. Top-Level `div` with `@TYPE="DSpace Object Contents"`.
 - For every Top-level Community in Site, it contains a `div` with `@TYPE="DSpace COMMUNITY"`. Each Item `div` has up to two child `mptrelements`:
 - a. One linking to the Handle of that Community. Its `@LOCTYPE="HANDLE"`, and `@xlink:href` value is the raw Handle.
 - b. (Optional) one linking to the location of the local AIP for that Community (if known). Its `@LOCTYPE="URL"`, and `@xlink:href` value is a relative link to the AIP file on the local filesystem.
- `mets/structMap` - Structure Map to indicate object's Parent, `@LABEL="Parent"`, `@TYPE="LOGICAL"`

- Contains one `div` element which has the unique attribute value `TYPE="AIP Parent Link"` to identify it as the older of the *parent pointer*.
 - It contains a `mptr` element whose `xlink:href` attribute value is the raw Handle of the parent object, e.g. `1721.1/4321`.

Metadata in METS

The following tables describe how various metadata schemas are populated (via DSpace Crosswalks) in the METS file for an AIP.

DIM (DSpace Intermediate Metadata) Schema

[DIM Schema](#) is essentially a way of representing DSpace internal metadata structure in XML. DSpace's internal metadata is very similar to a Qualified Dublin Core in its structure, and is primarily meant for descriptive metadata. However, DSpace's metadata allows for custom elements, qualifiers or schemas to be created (so it is extendable to any number of schemas, elements, qualifiers). These custom fields/schemas may or may not be able to be translated into normal Qualified Dublin Core. So, the DIM Schema must be able to express metadata schemas, elements or qualifiers which may or may not exist within Qualified Dublin Core.

In the METS structure, DIM metadata always appears within a `dmdSec` inside an `<mdWrap MDTYPE="OTHER" OTHERMDTYPE="DIM">` element. For example:

```
<dmdSec ID="dmdSec_2190">
  <mdWrap MDTYPE="OTHER" OTHERMDTYPE="DIM">
    ...
  </mdWrap>
</dmdSec>
```

By default, DIM metadata is always included in AIPs. It is controlled by the following configuration in your `dspace.cfg`:

```
aip.disseminate.dmd = MODS, DIM
```

DIM Descriptive Elements for Item objects

As all DSpace Items already have user-assigned DIM (essentially Qualified Dublin Core) metadata fields, those fields are just exported into the [DIM Schema](#) within the METS file.

DIM Descriptive Elements for Collection objects

For Collections, the following fields are translated to the DIM schema:

DIM Metadata Field	Database field or value
dc.description	'introductory_text' field

DIM Metadata Field	Database field or value
dc.description.abstract	'short_description' field
dc.description.tableofcontents	'side_bar_text' field
dc.identifier.uri	Collection's handle
dc.provenance	'provenance_description' field
dc.rights	'copyright_text' field
dc.rights.license	'license' field
dc.title	'name' field

DIM Descriptive Elements for Community objects

For Communities, the following fields are translated to the DIM schema:

DIM Metadata Field	Database field or value
dc.description	'introductory_text' field
dc.description.abstract	'short_description' field
dc.description.tableofcontents	'side_bar_text' field
dc.identifier.uri	Handle of Community
dc.rights	'copyright_text' field
dc.title	'name' field

DIM Descriptive Elements for Site objects

For the Site Object, the following fields are translated to the DIM schema:

Metadata Field	Value
dc.identifier.uri	Handle of Site (format: [handle_prefix]/0)
dc.title	Name of Site (from dspace.cfg 'dspace.name' config)

MODS Schema

By default, all DSpace descriptive metadata (DIM) is also translated into the [MODS Schema](#) by utilizing DSpace's MODSDisseminationCrosswalk. DSpace's DIM to MODS crosswalk is defined within your [dspace] / config / crosswalks / mods.properties configuration file. This file allows you to customize the MODS that is included within your AIPs.

For more information on the MODS Schema, see <http://www.loc.gov/standards/mods/mods-schemas.html>

In the METS structure, MODS metadata always appears within a `dmdSec` inside an `<mdWrap MDTYPE="MODS">` element. For example:

```
<dmdSec ID="dmdSec_2189">
  <mdWrap MDTYPE="MODS">
    ...
  </mdWrap>
</dmdSec>
```

By default, MODS metadata is always included in AIPs. It is controlled by the following configuration in your `dspace.cfg`:

```
aip.disseminate.dmd = MODS, DIM
```

The MODS metadata is included within your AIP to support interoperability. It provides a way for other systems to interact with or ingest the AIP without needing to understand the DIM Schema. You may choose to disable MODS if you wish, however this may decrease the likelihood that you'd be able to easily ingest your AIPs into a non-DSpace system (unless that non-DSpace system is able to understand the DIM schema). When restoring/ingesting AIPs, DSpace will always first attempt to restore DIM descriptive metadata. Only if no DIM metadata is found, will the MODS metadata be used during a restore.

AIP Technical Metadata Schema (AIP-TECHMD)

The AIP Technical Metadata Schema is a way to translate technical metadata about a DSpace object into the [DIM Schema](#). It is kept separate from DIM as it is considered technical metadata rather than descriptive metadata.

In the METS structure, AIP-TECHMD metadata always appears within a `sourceMD` inside an `<mdWrap MDTYPE="OTHER" OTHERMDTYPE="AIP-TECHMD">` element. For example:

```
<amdSec ID="amd_2191">
  ...
  <sourceMD ID="sourceMD_2198">
    <mdWrap MDTYPE="OTHER" OTHERMDTYPE="AIP-TECHMD">
      ...
    </mdWrap>
  </sourceMD>
  ...
</amdSec>
```

By default, AIP-TECHMD metadata is always included in AIPs. It is controlled by the following configuration in your `dspace.cfg`:

```
aip.disseminate.sourceMD = AIP-TECHMD
```

AIP Technical Metadata for Item

Metadata Field	Value
dc.contributor	Submitter's email address
dc.identifier.uri	Handle of Item
dc.relation.isPartOf	Owning Collection's Handle (<i>as a URN</i>)
dc.relation.isReferencedBy	All other Collection's this item is linked to (<i>Handle URN of each non-owner</i>)
dc.rights.accessRights	" <i>WITHDRAWN</i> " if item is withdrawn

AIP Technical Metadata for Bitstream

Metadata Field	Value
dc.title	Bitstream's name/title
dc.title.alternative	Bitstream's source
dc.description	Bitstream's description
dc.format	Bitstream Format Description
dc.format.medium	Short Name of Format
dc.format.mimetype	MIMEType of Format
dc.format.supportlevel	System Support Level for Format (necessary to recreate Format during restore, if the format isn't know to DSpace by default)
dc.format.internal	Whether Format is internal (necessary to recreate Format during restore, if the format isn't know to DSpace by default)

- Outstanding Question: Why are we recording the file format support status? That's a DSpace property, rather than an Item property. Do DSpace instances rely on objects to tell them their support status?
 - Possible answer (from Larry Stone): Format support and other properties of the BitstreamFormat are recorded here in case the Item is restored in an empty DSpace that doesn't have that format yet, and the relevant bits of the format entry have to be reconstructed from the AIP. --lcs

AIP Technical Metadata for Collection

Metadata Field	Value
dc.identifier.uri	Handle of Collection

Metadata Field	Value
dc.relation.isPartOf	Owning Community's Handle (<i>as a URM</i>)
dc.relation.isReferencedBy	All other Communities this Collection is linked to (<i>Handle URN of each non-owner</i>)

AIP Technical Metadata for Community

Metadata Field	Value
dc.identifier.uri	Handle of Community
dc.relation.isPartOf	Handle of Parent Community (<i>as a URM</i>)

AIP Technical Metadata for Site

Metadata Field	Value
dc.identifier.uri	Site Handle (format: [handle_prefix]/0)

PREMIS Schema

At this point in time, the [PREMIS Schema](#) is only used to represent technical metadata about DSpace Bitstreams (i.e. Files). The PREMIS metadata is generated by DSpace's `PREMISCrosswalk`. Only the [PREMIS Object Entity Schema](#) is used.

In the METS structure, PREMIS metadata always appears within a `techMD` inside an `<mdWrap MDTYPE="PREMIS">` element. PREMIS metadata is **always** wrapped within a `<premis:premis>` element. For example:

```

<amdSec ID="amd_2209">
  ...
  <techMD ID="techMD_2210">
    <mdWrap MDTYPE="PREMIS">
      <premis:premis>
        ...
      </premis:premis>
    </mdWrap>
  </techMD>
  ...
</amdSec>

```

Each Bitstream (file) has its own `amdSec` within a METS manifest. So, there will be a separate PREMIS `techMD` for each Bitstream within a single Item.

By default, PREMIS metadata is always included in AIPs. It is controlled by the following configuration in your `dspace.cfg`:

```
aip.disseminate.techMD = PREMIS, DSPACE-ROLES
```

PREMIS Metadata for Bitstream

The following Bitstream information is translated into PREMIS for each DSpace Bitstream (file):

Metadata Field	Value
<premis:objectIdentifier>	Contains Bitstream direct URL
<premis:objectCategory>	Always set to "File"
<premis:fixity>	Contains MD5 Checksum of Bitstream
<premis:format>	Contains File Format information of Bistream
<premis:originalName>	Contains original name of file

DSPACE-ROLES Schema

All DSpace Groups and EPeople objects are translated into a custom `DSPACE-ROLES` XML Schema. This XML Schema is a very simple representation of the underlying DSpace database model for Groups and EPeople. The `DSPACE-ROLES` Schemas is generated by DSpace's `RoleCrosswalk`.

Only the following DSpace Objects utilize the `DSPACE-ROLES` Schema in their AIPs:

- Site AIP – all Groups and EPeople are represented in `DSPACE-ROLES` Schema
- Community AIP – only Community-based groups (e.g. `COMMUNITY_1_ADMIN`) are represented in `DSPACE-ROLES` Schema
- Collection AIP – only Collection-based groups (e.g. `COLLECTION_2_ADMIN`, `COLLECTION_2_SUBMIT`, etc.) are represented in `DSPACE-ROLES` Schema

In the METS structure, `DSPACE-ROLES` metadata always appears within a `techMD` inside an `<mdWrap` `MDTYPE="OTHER" OTHERMDTYPE="DSPACE-ROLES">` element. For example:

```
<amdSec ID="amd_2068">
  ...
  <techMD ID="techMD_2070">
    <mdWrap MDTYPE="OTHER" OTHERMDTYPE="DSPACE-ROLES">
      ...
    </mdWrap>
  </techMD>
  ...
</amdSec>
```

By default, `DSPACE-ROLES` metadata is always included in AIPs. It is controlled by the following configuration in your `dspace.cfg`:

```
aip.disseminate.techMD = PREMIS, DSPACE-ROLES
```

Example of DSPACE-ROLES Schema for a SITE AIP

Below is a general example of the structure of a DSPACE-ROLES XML file, as it would appear in a SITE AIP.

```
<DSpaceRoles>
  <Groups>
    <Group ID="1" Name="Administrator">
      <Members>
        <Member ID="1" Name="bsmith@myu.edu" />
      </Members>
    </Group>
    <Group ID="0" Name="Anonymous" />
    <Group ID="70" Name="COLLECTION_hdl:123456789/57_ADMIN">
      <Members>
        <Member ID="1" Name="bsmith@myu.edu" />
      </Members>
    </Group>
    <Group ID="75" Name="COLLECTION_hdl:123456789/57_DEFAULT_READ">
      <MemberGroups>
        <MemberGroup ID="0" Name="Anonymous" />
      </MemberGroups>
    </Group>
    <Group ID="71" Name="COLLECTION_hdl:123456789/57_SUBMIT">
      <Members>
        <Member ID="1" Name="bsmith@myu.edu" />
      </Members>
    </Group>
    <Group ID="72" Name="COLLECTION_hdl:123456789/57_WORKFLOW_STEP_1">
      <MemberGroups>
        <MemberGroup ID="1" Name="Administrator" />
      </MemberGroups>
    </Group>
    <Group ID="73" Name="COLLECTION_hdl:123456789/57_WORKFLOW_STEP_2">
      <MemberGroups>
        <MemberGroup ID="1" Name="Administrator" />
      </MemberGroups>
    </Group>
    <Group ID="8" Name="COLLECTION_hdl:123456789/6703_DEFAULT_READ" />
    <Group ID="9" Name="COLLECTION_hdl:123456789/2_ADMIN">
      <Members>
        <Member ID="1" Name="bsmith@myu.edu" />
      </Members>
    </Group>
  </Groups>
  <People>
    <Person ID="1">
      <Email>bsmith@myu.edu</Email>
      <Netid>bsmith</Netid>
      <FirstName>Bob</FirstName>
    </Person>
  </People>
</DSpaceRoles>
```

```
<LastName>Smith</LastName>
<Language>en</Language>
<CanLogin />
</Person>
<Person ID="2">
  <Email>jjones@myu.edu</Email>
  <FirstName>Jane</FirstName>
  <LastName>Jones</LastName>
  <Language>en</Language>
  <CanLogin />
  <SelfRegistered />
</Person>
</People>
</DSpaceRoles>
```

Why are there Group Names with Handles?

You may have noticed several odd looking group names in the above example, where a Handle is embedded in the name (e.g. "COLLECTION_hdl:123456789/57_SUBMIT"). This is a translation of a Group name which included a Community or Collection *Internal ID* (e.g. "COLLECTION_45_SUBMIT"). Since you are exporting these Groups outside of DSpace, the *Internal ID* may no longer be valid or be understandable. Therefore, before export, these Group names are all translated to include an externally understandable identifier, in the form of a Handle. If you use this AIP to restore your groups later, they will be translated back to the normal DSpace format (i.e. the handle will be translated back to the new *Internal ID*).

Orphaned Groups are Renamed on Export

If a Group name includes a Community or Collection *Internal ID* (e.g. "COLLECTION_45_SUBMIT"), and that Community or Collection no longer exists, then the Group is considered "Orphaned".

- In 1.8.2 and above, the Group is renamed using the following format: "ORPHANED_[object-type]_GROUP_[obj-id]_[group-type]" (e.g. "ORPHANED_COLLECTION_GROUP_10_ADMIN").
- Prior to 1.8.2, the Group was renamed with a random key: "GROUP_[random-hex-key]_[object-type]_[group-type]" (e.g. "GROUP_123eb3a_COLLECTION_ADMIN"). *This old format was discontinued as giving the groups a randomly generated name caused the SITE AIP to have a different checksum every time it was regenerated (see [DS-1120](#)).*

The reasoning is that we were unable to translate an *Internal ID* into an *External ID* (i.e. Handle). If we are unable to do that translation, re-importing or restoring a group with an *old* internal ID could cause conflicts or instability in your DSpace system. In order to avoid such conflicts, these groups are renamed using a random, unique key.

Example of DSPACE-ROLES Schema for a Community or Collection

Below is a general example of the structure of a DSPACE-ROLES XML file, as it would appear in a Community or Collection AIP.

This specific example is for a Collection, which has associated Administrator, Submitter, and Workflow approver groups. In this very simple example, each group only has one Person as a member of it. Please notice that the Person's information (Name, NetID, etc) is NOT contained in this content (however they are available in the DSPACE-ROLES example for a SITE, as shown above)

```
<DSpaceRoles>
  <Groups>
    <Group ID="9" Name="COLLECTION_hdl:123456789/2_ADMIN" Type="ADMIN">
      <Members>
        <Member ID="1" Name="bsmith@myu.edu" />
      </Members>
    </Group>
    <Group ID="13" Name="COLLECTION_hdl:123456789/2_SUBMIT" Type="SUBMIT">
      <Members>
        <Member ID="2" Name="jjones@myu.edu" />
      </Members>
    </Group>
    <Group ID="10" Name="COLLECTION_hdl:123456789/2_WORKFLOW_STEP_1" Type="WORKFLOW_STEP_1">
      <Members>
        <Member ID="1" Name="bsmith@myu.edu" />
      </Members>
    </Group>
    <Group ID="11" Name="COLLECTION_hdl:123456789/2_WORKFLOW_STEP_2" Type="WORKFLOW_STEP_2">
      <Members>
        <Member ID="2" Name="jjones@myu.edu" />
      </Members>
    </Group>
    <Group ID="12" Name="COLLECTION_hdl:123456789/2_WORKFLOW_STEP_3" Type="WORKFLOW_STEP_3">
      <Members>
        <Member ID="1" Name="bsmith@myu.edu" />
      </Members>
    </Group>
  </Groups>
</DSpaceRoles>
```

METSRights Schema

All DSpace Policies (permissions on objects) are translated into the [METSRights schema](#). This is different than the above DSPACE-ROLES schema, which only represents Groups and People objects. Instead, the METSRights schema is used to translate the permission statements (e.g. a group named "Library Admins" has Administrative permissions on a Community named "University Library"). But the METSRights schema doesn't represent who is a member of a particular group (that is defined in the DSPACE-ROLES schema, as described above).



METSRights should always be used with DSPACE-ROLES

The METSRights Schema must be used in conjunction with the DSPACE-ROLES Schema for Groups, People and Permissions to all be restored properly. As mentioned above, the METSRights metadata can only be used to restore permissions (i.e. DSpace policies). The DSPACE-ROLES metadata must also exist if you wish to restore the actual Group or EPeople objects to which those permissions apply.

All DSpace Object's AIPs (except for the SITE AIP) utilize the METSRights Schema in order to define what permissions people and groups have on that object. Although there are several sections to the METSRights Schema, DSpace AIPs *only use* the <RightsDeclarationMD> section, as this is what is used to describe rights on an object.

In the METS structure, METSRights metadata always appears within a `rightsMD` inside an <mdWrap MDTYPE="OTHER" OTHERMDTYPE="METSRIGHTS"> element. For example:

```
<amdSec ID="amd_2068">
  ...
  <rightsMD ID="rightsMD_2074">
    <mdWrap MDTYPE="OTHER" OTHERMDTYPE="METSRIGHTS">
      ...
    </mdWrap>
  </rightsMD>
  ...
</amdSec>
```

By default, METSRights metadata is always included in AIPs. It is controlled by the following configuration in your `dspace.cfg`:

```
aip.disseminate.rightsMD = DSpaceDepositLicense:DSPACE_DEPLICENSE, \
  CreativeCommonsRDF:DSPACE_CCRDF, CreativeCommonsText:DSPACE_CCTEXT, METSRIGHTS
```

Example of METSRights Schema for an Item

An Item AIP will almost always contain several METSRights metadata sections within its METS Manifest. A separate METSRights metadata section is used to describe the permissions on:

- the Item itself
- each Bundle (group of files) in the Item
- each Bitstream (file) within an Item's bundle

Below is an example of a METSRights sections for a publicly visible Bitstream, Bundle or Item. Notice it specifies that the "GENERAL PUBLIC" has the permission to DISCOVER or DISPLAY this object.

```
<rights:RightsDeclarationMD xmlns:rights="http://cosimo.stanford.edu/sdr/metsrights/"
RIGHTSCATEGORY="LICENSED">
  <rights:Context CONTEXTCLASS="GENERAL_PUBLIC">
    <rights:Permissions DISCOVER="true" DISPLAY="true" MODIFY="false" DELETE="false" />
  </rights:Context>
</rights:RightsDeclarationMD>
```

As of DSpace 3, DSpace policies/permissions may also have a "start-date" or "end-date" (to support [Embargo](#) functionality). Such a policy on an Item may look like this. Notice it specifies that the "GENERAL PUBLIC" has the permission to DISCOVER or DISPLAY this object *starting on* 2015-01-01, while the Group "Staff" has permission to DISCOVER or DISPLAY this object *until* 2015-01-01.

```
<rights:RightsDeclarationMD xmlns:rights="http://cosimo.stanford.edu/sdr/metsrights/"
RIGHTSCATEGORY="LICENSED">
  <rights:Context CONTEXTCLASS="GENERAL_PUBLIC" start-date="2015-01-01" in-effect="false">
    <rights:Permissions DISCOVER="true" DISPLAY="true" MODIFY="false" DELETE="false" />
  </rights:Context>
  <rights:Context CONTEXTCLASS="MANAGED_GRP" end-date="2015-01-01" in-effect="true">
    <rights:UserName USERTYPE="GROUP">Staff</rights:UserName>
    <rights:Permissions DISCOVER="true" DISPLAY="true" MODIFY="false" DELETE="false" />
  </rights:Context>
</rights:RightsDeclarationMD>
```

Example of METSRights Schema for a Collection

A Collection AIP contains one METSRights section, which describes the permissions different Groups or People have within the Collection

Below is an example of a METSRights sections for a publicly visible Collection, which also has an Administrator group, a Submitter group, and a group for each of the three DSpace workflow approval steps. You'll notice that each of the groups is provided with very specific permissions within the Collection. Submitters & Workflow approvers can "ADD CONTENTS" to a collection (but cannot delete the collection). Administrators have full rights.

```
<rights:RightsDeclarationMD xmlns:rights="http://cosimo.stanford.edu/sdr/metsrights/"
RIGHTSCATEGORY="LICENSED">
  <rights:Context CONTEXTCLASS="MANAGED_GRP">
    <rights:UserName USERTYPE="GROUP">COLLECTION_hdl:123456789/2_SUBMIT</rights:UserName>
    <rights:Permissions DISCOVER="true" DISPLAY="true" MODIFY="true" DELETE="false" OTHER="true"
OTHERPERMITTYPE="ADD CONTENTS" />
  </rights:Context>
  <rights:Context CONTEXTCLASS="MANAGED_GRP">
    <rights:UserName USERTYPE="GROUP">COLLECTION_hdl:123456789/2_WORKFLOW_STEP_3</rights:UserName>
    <rights:Permissions DISCOVER="true" DISPLAY="true" MODIFY="true" DELETE="false" OTHER="true"
OTHERPERMITTYPE="ADD CONTENTS" />
  </rights:Context>
  <rights:Context CONTEXTCLASS="MANAGED_GRP">
    <rights:UserName USERTYPE="GROUP">COLLECTION_hdl:123456789/2_WORKFLOW_STEP_2</rights:UserName>
```

```

    <rights:Permissions DISCOVER="true" DISPLAY="true" MODIFY="true" DELETE="false" OTHER="true"
OTHERPERMITTYPE="ADD CONTENTS" />
  </rights:Context>
  <rights:Context CONTEXTCLASS="MANAGED_GRP">
    <rights:UserName USERTYPE="GROUP">COLLECTION_hdl:123456789/2_WORKFLOW_STEP_1</rights:UserName>
    <rights:Permissions DISCOVER="true" DISPLAY="true" MODIFY="true" DELETE="false" OTHER="true"
OTHERPERMITTYPE="ADD CONTENTS" />
  </rights:Context>
  <rights:Context CONTEXTCLASS="MANAGED_GRP">
    <rights:UserName USERTYPE="GROUP">COLLECTION_hdl:123456789/2_ADMIN</rights:UserName>
    <rights:Permissions DISCOVER="true" DISPLAY="true" COPY="true" DUPLICATE="true" MODIFY="true"
DELETE="true" PRINT="true" OTHER="true" OTHERPERMITTYPE="ADMIN" />
  </rights:Context>
  <rights:Context CONTEXTCLASS="GENERAL_PUBLIC">
    <rights:Permissions DISCOVER="true" DISPLAY="true" MODIFY="false" DELETE="false" />
  </rights:Context>
</rights:RightsDeclarationMD>

```

Example of METSRights Schema for a Community

A Community AIP contains one METSRights section, which describes the permissions different Groups or People have within that Community.

Below is an example of a METSRights sections for a publicly visible Community, which also has an Administrator group. As you'll notice, this content looks very similar to the Collection METSRights section (as described above)

```

<rights:RightsDeclarationMD xmlns:rights="http://cosimo.stanford.edu/sdr/metsrights/"
RIGHTSCATEGORY="LICENSED">
  <rights:Context CONTEXTCLASS="MANAGED_GRP">
    <rights:UserName USERTYPE="GROUP">COMMUNITY_hdl:123456789/10_ADMIN</rights:UserName>
    <rights:Permissions DISCOVER="true" DISPLAY="true" COPY="true" DUPLICATE="true" MODIFY="true"
DELETE="true" PRINT="true" OTHER="true" OTHERPERMITTYPE="ADMIN" />
  </rights:Context>
  <rights:Context CONTEXTCLASS="GENERAL_PUBLIC">
    <rights:Permissions DISCOVER="true" DISPLAY="true" MODIFY="false" DELETE="false" />
  </rights:Context>
</rights:RightsDeclarationMD>

```

5.3 Ant targets and options

- [1 Options](#)
- [2 Targets](#)



Ant targets should be run as the service user

A word of warning: in order to ensure proper permissions and file ownership are maintained, you are advised to run these ant targets as the service user (commonly 'dspace' or 'tomcat'). This is a change for DSpace 5.0. Running them as any other user is likely to cause problems, especially with the new Solr index maintenance targets.

5.3.1 Options

DSpace allows three property values to be set using the `-D<property>=<value>` option. They may be used in other contexts than noted below, but take care to understand how a particular property will affect a target's outcome.

overwrite

Whether to overwrite configuration files in `[dspace]/config`. If true, files from `[dspace]/config` and subdirectories are backed up with `.old` extension and new files are installed from `[dspace-src]/dspace/config` and subdirectories; if false, existing config files are untouched, and new files are written beside them with `.new` extension.

Possible values:	true, false
Default:	true
Context:	update, init_configs

config

If a path is specified, ant uses values from the specified file and installs it in `[dspace]/config` in the appropriate contexts.

Possible values:	path to configuration file to be used
Default:	<code>[dspace-src]/config/dspace.cfg</code>
Context:	update, update_configs, update_code, update_webapps, init_configs, fresh_install, test_database, setup_database, load_registries, clean_database

wars

If true, builds `.war` files; if false, no `.war` files are built.

Possible values:	true, false
Default:	true

Context:	update, update_webapps, fresh_install
----------	---------------------------------------

5.3.2 Targets

Target	Effect
update	Creates backup copies of the [dspace]/bin, /etc, /lib, and /webapps directories with the form /<directory>.bak-<date-time>. Creates new copies of [dspace]/config, /etc, and /lib directories. Does not affect data files or the database. (See <i>overwrite</i> , <i>config</i> , <i>war</i> options.)
update_configs	Updates the [dspace]/config directory with new configuration files. (See <i>config</i> option.)
update_geolite	Download and install GeoCity database into [dspace]/config.
update_code	Creates backup copies of the [dspace]/bin, /etc, and /lib directories with the form /<directory>.bak-<date-time>. Creates new copies of [dspace]/config, /etc, and /lib directories. (See <i>config</i> option.)
update_webapps	Updates [dspace]/webapps directory. (See <i>config</i> , <i>war</i> options.)
update_solr_indexes	Checks if any Solr indexes need upgrading (to latest Solr), and if so, upgrades them.
init_configs	Writes configuration files to [dspace]/config. (See <i>overwrite</i> , <i>config</i> options.)
install_code	Deletes existing [dspace]/bin, /lib, and /etc directories, and installs new copies; overwrites /solr application files, leaving data intact. (See <i>config</i> option.)
fresh_install	Performs a fresh installation of the software, including the database & config. (See <i>config</i> , <i>war</i> options.)
test_database	Tests database connection using parameters specified in dspace.cfg. (See <i>config</i> option.)
setup_database	Creates database tables. Database schema must exist and relevant parameters specified in dspace.cfg. (See <i>config</i> option.)
load_registries	Loads metadata & file format registries into the database. (See <i>config</i> option.)
clean_backups	Removes [dspace]/bin, /etc, /lib, and /webapps directories with .bak* extensions.
clean_database	Drops all DSpace database tables, destroying all data. (See <i>config</i> option.)

5.4 Command Line Operations

- [1 Executing command line operations](#)
- [2 Available operations](#)
 - [2.1 General use](#)
 - [2.2 Legacy statistics](#)
 - [2.3 SOLR Statistics](#)

The DSpace command launcher or CLI interface offers the execution of different maintenance operations. As most of these are already documented in related parts of the documentation, this page is mainly intended to provide an overview of all available CLI operations, with links to the appropriate documentation.

5.4.1 Executing command line operations

The CLI interface is found at `[dspace]/bin/dspace`. Execute it without arguments or with the `-h` option to see all available operations. Execute `dspace op -h` to see details about the `op` operation.

Examples:

```
bin/dspace -h
bin/dspace cleanup -h
bin/dspace cleanup
bin/dspace cleanup --verbose
```

5.4.2 Available operations

General use

- [checker](#): Run the checksum checker
- [checker-emailer](#): Send emails related to the checksum checker
- [classpath](#): Calculate and display the DSpace classpath
- [cleanup](#): Remove deleted bitstreams from the assetstore
- [community-filiator](#): Tool to manage community and sub-community relationships
- [create-administrator](#): Create a DSpace administrator account
- [curate](#): Perform curation tasks on DSpace objects
- [database](#): Perform various tasks / checks of the DSpace database
- [doi-organisier](#): Transmit information about DOIs to the registration agency.
- [dsprop](#): View a DSpace property from `dspace.cfg`
- [dsrun](#): Run a class directly
- [embargo-lifter](#): Pre DSpace 3.0 embargo manager tool used to check, list and lift embargoes
- [export](#): Export items or collections

- [filter-media](#): Perform the media filtering to extract full text from documents and to create thumbnails
- [generate-sitemaps](#): Generate search engine and html sitemaps
- [harvest](#): Manage the OAI-PMH harvesting of external collections
- [import](#): Import items into DSpace
- [index-db-browse](#): General index command (requires extra parameters)
- [index-discovery](#): Update Discovery Solr Search Index
- [index-lucene-init](#): Initialise the search and browse indexes
- [index-lucene-update](#): Update the search and browse indexes
- [itemcounter](#): Update the item strength counts in the user interface
- [itemupdate](#): Item update tool for altering metadata and bitstream content in items
- [make-handle-config](#): Run the handle server simple setup command
- [metadata-export](#): Export metadata for batch editing
- [metadata-import](#): Import metadata after batch editing
- [migrate-embargo](#): Embargo manager tool used to migrate old version of Embargo to the new one included in dspace3
- [oai](#): OAI script manager
- [packager](#): Execute a packager
- [rdfizer](#): tool to convert contents to RDF
- [read](#) : execute a stream of commands from a file or pipe
- [registry-loader](#): Load entries into a registry
- [setup-database](#): Create the database tables
- [structure-builder](#): Build DSpace community and collection structure
- [sub-daily](#): Send daily subscription notices
- ~~[test-database](#): Test the DSpace database connection~~ (Replaced by "database test" command.)
- [test-email](#): Test the DSpace email server settings are OK
- [update-handle-prefix](#): Update handle records and metadata when moving from one Handle prefix to another
- [user](#): Create, List, Update, Delete EPerson (user) records
- [validate-date](#): Test date-time format rules
- [version](#): Show DSpace version and other troubleshooting information

Legacy statistics

- [stat-general](#): Compile the general statistics
- [stat-initial](#): Compile the initial statistics
- [stat-monthly](#): Compile the monthly statistics
- [stat-report-general](#): Create the general statistics report
- [stat-report-initial](#): Create the initial statistics report
- [stat-report-monthly](#): Create the monthly statistics report

SOLR Statistics

Scripts for the statistics that are stored in SOLR, added in DSpace 1.6.

- [stats-log-converter](#): Convert dspace.log files ready for import into solr statistics
- [stats-log-importer](#): Import previously converted log files into solr statistics
- [stats-log-importer-elasticsearch](#): Import solr-format converted log files into Elastic Search statistics
- [stats-util](#): Statistics Client for Maintenance of Solr Statistics Indexes

5.4.3 Executing streams of commands

You can pass a sequence of commands into the `dspace` command-line tool using the `read` command.

Execute commands...	this way
...from a file	<code>[dspace]/bin/dspace read a-command-file</code>
...in a pipeline	<code>some-other-command [dspace]/bin/dspace read -</code> <code>some-other-command [dspace]/bin/dspace read</code>


5.4.4 Database Utilities

This command can be used at any time to manage or upgrade the Database. It will also assist in troubleshooting PostgreSQL and Oracle connection issues with the database.

Valid Arguments	Description
:	
Command used:	<code>[dspace]/bin/dspace database</code>
Java class	<code>org.dspace.storage.rdbms.DatabaseUtils</code>
:	
<code>test</code>	Test the database connection settings (in <code>[dspace]/config/dspace.cfg</code>) are OK and working properly
<code>info</code>	Provide detailed information about the DSpace database itself. This includes the database type, version, driver, schema, and any successful/failed/pending database migrations. This command, along with "test", is very useful in debugging issues with your database.
<code>migrate</code>	

Valid Arguments	Description
:	<p>Migrate the database to the latest version (if not already on the latest version). This uses FlywayDB along with embedded migrations scripts to automatically update your database to the latest version.</p> <p>Optionally, you can run "migrate ignored" to also include any database migrations which are flagged as "Ignored" by the "info" command.</p>
repair	<p>Attempt to "repair" any migrations which are flagged as "Failed" by the "info" command. This uses the FlywayDB repair command.</p> <p>Please note however, this will NOT automatically repair corrupt or broken data in your database. It merely tries to re-run previously "Failed" migrations.</p>
clean	<p>Completely and permanently delete all tables and data in this database. WARNING: There is no turning back! If you run this command, you will lose your entire database and all its contents.</p> <p>This command is only useful for testing or for reverting your database to a "fresh install" state (e.g. running "dspace database clean" followed by "dspace database migrate" will return your database to a fresh install state)</p>

5.5 Legacy methods for re-indexing content

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

- [1 Overview](#)
- [2 Re-Enabling the legacy Lucene Search and/or DBMS Browse providers](#)
 - [2.1 Configure search and browse to use PostgreSQL](#)
 - [2.2 Configure search and browse to use Oracle](#)
- [3 Creating the Browse & Search Indexes](#)
- [4 Running the Indexing Programs](#)
 - [4.1 Complete Index Regeneration](#)
 - [4.2 Updating the Indexes](#)
 - [4.3 Destroy and Rebuild Browse Tables](#)
- [5 Indexing Customization](#)

- [5.1 Browse Index Customization](#)
- [5.2 Search Index Customization](#)
 - [5.2.1 Configuring Lucene Search Indexes](#)
 - [5.2.2 Customize the advanced search form](#)

5.5.1 Overview

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


1. Faceted/Filtered Search & Browse (via Solr & [DSpace Discovery](#)) - **enabled** by default since DSpace 4.0
2. Traditional Browse & Search (via Lucene & Database tables) - this is **disabled** by default, it was used in older versions of dspace and is being phased out

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

5.5.2 Re-Enabling the legacy Lucene Search and/or DBMS

Browse providers

The old Lucene/DB search and browse will still work, but will need to be configured to work. The three options are SOLR (default), or Postgres/Lucene, or OracleDB/Lucene. When you set the value to these configs, ensure that at most only one provider is chosen.

 If a DAOs configuration is not provided the system will default to using the SOLR Browse Engine

Configure search and browse to use PostgreSQL

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

Alter dspace.cfg to set the browseDAO to postgres.

browseDAO - Postgres

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

Alter dspace.cfg to have ItemCount use Postgres

ItemCount - Postgres

```
ItemCountDAO.class = org.dspace.browse.ItemCountDAOPostgres
```

Configure search and browse to use Oracle

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

Alter dspace.cfg to set the browseDAO to Oracle

browseDAO - Oracle

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

Alter dspace.cfg to have ItemCount use Oracle

ItemCount - Oracle

```
ItemCountDAO.class = org.dspace.browse.ItemCountDAOOracle
```

5.5.3 Creating the Browse & Search Indexes

It is possible that the database tables to hold the search and browse data do not exist, so they must be created:


Create the DB search/browse tables

```
[dspace]/bin/dspace index-db-browse
```

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


Command used:	[dspace]/bin/dspace index-db-browse
Java class:	org.dspace.browse.IndexBrowse
Arguments short and long forms):	Description
-r or -rebuild	Should we rebuild all the indexes, which removes old tables and creates new ones. For use with -f. Mutually exclusive with -d
-s or -start	-s <int> start from this index number and work upwards (mostly only useful for debugging). For use with -t and -f

<code>-x</code> or <code>-execute</code>	Execute all the remove and create SQL against the database. For use with <code>-t</code> and <code>-f</code> .
<code>-i</code> or <code>-index</code>	Actually do the indexing. Mutually exclusive with <code>-t</code> and <code>-f</code> .
<code>-o</code> or <code>-out</code>	<code>-o <filename></code> write the remove and create SQL to the given file. For use with <code>-t</code> and <code>-f</code>
<code>-p</code> or <code>-print</code>	Write the remove and create SQL to the stdout. For use with <code>-t</code> and <code>-f</code> .
<code>-t</code> or <code>-tables</code>	Create the tables only, do no attempt to index. Mutually exclusive with <code>-f</code> and <code>-i</code>
<code>-f</code> or <code>-full</code>	Make the tables, and do the indexing. This forces <code>-x</code> . Mutually exclusive with <code>-f</code> and <code>-i</code> .
<code>-v</code> or <code>-verbose</code>	Print extra information to the stdout. If used in conjunction with <code>-p</code> , you cannot use the stdout to generate your database structure.
<code>-d</code> or <code>-delete</code>	Delete all the indexes, but do not create new ones. For use with <code>-f</code> . This is mutually exclusive with <code>-r</code> .
<code>-h</code> or <code>-help</code>	Show this help documentation. Overrides all other arguments.



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

5.5.4 Running the Indexing Programs

Complete Index Regeneration


Requires that you stop Tomcat first

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


Known Oracle Issues

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

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

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

Updating the Indexes

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

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

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

Destroy and Rebuild Browse Tables

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

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

At the CLI screen:

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

5.5.5 Indexing Customization

Browse Index Customization

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

Through customization it is possible to:

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

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

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


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

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



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

Search Index Customization

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

Configuring Lucene Search Indexes

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

Property:	<code>search.dir</code>
Example Value:	<code>search.dir = \${dspace.dir}/search</code>
Informational Note:	Where to put the search index files
Property:	<code>search.max-clauses</code>
Example Value:	<code>search.max-clauses = 2048</code>
Informational Note:	By setting higher values of <code>search.max-clauses</code> will enable prefix searches to work on larger repositories.
Property:	<code>search.index.delay</code>
Example Value:	<code>search.index.delay = 5000</code>
Informational Note:	It is possible to create a 'delayed index flusher'. If a web application pushes multiple search requests (i.e. a barrage or sword deposits, or multiple quick edits in the user interface), then this will combine them into a single index update. You set the property key to the number of milliseconds to wait for an update. The example value will hold a Lucene update in a queue for up to 5 seconds. After 5 seconds all waiting updates will be written to the Lucene index.
Property:	<code>search.analyzer</code>

Example Value:	<code>search.analyzer = org.dspace.search.DSAnalyzer</code>
Informational Note:	Which Lucene Analyzer implementation to use. If this is omitted or commented out, the standard DSpace analyzer (designed for English) is used by default. This standard DSpace analyzer removes common stopwords, lowercases all words and performs stemming (removing common word endings, like "ing", "s", etc).
Property:	<code>search.analyzer</code>
Example Value:	<code>search.analyzer = org.dspace.search.DSNonStemmingAnalyzer</code>
Informational Note:	Instead of the standard DSpace Analyzer (DSAnalyzer), use an analyzer which doesn't "stem" words/terms. When using this analyzer, a search for "wellness" will always return items matching "wellness" and not "well". However, similarly a search for "experiments" will only return objects matching "experiments" and not "experiment" or "experimenting". When using this analyzer, you may still use WildCard searches like "experiment*" to match the beginning of words.
Property:	<code>search.analyzer</code>
Example Value:	<code>search.analyzer = org.apache.lucene.analysis.cn.ChineseAnalyzer</code>
Informational Note:	Instead of the standard English analyzer, the Chinese analyzer is used.
Property:	<code>search.operator</code>
Example Value:	<code>search.operator = OR</code>
Informational Note:	Boolean search operator to use. The currently supported values are OR and AND. If this configuration item is missing or commented out, OR is used. AND requires all the search terms to be present. OR requires one or more search terms to be present.
Property:	<code>search.maxfieldlength</code>
Example Value:	<code>search.maxfieldlength = 10000</code>
Informational Note:	This is the maximum number of terms indexed for a single field in Lucene. The default is 10,000 words, often not enough for full-text indexing. If you change this, you will need to re-index for the change to take effect on previously added items. -1 = unlimited (Integer.MAG_VALUE)

Property:	<code>search.index.n</code>
Example Value:	<code>search.index.1 = author:dc.contributor.*</code>
Informational Note	This property determines which of the metadata fields are being indexed for search. As an example, if you do not include the title field here, searching for a word in the title will not be matched with the titles of your items..

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

```

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

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

<id>	is an incremental number to distinguish each search index entry
<search index name>	is the identifier for the search field this index will correspond to
<schema>	is the schema used. Dublin Core (DC) is the default. Others are possible.
<metadata field>	is the DSpace metadata field to be indexed.
<index type>	<p>can be used to specify how manipulate the values before indexing.</p> <p><i>Example: <code>search.index.12 = language:dc.language.iso:inputform</code></i></p> <p>Possible values are:</p> <p>text - default, no special treatment. Metadata value are passed to lucene as text</p>

	<p>timestamp - the values are interpreted as date with second granularity. An additional index postfixed with <code>.year</code> is created with year granularity</p> <p>date - the values are interpreted as date with day granularity. An additional index postfixed with <code>.year</code> is created with year granularity</p> <p>inputform - in addition to the values stored in the metadata the displayed form of this value as derivable from the input-form (in any of the available languages) are stored</p>
--	---

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

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

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

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

Authority Control Note:

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

Customize the advanced search form

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

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

Property:	<code>jspui.search.index.display.<n></code>
Example Value	<code>jspui.search.index.display.1 = ANY</code>

Informational Note:	Set the <i>N</i> -value of the index dropdown in the advanced search form. The value must match one of the defined index
---------------------	--

5.6 Mediafilters for Transforming DSpace Content

- 1 [MediaFilters: Transforming DSpace Content](#)
 - 1.1 [Overview](#)
 - 1.2 [Available Media Filters](#)
 - 1.3 [Enabling/Disabling MediaFilters](#)
 - 1.4 [Executing \(via Command Line\)](#)
 - 1.5 [Creating Custom MediaFilters](#)
 - 1.5.1 [Creating a simple Media Filter](#)
 - 1.5.2 [Creating a Dynamic or "Self-Named" Format Filter](#)
 - 1.6 [Configuration parameters](#)

5.6.1 MediaFilters: Transforming DSpace Content

Overview

DSpace can apply filters or transformations to files/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 `dspace filter-media` script which traverses the asset store, invoking all configured `MediaFilter` or `FormatFilter` classes on files/bitstreams (see [Configuring Media Filters](#) for more information on how they are configured).

Available Media Filters

Below is a listing of all currently available Media Filters, and what they actually do:

Name	Java Class	Function	Enabled by Default ?
HTML Text Extractor	<code>org.dspace.app.mediafilter.HTMLFilter</code>	extracts the full text of HTML documents for full text indexing. (true

Name	Java Class	Function	Enabled by Default ?
		Uses Swing's HTML Parser)	
JPEG Thumbnail	<code>org.dspace.app.mediafilter.JPEGFilter</code>	creates thumbnail images of GIF, JPEG and PNG files	true
Branded Preview JPEG	<code>org.dspace.app.mediafilter.BrandedPreviewJPEGFilter</code>	creates a branded preview image for GIF, JPEG and PNG files	false
PDF Text Extractor	<code>org.dspace.app.mediafilter.PDFFilter</code>	extracts the full text of Adobe PDF documents (only if text-based or OCR'd) for full text indexing. (Uses the Apache PDFBox tool)	true
XPDF Text Extractor	<code>org.dspace.app.mediafilter.XPDF2Text</code>	extracts the full text of Adobe PDF documents (only if	false

Name	Java Class	Function	Enabled by Default ?
		text-based or OCRred) for full text indexing (Uses the XPDF command line tools available for Unix.) See XPDF Filter Configuration for details on installing/ enabling.	
Word Text Extractor	org.dspace.app.mediafilter.WordFilter	extracts the full text of Microsoft Word or Plain Text documents for full text indexing. (Uses the " Microsoft Word Text Mining " tools .)	true
PowerPoint Text Extractor	org.dspace.app.mediafilter.PowerPointFilter	extracts the full text of slides and notes in Microsoft PowerPoint and PowerPoint	true

Name	Java Class	Function	Enabled by Default ?
		XML documents for full text indexing (Uses the Apache POI tools.)	
ImageMagick Image Thumbnail Generator	org.dspace.app.mediafilter.ImageMagickImageThumbnailFilter	uses ImageMagick to generate thumbnails for image bitstreams. Requires installation of ImageMagick on your server. See ImageMagick Media Filters .	false
ImageMagick PDF Thumbnail Generator	org.dspace.app.mediafilter.ImageMagickPdfThumbnailFilter	uses ImageMagick and Ghostscript to generate thumbnails for PDF bitstreams. Requires installation of ImageMagick and Ghostscript on your server. See	false

Name	Java Class	Function	Enabled by Default ?
		ImageMagick Media Filters	

Please note that the `filter-media` script will automatically update the DSpace search index by default (see [Legacy methods for re-indexing content](#)) This is the recommended way to run these scripts. But, should you wish to disable it, you can pass the `-n` flag to either script to do so (see [Executing \(via Command Line\)](#) below).

Enabling/Disabling MediaFilters

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). By modifying the value of `filter.plugins` you can disable or enable MediaFilter plugins.

Executing (via Command Line)

The media filter system is intended to be run from the command line (or regularly as a cron task):

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

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

Available Command-Line Options:

- **Help** : `[dspace]/bin/dspace filter-media -h`
 - Display help message describing all command-line options.
- **Force mode** : `[dspace]/bin/dspace filter-media -f`
 - Apply filters to ALL bitstreams, even if they've already been filtered. If they've already been filtered, the previously filtered content is overwritten.
- **Identifier mode** : `[dspace]/bin/dspace filter-media -i 123456789/2`
 - Restrict processing to the community, collection, or item named by the identifier - by default, all bitstreams of all items in the repository are processed. The identifier must be a Handle, not a DB key. This option may be combined with any other option.
- **Maximum mode** : `[dspace]/bin/dspace filter-media -m 1000`
 - Suspend operation after the specified maximum number of items have been processed - by default , no limit exists. This option may be combined with any other option.
- **No-Index mode** : `[dspace]/bin/dspace filter-media -n`

- Suppress index creation - by default, a new search index is created for full-text searching. This option suppresses index creation if you intend to run `index-update` elsewhere.
- **Plugin mode**: `[dspace]/bin/dspace filter-media -p "PDF Text Extractor", "Word Text Extractor"`
 - Apply ONLY the filter plugin(s) listed (separated by commas). By default all named filters listed in the `filter.plugins` field of `dspace.cfg` are applied. This option may be combined with any other option. *WARNING*: multiple plugin names must be separated by a comma (i.e. ',') and NOT a comma followed by a space (i.e. ', ').
- **Skip mode**: `[dspace]/bin/dspace filter-media -s 123456789/9,123456789/100`
 - SKIP the listed identifiers (separated by commas) during processing. The identifiers must be Handles (not DB Keys). They may refer to items, collections or communities which should be skipped. This option may be combined with any other option. *WARNING*: multiple identifiers must be separated by a comma (i.e. ',') and NOT a comma followed by a space (i.e. ', ').
 - NOTE: If you have a large number of identifiers to skip, you may maintain this comma-separated list within a separate file (e.g. `filter-skiplist.txt`). Use the following format to call the program. *Please note the use of the "grave" or "tick" (`) symbol and do not use the single quotation.*
 - `[dspace]/bin/dspace filter-media -s `less filter-skiplist.txt``
- **Verbose mode**: `[dspace]/bin/dspace filter-media -v`
 - Verbose mode - print all extracted text and other filter details to STDOUT. Adding your own filters is done by creating a class which *implements* the `org.dspace.app.mediafilter.FormatFilter` interface. See the [Creating a new Media/Format Filter](#) topic and comments in the source file `FormatFilter.java` for more information. In theory filters could be implemented in any programming language (C, Perl, etc.) However, they need to be invoked by the Java code in the Media Filter class that you create.

Creating Custom MediaFilters

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
```

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 described in Chapter 13.3.2.2 .

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 Chapter 13.3.2.2 *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).

 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 parameters

Property	<code>filter.org.dspace.app.mediafilter.publicPermission</code>
Example Value	<code>filter.org.dspace.app.mediafilter.publicPermission = JPEGFilter, XPDF2Thumbnail</code>
Informational Note	By default mediafilter derivatives / thumbnails inherit the same permissions of the parent bitstream, but you can override this, in case you want to make publicly accessible derivative / thumbnail content, typically the thumbnails of objects for the browse list. List the MediaFilter name's that would get public accessible permissions. Any media filters not listed will instead inherit the permissions of the parent bitstream.

5.6.2 ImageMagick Media Filters

- 1 [ImageMagick Media Filters](#)
 - 1.1 [Overview](#)
 - 1.2 [Installation](#)
 - 1.3 [DSpace Configuration](#)
 - 1.3.1 [Thumbnail Dimensions](#)
 - 1.3.2 [Conversion Utility Path](#)
 - 1.3.3 [Overwriting Existing Thumbnails](#)
 - 1.4 [Additional Customization](#)

Overview

The ImageMagick Media Filters provide consistent, high quality thumbnails for image bitstreams and PDF bitstreams.

These filters require a separate software installation of the conversion utilities (ImageMagick and Ghostscript).

The media filters use the library [im4java](#) to invoke the conversion utilities. This library constructs a conversion command launches a sub-process to perform the generation of media files.

Installation

- Install [ImageMagick](#) on your server
- If you wish to generate PDF thumbnails, install [Ghostscript](#) on your server
- The ImageMagick and Ghostscript executables should be accessible from the same directory (e.g. `/usr/bin`)

DSpace Configuration

In the `filter.plugins` section your `dspace.cfg` file, uncomment the ImageMagick media filter definition.

```
ImageMagick Image Thumbnail, ImageMagick PDF Thumbnail, \
```

This will activate the following settings which are already present in `dspace.cfg`

```
org.dspace.app.mediafilter.ImageMagickImageThumbnailFilter = ImageMagick Image Thumbnail, \  
org.dspace.app.mediafilter.ImageMagickPdfThumbnailFilter = ImageMagick PDF Thumbnail
```

These media filters contain the following configuration properties

Thumbnail Dimensions

The following properties are used to define the dimensions of the generated thumbnails.

```
# maximum width and height of generated thumbnails
thumbnail.maxwidth = 80
thumbnail.maxheight = 80
```

Conversion Utility Path

The following property provides a path to the ImageMagick and GhostScript utilities.

```
org.dspace.app.mediafilter.ImageMagickThumbnailFilter.ProcessStarter = /usr/bin
```

Overwriting Existing Thumbnails

The ImageMagick media filters can differentiate thumbnails created by the DSpace default thumbnail generator and thumbnails that were manually uploaded by a user. The media filter reads the bitstream description field to make this determination. A regular expression can be provided to define the set of thumbnails that should be overwritten by the ImageMagick thumbnail generator. Thumbnail descriptions matching this pattern will be overwritten even if the `-f` option is not passed to the filter media process.

```
org.dspace.app.mediafilter.ImageMagickThumbnailFilter.replaceRegex = ^Generated Thumbnail$
```

The ImageMagick media filter will use the bitstream description field to identify bitstreams that it has created using the following setting. Bitstreams containing this label will be overwritten only if the `-f` filter is applied.

```
org.dspace.app.mediafilter.ImageMagickThumbnailFilter.bitstreamDescription = IM Thumbnail
```

Thumbnail descriptions that do not match either of the patterns listed above are presumed to be manually uploaded thumbnails. These thumbnails will not be replaced even if the `-f` option is passed to the filter media process.

Additional Customization


The ImageMagick conversion software provides a large number of conversion options. Subclasses of these media filters could be written to take advantage of the additional conversion properties available in the software.

Note: The PDF thumbnail generator is hard-coded to generate a thumbnail from the first page of the PDF.

5.7 Performance Tuning DSpace

- 1 [Give Tomcat \(DSpace UIs\) More Memory](#)

- 1.1 [Give Tomcat More Java Heap Memory](#)
- 1.2 [Give Tomcat More Java PermGen Memory](#)
- 2 [Choosing the size of memory spaces allocated to DSpace](#)
- 3 [Give the Command Line Tools More Memory](#)
 - 3.1 [Give the Command Line Tools More Java Heap Memory](#)
 - 3.2 [Give the Command Line Tools More Java PermGen Space Memory](#)
- 4 [Give PostgreSQL Database More Memory](#)
- 5 [SOLR Statistics Performance Tuning](#)

 The software DSpace relies on does not come out of the box optimized for large repositories. Here are some tips to make it all run faster.

5.7.1 Give Tomcat (DSpace UIs) More Memory

Give Tomcat More Java Heap Memory

Java Heap Memory Recommendations

At the time of writing, DSpace recommends you should give Tomcat \geq 512MB of Java Heap Memory to ensure optimal DSpace operation. Most larger sized or highly active DSpace installations however tend to allocate more like 1024MB to 2048MB of Java Heap Memory.

Performance tuning in Java basically boils down to memory. If you are seeing "`java.lang.OutOfMemoryError: Java heap space`" errors, this is a sure sign that Tomcat isn't being provided with enough Heap Memory.

Tomcat is especially memory hungry, and will benefit from being given lots of RAM. To set the amount of memory available to Tomcat, use either the `JAVA_OPTS` or `CATALINA_OPTS` environment variable, e.g:

```
CATALINA_OPTS=-Xmx512m -Xms512m
```

OR

```
JAVA_OPTS=-Xmx512m -Xms512m
```

The above example sets the maximum Java Heap memory to 512MB.



Difference between JAVA_OPTS and CATALINA_OPTS

You can use either environment variable. JAVA_OPTS is also used by other Java programs (besides just Tomcat). CATALINA_OPTS is *only used* by Tomcat. So, if you only want to tweak the memory available to Tomcat, it is recommended that you use CATALINA_OPTS. If you set **both** CATALINA_OPTS and JAVA_OPTS, Tomcat will default to using the settings in CATALINA_OPTS.

If the machine is dedicated to DSpace a decent rule of thumb is to give tomcat half of the memory on your machine. **At a minimum, you should give Tomcat >= 512MB of memory for optimal DSpace operation.** (*NOTE: As your DSpace instance gets larger in size, you may need to increase this number to the several GB range.*) The latest guidance is to also set -Xms to the same value as -Xmx for server applications such as Tomcat.

Give Tomcat More Java PermGen Memory



Java PermGen Memory Recommendations

At the time of writing, DSpace recommends you should give Tomcat >= 128MB of PermGen Space to ensure optimal DSpace operation.

If you are seeing "java.lang.OutOfMemoryError: PermGen space" errors, this is a sure sign that Tomcat is running out PermGen Memory. (More info on PermGen Space: http://blogs.sun.com/fkieviet/entry/classloader_leaks_the_dreaded_java)

To increase the amount of PermGen memory available to Tomcat (default=64MB), use either the JAVA_OPTS or CATALINA_OPTS environment variable, e.g:

```
CATALINA_OPTS=-XX:MaxPermSize=128m
```

OR

```
JAVA_OPTS=-XX:MaxPermSize=128m
```


The above example sets the maximum PermGen memory to 128MB.




Difference between JAVA_OPTS and CATALINA_OPTS

You can use either environment variable. JAVA_OPTS is also used by other Java programs (besides just Tomcat). CATALINA_OPTS is *only used* by Tomcat. So, if you only want to tweak the memory

available to Tomcat, it is recommended that you use `CATALINA_OPTS`. If you set **both** `CATALINA_OPTS` and `JAVA_OPTS`, Tomcat will default to using the settings in `CATALINA_OPTS`.

 Please note that you can obviously set **both** Tomcat's Heap space and PermGen Space together similar to:

```
CATALINA_OPTS=-Xmx512m -Xms512m -XX:MaxPermSize=128m
```

 On an Ubuntu machine (10.04) at least, the file `/etc/default/tomcat6` appears to be the best place to put these environmental variables.

5.7.2 Choosing the size of memory spaces allocated to DSpace

psi-probe is a webapp that can be deployed in DSpace and be used to watch memory usage of the other webapps deployed in the same instance of Tomcat (in our case, the DSpace webapps).

1. Download the latest version of psi-probe from <https://code.google.com/p/psi-probe/>
2. Unzip probe.war into `[dspace]/webapps/`

```
cd [dspace]/webapps/
wget https://psi-probe.googlecode.com/files/probe-2.3.3.zip
unzip probe-2.3.3.zip
unzip probe.war -d probe
```

3. Add a Context element in Tomcat's configuration (`in` or `in`) and make it privileged (so that it can monitor the other webapps):

EITHER in `$(CATALINA_HOME)/conf/server.xml`

```
<Context docBase="[dspace]/webapps/probe" privileged="true" path="/probe" />
```

OR in `$(CATALINA_HOME)/conf/Catalina/localhost/probe.xml`

```
<Context docBase="[dspace]/webapps/probe" privileged="true" />
```

4. Edit `$(CATALINA_HOME)/conf/tomcat-users.xml` to add a user for login into psi-probe (see more in <https://code.google.com/p/psi-probe/wiki/InstallationApacheTomcat#Security>)

```
<?xml version='1.0' encoding='utf-8'?>
```

```
<tomcat-users>
  <user username="admin" password="t0psecret" roles="manager" />
</tomcat-users>
```

5. Restart Tomcat
6. Open <http://yourdspace.com:8080/probe/> (edit domain and port number as necessary) in your browser and use the username and password from tomcat-users.xml to log in.

In the " System Information " tab, go to the " Memory utilization " menu. Note how much memory Tomcat is using upon startup and use a slightly higher value than that for the `-Xms` parameter (initial Java heap size). Watch how big the various memory spaces get over time (hours or days), as you run various common DSpace tasks that put load on memory, including indexing, reindexing, importing items into the oai index etc. These maximum values will determine the `-Xmx` parameter (maximum Java heap size). Watching PS Perm Gen grow over time will let you choose the value for the `-XX:MaxPermSize` parameter.

5.7.3 Give the Command Line Tools More Memory

Give the Command Line Tools More Java Heap Memory

Similar to Tomcat, you may also need to give the DSpace Java-based command-line tools more Java Heap memory. If you are seeing "java.lang.OutOfMemoryError: Java heap space" errors, when running a command-line tool, this is a sure sign that it isn't being provided with enough Heap Memory.

By default, DSpace only provides 256MB of maximum heap memory to its command-line tools.

If you'd like to provide **more** memory to command-line tools, you can do so via the `JAVA_OPTS` environment variable (which is used by the `[dspace]/bin/dspace` script). Again, it's the same syntax as above:

```
JAVA_OPTS=-Xmx512m -Xms512m
```

This is especially useful for big batch jobs, which may require additional memory.



You can also edit the `[dspace]/bin/dspace` script and add the environmental variables to the script directly.

Give the Command Line Tools More Java PermGen Space Memory

Similar to Tomcat, you may also need to give the DSpace Java-based command-line tools more PermGen Space. If you are seeing "java.lang.OutOfMemoryError: PermGen space" errors, when running a command-line tool, this is a sure sign that it isn't being provided with enough PermGen Space.

By default, Java only provides 64MB of maximum PermGen space.

If you'd like to provide **more** PermGen Space to command-line tools, you can do so via the `JAVA_OPTS` environment variable (which is used by the `[dspace] / bin / dspace` script). Again, it's the same syntax as above:

```
JAVA_OPTS=-XX:MaxPermSize=128m
```

This is especially useful for big batch jobs, which may require additional memory.



Please note that you can obviously set **both** Java's Heap space and PermGen Space together similar to:

```
JAVA_OPTS=-Xmx512m -Xms512m -XX:MaxPermSize=128m
```

5.7.4 Give PostgreSQL Database More Memory

On many linux distros PostgreSQL comes out of the box with an incredibly conservative configuration - it uses only 8Mb of memory! To put some more fire in its belly edit the `shared_buffers` parameter in `postgresql.conf`. The memory usage is 8KB multiplied by this value. The advice in the Postgres docs is not to increase it above 1/3 of the memory on your machine.



For More PostgreSQL Tips

For more hints/tips with PostgreSQL configurations and performance tuning, see also:

- [PostgresPerformanceTuning](#)
- [PostgresqlConfiguration](#)

5.7.5 SOLR Statistics Performance Tuning

[This @mire article](#) covers two different methods to enhance performance for the SOLR statistics, that are part of DSpace 1.6 and newer versions.

Note that the **Auto Commit** method is already integrated in DSpace 1.7 and above.

5.8 Scheduled Tasks via Cron

Several DSpace features **require** that a script is run regularly (via cron, or similar). Some of these features include:

- the [e-mail subscription feature](#) that alerts users of new items being deposited;
- the '[media filter](#)' tool, that generates thumbnails of images and extracts the full-text of documents for indexing;
- the '[checksum checker](#)' that tests the bitstreams in your repository for corruption;
- the [sitemap generator](#), which enhances the ability of major search engines to index your content and make it findable;
- the [curation system queueing feature](#), which allows administrators to "queue" tasks (to run at a later time) from the Admin UI;
- and [Discovery](#) (search & browse), [OAI-PMH](#) and [Usage Statistics](#) all receive performance benefits from regular re-optimization.

These regularly scheduled tasks should be setup via either [cron](#) (for Linux/Mac OSX) or [Windows Task Scheduler](#) (for Windows).

5.8.1 Recommended Cron Settings

If you are on Linux or Mac OSX, you should add these cron settings under the OS account which is running DSpace (or Tomcat). For example, login as that user and type the following to edit the user's crontab.

```
crontab -e
```

While every DSpace installation is unique, in order to get the most out of DSpace, we highly recommend enabling these basic cron settings (the settings are described in the comments):

```
## SAMPLE CRONTAB FOR A PRODUCTION DSPACE
## You obviously may wish to tweak this for your own installation,
## but this should give you an idea of what you likely wish to schedule via cron.
##
## NOTE: You may also need to add additional sysadmin related tasks to your crontab
## (e.g. zipping up old log files, or even removing old logs, etc).
#-----
# GLOBAL VARIABLES
#-----
# Full path of your local DSpace Installation (e.g. /home/dspace or /dspace or similar)
# MAKE SURE TO CHANGE THIS VALUE!!!
DSPACE = [dspace]
# Shell to use
SHELL=/bin/sh
```

```

# Add all major 'bin' directories to path
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
# Set JAVA_OPTS with defaults for DSpace Cron Jobs.
# Only provides 512MB of memory by default (which should be enough for most sites).
JAVA_OPTS="-Xmx512M -Xms512M -Dfile.encoding=UTF-8"
#-----
# HOURLY TASKS (Recommended to be run multiple times per day, if possible)
# At a minimum these tasks should be run daily.
#-----
# Regenerate DSpace Sitemaps every 8 hours (12AM, 8AM, 4PM).
# SiteMaps ensure that your content is more findable in Google, Google Scholar, and other major
search engines.
0 0,8,16 * * * $DSPACE/bin/dspace generate-sitemaps > /dev/null
#-----
# DAILY TASKS
# (Recommended to be run once per day. Feel free to tweak the scheduled times below.)
#-----
# Update the OAI-PMH index with the newest content (and re-optimize that index) at midnight every
day
# NOTE: ONLY NECESSARY IF YOU ARE RUNNING OAI-PMH
# (This ensures new content is available via OAI-PMH and ensures the OAI-PMH index is optimized for
better performance)
0 0 * * * $DSPACE/bin/dspace oai import -o > /dev/null
# Clean and Update the Discovery indexes at midnight every day
# (This ensures that any deleted documents are cleaned from the Discovery search/browse index)
0 0 * * * $DSPACE/bin/dspace index-discovery > /dev/null
# Re-Optimize the Discovery indexes at 12:30 every day
# (This ensures that the Discovery Solr Index is re-optimized for better performance)
30 0 * * * $DSPACE/bin/dspace index-discovery -o > /dev/null
# Cleanup Web Spiders from DSpace Statistics Solr Index at 01:00 every day
# NOTE: ONLY NECESSARY IF YOU ARE RUNNING SOLR STATISTICS
# (This removes any known web spiders from your usage statistics)
0 1 * * * $DSPACE/bin/dspace stats-util -i
# Re-Optimize DSpace Statistics Solr Index at 01:30 every day
# NOTE: ONLY NECESSARY IF YOU ARE RUNNING SOLR STATISTICS
# (This ensures that the Statistics Solr Index is re-optimized for better performance)
30 1 * * * $DSPACE/bin/dspace stats-util -o
# Send out subscription e-mails at 02:00 every day
# (This sends an email to any users who have "subscribed" to a Collection, notifying them of newly
added content.)
0 2 * * * $DSPACE/bin/dspace sub-daily
# Run the media filter at 03:00 every day.
# (This task ensures that thumbnails are generated for newly add images,
# and also ensures full text search is available for newly added PDF/Word/PPT/HTML documents)
0 3 * * * $DSPACE/bin/dspace filter-media
# Run any Curation Tasks queued from the Admin UI at 04:00 every day
# (Ensures that any curation task that an administrator "queued" from the Admin UI is executed
# asynchronously behind the scenes)
0 4 * * * $DSPACE/bin/dspace curate -q admin_ui
#-----
# WEEKLY TASKS
# (Recommended to be run once per week, but can be run more or less frequently, based on your local
needs/policies)

```



```

#-----
# Run the checksum checker at 04:00 every Sunday
# By default it runs through every file (-l) and also prunes old results (-p)
# (This re-verifies the checksums of all files stored in DSpace. If any files have been changed/
# corrupted, checksums will differ.)
0 4 * * * $DSpace/bin/dspace checker -l -p
# NOTE: LARGER SITES MAY WISH TO USE DIFFERENT OPTIONS. The above "-l" option tells DSpace to check
# everything*.
# If your site is very large, you may need to only check a portion of your content per week. The
# below commented-out task
# would instead check all the content it can within *one hour*. The next week it would start again
# where it left off.
#0 4 * * 0 $DSpace/bin/dspace checker -d 1h -p

# Mail the results of the checksum checker (see above) to the configured "mail.admin" at 05:00
# every Sunday.
# (This ensures the system administrator is notified whether any checksums were found to be
# different.)
0 5 * * 0 $DSpace/bin/dspace checker-emailer
#-----
# MONTHLY TASKS
# (Recommended to be run once per month, but can be run more or less frequently, based on your
# local needs/policies)
#-----
# Permanently delete any bitstreams flagged as "deleted" in DSpace, on the first of every month at
# 01:00
# (This ensures that any files which were deleted from DSpace are actually removed from your local
# filesystem.
# By default they are just marked as deleted, but are not removed from the filesystem.)
0 1 1 * * $DSpace/bin/dspace cleanup > /dev/null
#-----
# YEARLY TASKS (Recommended to be run once per year)
#-----
# At 2:00AM every January 1, "shard" the DSpace Statistics Solr index.
# This ensures each year has its own Solr index, which improves performance.
# NOTE: ONLY NECESSARY IF YOU ARE RUNNING SOLR STATISTICS
# NOTE: This is scheduled here for 2:00AM so that it happens *after* the daily cleaning &
# re-optimization of this index.
0 2 1 1 * $DSpace/bin/dspace stats-util -s

```

5.9 Search Engine Optimization

5.9.1 Ensuring your DSpace is indexed

Anyone who has analyzed traffic to their DSpace site (e.g. using Google Analytics or similar) will notice that a significant (and in many cases a majority) of visitors arrive via a search engine such as Google or Yahoo. Hence , to help maximize the impact of content and thus encourage further deposits, it is important to ensure that your DSpace instance is indexed effectively.

DSpace comes with tools that ensure major search engines (Google, Bing, Yahoo, Google Scholar) are able to easily and effectively index all your content. However, many of these tools provide some basic setup. Here's how to ensure your site is indexed.

For the optimum indexing, you should:

1. [Keep your DSpace up to date](#). We are constantly adding new indexing improvements in new releases
2. [Ensure your DSpace is visible to search engines](#).
3. [Enable the sitemaps feature](#) – this does not require e.g. registering with Google Webmaster tools.
4. [Ensure your robots.txt allows access to item "splash" pages and full text](#).
5. [Ensure item metadata appears in HTML headers correctly](#).
6. [Avoid redirecting file downloads to Item landing pages](#)
7. As an aside, it's worth noting that [OAI-PMH is generally not useful to search engines](#). OAI-PMH has its own uses, but do not expect search engines to use it.

Keep your DSpace up to date

We are constantly adding new indexing improvements to DSpace. In order to ensure your site gets all of these improvements, you should strive to keep it up-to-date. For example:

- As of DSpace 5.0, the DSpace robots.txt file now includes references to [Sitemaps](#) by default (see [DS-1936](#)), and also blocks known bad bots (see [DS-2335](#)).
- As of DSpace 4.0, DSpace has provided several enhancements, which were requested by the Google Scholar team. These included providing users (and web indexers) a way to browse content by the date it was added to DSpace (see [DS-1482](#)), ensuring the "dc.date.issued" field is set more accurately (see [DS-1481](#)), and enhancing the logic behind the "citation_pdf_url" HTML <meta> tag (see [DS-1483](#))
- As of DSpace 1.7, DSpace has improved how its Item-level metadata is made available to Google Scholar. For the 1.7.0 release, the DSpace Developers worked directly with the Google Scholar developers, to ensure DSpace is generating the "citation_*" HTML "<meta>" tags (i.e. Highwire Press tags) that Google Scholar recommends in their [Indexing Guidelines](#).
- As of DSpace 1.5, DSpace has support for sitemaps (both simple HTML pages of links, as well as the [sitemaps.org protocol](#)). It also includes item metadata in the HTML HEAD element of item display pages, ensuring that the metadata can be effectively indexed no matter what changes you might have made to your DSpace's layout or style.
- As of DSpace 1.4, DSpace has support for the "if-modified-since" HTTP header. This basically means that if an item (or bitstream therein) has not changed since the last time a search engine's crawler indexed it, that item/bitstream does not have to be re-retrieved, sparing your server.

Additional minor improvements / bug fixes have been made to more recent releases of DSpace.

Ensure your DSpace is visible to search engines

First ensure your DSpace instance is visible, e.g. with: <https://www.google.com/webmasters/tools/sitestatus>

If your site is not indexed at all, all search engines have a way to add your URL, e.g.:

- Google: <http://www.google.com/addurl>
- Yahoo: <http://siteexplorer.search.yahoo.com/submit>
- Bing: <http://www.bing.com/docs/submit.aspx>

Enable the sitemaps feature

DSpace provides a sitemap feature that we **highly recommend** you enable to ensure proper indexing. Sitemaps allow DSpace to expose its content in a way that makes it easily accessible to search engine crawlers. Sitemaps also help ensure that crawlers do NOT have to visit every page in your DSpace (which means the crawlers can get in and get out quickly, without taxing your site). Without sitemaps, search engine indexing activity may impose significant loads on your repository.

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 enable sitemaps, all you need to do is run `[dspace]/bin/dspace generate-sitemaps` once a day.

Just set up a cron job (or scheduled task in Windows), e.g. (cron):

```
# Regenerate sitemaps at 6:00 AM local time each morning
0 6 * * * [dspace]/bin/dspace generate-sitemaps
```

Once you've enabled your sitemaps, they will be accessible at the following URLs:

- XML Sitemaps / Sitemaps.org syntax: `[dspace.url]/sitemap`
- HTML Sitemaps: `[dspace.url]/htmlmap`

So, for example, if your "dspace.url" = <http://mysite.org/xmlui> in your "dspace.cfg" configuration file, then the HTML Sitemaps would be at: "<http://mysite.org/xmlui/htmlmap>"

Make your sitemap discoverable to search engines

Even if you've enabled your sitemaps, search engines may not be able to find them unless you provide them with a link. There are two main ways to notify a search engine of your sitemaps:

1. **Provide a hidden link to the sitemaps in your DSpace's homepage.** If you've customized your site's look and feel (as most have), ensure that there is a link to `/htmlmap` in your DSpace's front or home page. *By default, both the JSPUI and XMLUI provide this link in the footer.*

```
<a href="/htmlmap"></a>
```

- 2. Announce your sitemap in your robots.txt.** Most major search engines will also automatically discover your sitemap if you announce it in your robots.txt file. *By default, both the JSPUI and XMLUI provide these references in their robots.txt file.* For example:

```
# The FULL URL to the DSpace sitemaps
# XML sitemap is listed first as it is preferred by most search engines
# Make sure to replace "[dspace.url]" with the value of your 'dspace.url' setting in your
dspace.cfg file.
Sitemap: [dspace.url]/sitemap
Sitemap: [dspace.url]/htmlmap
```

- These "Sitemap:" lines can be placed anywhere in your robots.txt file. You can also specify multiple "Sitemap:" lines, so that search engines can locate both formats. For more information, see: <http://www.sitemaps.org/protocol.html#informing>
- Be sure to include the FULL URL in the "Sitemap:" line. Relative paths are not supported.

Search engines will now look at your XML and HTML sitemaps, which serve pre-generated (and thus served with minimal impact on your hardware) XML or HTML files linking directly to items, collections and communities in your DSpace instance. Crawlers will not have to work their way through any browse screens, which are intended more for human consumption, and more expensive for the server.

Create a good robots.txt

The trick here is to minimize load on your server, but without actually blocking anything vital for indexing. Search engines need to be able to index item, collection and community pages, and all bitstreams within items – full-text access is critically important for effective indexing, e.g. for citation analysis as well as the usual keyword searching.

If you have restricted content on your site, search engines will not be able to access it; they access all pages as an anonymous user.

Ensure that your robots.txt file is at the top level of your site: i.e. at <http://repo.foo.edu/robots.txt>, and NOT e.g. <http://repo.foo.edu/dspace/robots.txt>. If your DSpace instance is served from e.g. <http://repo.foo.edu/dspace/>, you'll need to add /dspace to all the paths in the examples below (e.g. /dspace/browse-subject).

NEVER BLOCK THESE PATHS

Some URLs can be disallowed without negative impact, but be ABSOLUTELY SURE the following URLs can be reached by crawlers, i.e. DO NOT put these on Disallow: lines, or your DSpace instance might not be indexed properly.

- /bitstream
- /browse (UNLESS USING SITEMAPS)
- /*/browse (UNLESS USING SITEMAPS)
- /browse-date (UNLESS USING SITEMAPS)

- /*/browse-date (UNLESS USING SITEMAPS)
- /community-list (UNLESS USING SITEMAPS)
- /handle
- /html
- /htmlmap

Example good robots.txt

Below is an example good robots.txt. The highly recommended settings are uncommented. Additional, optional settings are displayed in comments – based on your local configuration you may wish to enable them by uncommenting the corresponding "Disallow:" line.

```
# The FULL URL to the DSpace sitemaps
# XML sitemap is listed first as it is preferred by most search engines
# Make sure to replace "[dspace.url]" with the value of your 'dspace.url' setting in your
dspace.cfg file.
Sitemap: [dspace.url]/sitemap
Sitemap: [dspace.url]/htmlmap
#####
# Default Access Group
# (NOTE: blank lines are not allowable in a group record)
#####
User-agent: *
# Disable access to Discovery search and filters
Disallow: /discover
Disallow: /search-filter
# For JSPUI, replace "/search-filter" above with "/simple-search"
#
# Optionally uncomment the following line ONLY if sitemaps are working
# and you have verified that your site is being indexed correctly.
# Disallow: /browse
#
# If you have configured DSpace (Solr-based) Statistics to be publicly
# accessible, then you may not want this content to be indexed
# Disallow: /statistics
#
# You also may wish to disallow access to the following paths, in order
# to stop web spiders from accessing user-based content
# Disallow: /contact
# Disallow: /feedback
# Disallow: /forgot
# Disallow: /login
# Disallow: /register
```

WARNING: for your additional disallow statements to be recognized under the `User-agent: *` group, they *cannot be separated by white lines* from the declared `user-agent: *` block. A white line indicates the start of a new user agent block. Without a leading user-agent declaration on the first line, blocks are ignored. Comment lines are allowed and will not break the user-agent block.

This is OK:

```
User-agent: *
# Disable access to Discovery search and filters
Disallow: /discover
Disallow: /search-filter
Disallow: /statistics
Disallow: /contact
```

This is **not OK**, as the two lines at the bottom will be completely ignored.

```
User-agent: *
# Disable access to Discovery search and filters
Disallow: /discover
Disallow: /search-filter

Disallow: /statistics
Disallow: /contact
```

To identify if a specific user agent has access to a particular URL, you can use [this handy robots.txt tester](#).

For more information on the robots.txt format, please see the [Google Robots.txt documentation](#).

Ensure Item Metadata appears in the HTML HEAD

It's possible to greatly customize the look and feel of your DSpace, which makes it harder for search engines, and other tools and services such as [Zotero](#), [Connotea](#) and [SIMILE Piggy Bank](#), to correctly pick out item metadata fields. To address this, DSpace (both XMLUI and JSPUI) includes item metadata in the <head> element of each item's HTML display page.

```
<meta name="DC.type" content="Article" />
<meta name="DCTERMS.contributor" content="Tansley, Robert" />
```

If you have heavily customized your metadata fields away from Dublin Core, you can modify the crosswalk that generates these elements by modifying `[dspace]/config/crosswalks/xhtml-head-item.properties`.

Google Scholar Metadata in HTML HEAD

In addition to Dublin Core <meta> tags in the HTML HEAD, DSpace also includes Google Scholar specific metadata fields in each item's HTML display page.

```
<meta content="Tansley, Robert; Donohue, Timothy" name="citation_authors" />
<meta content="Ensuring your DSpace is indexed" name="citation_title" />
```

These meta tags are the "[Highwire Press tags](#)" which [Google Scholar recommends](#). If you have heavily customized your metadata fields, or wish to change the default "mappings" to these Highwire Press tags, they are configurable in `[dspace]/config/crosswalks/google-metadata.properties`

Much more information is available in the Configuration section on [Google Scholar Metadata Mappings](#).

Avoid redirecting file downloads to Item landing pages

Make sure that you never redirect "direct file downloads" (i.e. users who directly jump to downloading a file, often from a search engine) to the associated Item's splash/landing page. In the past, some DSpace sites have added these custom URL redirects in order to facilitate capturing statistics via Google Analytics or similar.

While these URL redirects may seem harmless, they may be flagged as [cloaking](#) or spam by Google, Google Scholar and other major search engines. This may hurt your site's search engine ranking or even cause your entire site to be flagged for removal from the search engine.

If you have these URL redirects in place, it is highly recommended to remove them immediately. If you created these redirects to facilitate capturing download statistics in Google Analytics, you should consider upgrading to DSpace 5.0 or above, which is able to automatically record bitstream downloads in Google Analytics (see [DS-2088](#)) without the need for any URL redirects.

In general, OAI-PMH is not useful to Search Engines

Feel free to support OAI-PMH, but be aware that in general it is not useful for search engines:

- No reliable way to determine OAI-PMH base URL for a DSpace site.
- No standard or predictable way to get to item display page or full text from an OAI-PMH record, making effective indexing and presenting meaningful results difficult.
- In most cases provides only access to simple Dublin Core, a subset of available metadata.
- **NOTE:** Back in 2008, Google officially announced they were [retiring support for OAI-PMH based Sitemaps](#). So, OAI-PMH will no longer help you get better indexing through Google. Instead, you should be using the DSpace 'generate-sitemaps' feature described above.

5.9.2 Google Scholar Metadata Mappings

Google Scholar, in crawling sites, prefers [Highwire Press tags](#). This schema contains names which are all prefixed by the string "citation_", and provide various metadata about the article/item being indexed.

As of DSpace 1.7, there is a mapping facility to connect metadata fields with these citation fields in HTML. In order to enable this functionality, the switch needs to be flipped in `dspace.cfg`:

```
google-metadata.enable = true
```

Once the feature is enabled, the mapping is configured by a separate configuration file located here:

```
[dspace]/config/crosswalks/google-metadata.properties
```



Please, note that the file location changed between DSpace 1.7 and 1.8. It's now in the "crosswalks" directory, so check that the google-metadata.config configuration property points to the right file:

```
google-metadata.config = ${dspace.dir}/config/crosswalks/  
google-metadata.properties
```

This file contains name/value pairs linking meta-tags with DSpace metadata fields. E.g...

```
google.citation_title = dc.title  
google.citation_publisher = dc.publisher  
google.citation_author = dc.author | dc.contributor.author | dc.creator
```

There is further documentation in this configuration file explaining proper syntax in specifying which metadata fields to use. If a value is omitted for a meta-tag field, the meta-tag is simply not included in the HTML output.

The values for each item are interpolated when the item is viewed, and the appropriate meta-tags are included in the HTML head tag, on both the Brief Item Display and the Full Item Display. This is implemented in the XMLUI and JSPUI.

Note: In DSpace 5, the field google.citation_authors was changed to google.citation_author.

5.10 Troubleshooting Information

You can quickly get some basic information about the DSpace version and the products supporting it by using the `[dspace]/bin/dspace version` command.

```
$ bin/dspace version
```

```
DSpace version: 4.0-SNAPSHOT
```

```
SCM revision: da53991b6b7e9f86c2a7f5292e3c2e9606f9f44c
```

```
SCM branch: UNKNOWN
```

```
OS: Linux(amd64) version 3.7.10-gentoo
```

```
Discovery enabled.
```

```
Lucene search enabled.
```

```
JRE: Oracle Corporation version 1.7.0_21
```

```
Ant version: Apache Ant(TM) version 1.8.4 compiled on June 25 2012
```

```
Maven version: 3.0.4
```


DSpace home: /home/dspace

\$

5.11 Validating CheckSums of Bitstreams

- 1 [Checksum Checker](#)
 - 1.1 [Checker Execution Mode](#)
 - 1.2 [Checker Results Pruning](#)
 - 1.3 [Checker Reporting](#)
 - 1.4 [Cron or Automatic Execution of Checksum Checker](#)
 - 1.5 [Automated Checksum Checkers' Results](#)

5.11.1 Checksum Checker

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

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

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

- the execution mode
- the logging output
- the policy for removing old checksum results from the database

The user should refer to Chapter 5. Configuration for specific configuration keys in the *dspace.cfg* file.

Checker Execution Mode

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

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

Available command line options

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

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

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

- **Specific Bitstream mode:** `[dspace]/bin/dspace checker -b` Checker will only look at the internal bitstream IDs. Example: `[dspace]/bin/dspace checker -b 112 113 4567` Checker will only check bitstream IDs 112, 113 and 4567.
- **Specific Handle mode:** `[dspace]/bin/dspace checker -a` Checker will only check bitstreams within the Community, Community or the item itself. Example: `[dspace]/bin/dspace checker -a 123456/999` Checker will only check this handle. If it is a Collection or Community, it will run through the entire Collection or Community.

- **Looping mode:** `[dspace]/bin/dspace checker -l` or `[dspace]/bin/dspace checker -L`
There are two modes. The lowercase 'l' (-l) specifies to check every bitstream in the repository once. This is recommended for smaller repositories who are able to loop through all their content in just a few hours maximum. An uppercase 'L' (-L) specifies to continuously loops through the repository. This is not recommended for most repository systems. **Cron Jobs.** For large repositories that cannot be completely checked in a couple of hours, we recommend the -d option in cron.
- **Pruning mode:** `[dspace]/bin/dspace checker -p` The Checksum Checker will store the result of every check in the `checksum_history` table. By default, successful checksum matches that are eight weeks old or older will be deleted when the -p option is used. (Unsuccessful ones will be retained indefinitely). Without this option, the retention settings are ignored and the database table may grow rather large!

Checker Results Pruning

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

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

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

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

Checker Reporting

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

```
[dspace]/bin/dspace checker -l -v
```

(This will loop through the repository once and report in detail about every bitstream checked.)

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

Cron or Automatic Execution of Checksum Checker

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

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

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

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

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

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

(This command should appear on a single line).

Automated Checksum Checkers' Results

Optionally, you may choose to receive automated emails listing the Checksum Checkers' results to the email address specified in the `mail.admin` configuration property. Schedule it to run **after** the Checksum Checker has completed its processing (otherwise the email may not contain all the results). As of DSpace 4.1, an email is only generated if the selected report contains at least one bitstream needing attention.

Command used:	<code>[dSPACE]/bin/dSPACE checker-emailer</code>
Java class:	<code>org.dSPACE.checker.DailyReportEmailer</code>
Arguments short and (long) forms):	Description
<code>-a</code> or <code>--All</code>	Send all the results (everything specified below)
<code>-d</code> or <code>--Deleted</code>	Send E-mail report for all bitstreams set as deleted for today.
<code>-m</code> or <code>--Missing</code>	Send E-mail report for all bitstreams not found in assetstore for today.

-c or --Changed	Send E-mail report for all bitstreams where checksum has been changed for today.
-u or --Unchanged	Send the Unchecked bitstream report.
-n or --Not Processed	Send E-mail report for all bitstreams set to longer be processed for today.
-h or --help	Help

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

Cron. Follow the same steps above as you would running checker in cron. Change the time but match the regularity. Remember to schedule this **after** Checksum Checker has run. For an example cron setup, see [Scheduled Tasks via Cron](#).

6 DSpace Development

This section contains information on how to modify, extend and customize the DSpace source code.

6.1 Advanced Customisation

It is anticipated that the customisation features described in the JSPUI and XMLUI customisation sections will be sufficient to satisfy the needs of the majority of users, however, some users may want to customise DSpace further, or just have a greater understanding of how to do so.

- 1 [Additions module](#)
- 2 [Maven WAR Overlays](#)
- 3 [DSpace Source Release](#)

6.1.1 Additions module

This module was added in DSpace 3.0 and should be used to store dspace-api changes, custom plugins, ... Classes placed in `[dspace-source]/dspace/modules/additions` will override those located in the dspace-api. This module will be used for all the webapps located in the `[dspace-source]/dspace/modules` directory and in the command line interface. **It is recommended to place all dspace-api changes in this module** so the changes made are contained in a single module, making it easier to get an overview of changes made.

6.1.2 Maven WAR Overlays

Much of the customisation described in the JSPUI and XMLUI customisation sections is based on [Maven WAR Overlays](#). In short, any classes or files placed in `[dspace-source]/dspace/modules/*` will be overlaid onto the selected WAR. This includes both new and amended files.

For more details on Maven WAR Overlays and how they relate to DSpace, see this presentation from Fall 2009: [Making DSpace XMLUI Your Own](#)

(Please note that this presentation was made for DSpace 1.5.x and 1.6.x, but much of it still applies to current versions of DSpace.)

6.1.3 DSpace Source Release

If you have downloaded the 'dspace-src-release' (or checked out the latest DSpace Code via [GitHub](#)), there are two main build options that are available to you:

1. **Full Build:** Running `mvn package` from the root `[dspace-source]` directory
 - This option will rebuild **all** DSpace modules from their Java Source code, then apply any [Maven WAR Overlays](#). In other words, all subdirectories of `[dspace-source]` are recompiled/rebuilt.
2. **Quick Build:** Running `mvn package` from the `[dspace-source]/dspace/` directory
 - This option performs a "quick build". It does **not** recompile/rebuild all DSpace modules. All it does is rebuild and re-apply any [Maven WAR Overlays](#) to the previously compiled source code. In other words, the **ONLY** code that will be recompiled/rebuilt is code that exists in `[dspace-source]/dspace/modules/*` (the Maven WAR Overlay directories)

Which build option you need to use will depend on your local development practices. If you have been careful to utilize [Maven WAR Overlays](#) for your local code/changes (putting everything under `[dspace-source]/dspace/modules/*`), then the **Quick Build** option may be the best way for you to recompile & reapply your local modifications. However, if you have made direct changes to code within a subdirectory of `[dspace-source]` (e.g. `/dspace-api`, `/dspace-xmlui`, `/dspace-jspui`, etc.) then you will need to utilize the **Full Build** option in order to ensure those modifications are included in the final WAR files.

6.1.4 DSpace Service Manager

- 1 [Introduction](#)
- 2 [Configuration](#)
 - 2.1 [Configuring Addons to Support Spring Services](#)
 - 2.2 [Configuration Priorities](#)
 - 2.2.1 [Configuring a new Addon](#)
 - 2.2.1.1 [Addon located as resource in jar](#)
 - 2.2.1.2 [Addon located in the \[dspace\]/config/spring directory](#)
 - 2.2.2 [The Core Spring Configuration](#)
 - 2.2.3 [Utilizing Autowiring to minimize configuration complexity.](#)
 - 2.3 [Accessing the Services Via Service Locator / Java Code](#)
- 3 [Architectural Overview](#)
 - 3.1 [Service Manager Startup in Webapplications and CLI](#)
- 4 [Tutorials](#)

Introduction

The DSpace Spring Service Manager supports overriding configuration at many levels.

Configuration

Configuring Addons to Support Spring Services

Configuring Addons to support Spring happens at two levels. Default Spring configuration is available in the DSpace JAR or WAR resources directory and allows the addon developer to inject configuration into the service

manager at load time. The second level is in the deployed [dspace]/config/spring directory where configurations can be provided on a addon module by addon module basis.

This latter method requires the addon to implement a SpringLoader to identify the location to look for Spring configuration and a place configuration files into that location. This can be seen inside the current [dspace-source]/config/modules/spring.cfg

Configuration Priorities

The ordering of the loading of Spring configuration is the following:

1. configPath = "spring/spring-dspace-applicationContext.xml" relative to the current classpath
2. addonResourcePath = "classpath*:spring/spring-dspace-addon-*-services.xml" relative to the current classpath
3. coreResourcePath = "classpath*:spring/spring-dspace-core-services.xml" relative to the current classpath
4. Finally, an array of SpringLoader API implementations that are checked to verify "config/spring/module" can actually be loaded by its existence on the classpath. The configuration of these SpringLoader API classes can be found in dspace.dir/config/modules/spring.cfg.

Configuring a new Addon

There are 2 ways to create a new Spring addon: a new Spring file can be located in the resources directory or in the configuration [dspace]/config/spring directory. A Spring file can also be located in both of these locations but the configuration directory gets preference and will override any configurations located in the resources directory

Addon located as resource in jar

In the resources directory of a certain module, a Spring file can be added if it matches the following pattern: "spring/spring-dspace-addon-*-services.xml". An example of this can be found in the dspace-discovery-solr block in the DSpace trunk. (`spring-dspace-addon-discovery-services.xml`)

Wherever this jar is loaded (JSPUI module, XMLUI module, DSpace command line, ...) the Spring files will be processed into services.

Addon located in the [dspace]/config/spring directory

This directory has the following subdirectories in which Spring files can be placed:

- api: when placed in this module the Spring files will always be processed into services (since all of the DSpace modules are dependent on the API).
- discovery: when placed in this module the Spring files will only be processed when the discovery library is present (in the case of discovery in the xmlui & in the command line interface).
- jspui: only processed for the JSPUI.
- xmlui: only processed for the XMLUI (example: the configurable workflow).

The reason why there is a separate directory is that if a service cannot be loaded, which would be the case for the configurable workflow (the JSPUI would not be able to retrieve the XMLUI interface classes), the kernel will crash and DSpace will not start.

Configuring an additional subdirectory for a custom module

So you need to indeed create a new directory in [dspace]/config/spring. Next you need to create a class that inherits from the "org.dspace.kernel.config.SpringLoader". This class only contains one method named getResourcePaths(). What we do now at the moment is implement this in the following manner:

```
@Override
public String[] getResourcePaths(ConfigurationService configurationService) {
    StringBuffer filePath = new StringBuffer();
    filePath.append(configurationService.getProperty("dspace.dir"));
    filePath.append(File.separator);
    filePath.append("config");
    filePath.append(File.separator);
    filePath.append("spring");
    filePath.append(File.separator);
    filePath.append("{module.name}"); //Fill in the module name in this string
    filePath.append(File.separator);
    try {
        //By adding the XML_SUFFIX here it doesn't matter if there should be some kind of
        spring.xml.old file in there it will only load in the active ones.
        return new String[]{new File(filePath.toString()).toURI().toURL().toString() + XML_SUFFIX
    };
    } catch (MalformedURLException e) {
        return new String[0];
    }
}
```

After the class has been created you will also need to add it to the "springloader.modules" property located in the [dspace]/config/modules/spring.cfg.

The Spring service manager will check this property to ensure that only the interface implementations which it can find the class for are loaded in.

By doing this way we give some flexibility to the developers so that they can always create their own Spring modules and then Spring will not crash when it can't find a certain class.

The Core Spring Configuration

Utilizing Autowiring to minimize configuration complexity.

Please see the following tutorials:

- [DSpace Spring Services Tutorial](#)
- [The TAO of DSpace Services](#)

Accessing the Services Via Service Locator / Java Code

Please see the following tutorials:

- [DSpace Spring Services Tutorial](#)

- [The TAO of DSpace Services](#)

Architectural Overview

Please see Architectural Overview here: [DSpace Services Framework](#)

Service Manager Startup in Webapplications and CLI

Please see the [DSpace Services Framework](#)

Tutorials

Several good Spring / DSpace Services Tutorials are already available:

- [DSpace Spring Services Tutorial](#)
- [The TAO of DSpace Services](#)

6.2 REST API

- [What is DSpace REST API](#)
 - [Installing the REST API](#)
 - [REST Endpoints](#)
 - [Index](#)
 - [Communities](#)
 - [Collections](#)
 - [Items](#)
 - [Bitstreams](#)
 - [Model - Object data types](#)
- [Introduction to Jersey for developers](#)
- [Configuration for DSpace REST](#)
- [Recording Proxy Access by Tools](#)
- [Deploying the DSpace REST API in your Servlet Container](#)
- [Additional Information](#)

6.2.1 What is DSpace REST API

The REST API module provides a programmatic interface to DSpace Communities, Collections, Items, and Bitstreams.

DSpace 4 introduced the initial REST API, which did not allow for authentication, and provided only READ-ONLY access to publicly accessible Communities, Collections, Items, and Bitstreams. DSpace 5 builds off of this and allows authentication to access restricted content, as well as allowing Create, Edit and Delete on the DSpace Objects. DSpace 5 REST API also provides improved pagination over resources and searching.

There has been a minor drift between the DSpace 4 REST API and the DSpace 5 REST API, so client applications will need to be targeted per version.

Installing the REST API

The REST API deploys as a standard webapp for your servlet container / tomcat. For example, depending on how you deploy webapps, one way would be to alter tomcat-home/conf/server.xml and add:

```
<Context path="/rest" docBase="/dspace/webapps/rest" />
```

In DSpace 4, the initial/official Jersey-based REST API was added to DSpace. The DSpace 4 REST API provides READ-ONLY access to DSpace Objects.

In DSpace 5, the REST API adds authentication, allows Creation, Update, and Delete to objects, can access restricted materials if authorized, and it requires SSL. For localhost development purposes, SSL can add additional getting-started difficulty, so security can be disabled. To disable DSpace REST's requirement to require security/ssl, alter [dspace]/webapps/rest/WEB-INF/web.xml or [dspace-source]/dspace-rest/src/main/webapp/WEB-INF/web.xml and comment out the security-constraint block, and restart your servlet container. Production usages of the REST API should use SSL, as authentication credentials should not go over the internet unencrypted.

REST Endpoints

The REST API is modeled after the DSpace Objects of Communities, Collections, Items, and Bitstreams. The API is not a straight database schema dump of these entities, but provides some wrapping that makes it easy to follow relationships in the API output.



HTTP Header: Accept

Note: You must set your request header's "Accept" property to either JSON (application/json) or XML (application/xml) depending on the format you prefer to work with.

Example usage from command line in XML format with pretty printing:

```
curl -s -H "Accept: application/xml" http://localhost:8080/rest/communities | xmllint
```

For this documentation, we will assume that the URL to the "REST" webapp will be <http://localhost:8080/rest/> for production systems, this address will be slightly different, such as: <http://demo.dspace.org/rest/>. The path to an endpoint, will go after the /rest/, such as /rest/communities, all-together this is: <http://localhost:8080/rest/communities>

Another thing to note is that there are Query Parameters that you can tack on to the end of an endpoint to do extra things. The most commonly used one in this API is "?expand". Instead of every API call defaulting to giving you every possible piece of information about it, it only gives a most commonly used set by default and gives the more "expensive" information when you deliberately request it. Each endpoint will provide a list of available expands in the output, but for getting started, you can start with ?expand=all, to make the endpoint provide all of its information (parent objects, metadata, child objects). You can include multiple expands, such as: ?expand=collections,subCommunities .

Index

Method	Endpoint	Description
GET	/	REST API static documentation page
POST	/login	<p>Login to the REST API using a DSpace EPerson / User. It returns an dspace-rest-token, that can be used for future authenticated requests.</p> <p>Example Request:</p> <pre>curl -H "Content-Type: application/json" --data '{"email":"admin@dspace.org", "password":"dspace"}' http://localhost:8080/rest/login</pre> <p>Example Response:</p> <pre>1febef81-5eb6-4e76-a0ea-a5be245563a5</pre> <p>Invalid email/password combinations will receive an HTTP 403 Forbidden.</p>
POST	/logout	<p>Logout from the REST API, by providing a header rest-dspace-token. After being posted this token will no longer work.</p> <p>Example Request:</p> <pre>curl -X POST -H "Content-Type: application/json" -H "rest-dspace-token: 1febef81-5eb6-4e76-a0ea-a5be245563a5" http://localhost:8080/rest/logout</pre> <p>Invalid token will result in HTTP 400 Invalid Request</p>
GET	/test	<p>Returns string "REST api is running", for testing that the API is up.</p> <p>Example Request:</p> <pre>curl http://localhost:8080/rest/test</pre> <p>Example Response:</p> <pre>REST api is running.</pre>
GET	/status	Receive information about the currently authenticated user token.

Method	Endpoint	Description
		<p>Example Request:</p> <pre>curl -X GET -H "Content-Type: application/json" -H "Accept: application/json" -H "rest-dspace-token: f2f478e2-90f2-4e77-a757-4e838ae94154" http://localhost:8080/rest/status</pre> <p>Example Response:</p> <pre>{"okay":true,"authenticated":true,"email":"admin@dspace.org","fullname":"DSpace Administrator","token":"f2f478e2-90f2-4e77-a757-4e838ae94154"}</pre>

Communities

Communities in DSpace are used for organization and hierarchy, and are containers that hold sub-Communities and Collections. (ex: Department of Engineering)

- GET /communities - Returns array of all communities in DSpace.
- GET /communities/top-communities - Returns array of all top communities in DSpace.
- GET /communities/{communityId} - Returns community.
- GET /communities/{communityId}/collections - Returns array of collections of community.
- GET /communities/{communityId}/communities - Returns array of subcommunities of community.
- POST /communities - Create new community at top level. You must post community.
- POST /communities/{communityId}/collections - Create new collections in community. You must post Collection.
- POST /communities/{communityId}/communities - Create new subcommunity in community. You must post Community.
- PUT /communities/{communityId} - Update community. You must put Community
- DELETE /communities/{communityId} - Delete community.
- DELETE /communities/{communityId}/collections/{collectionId} - Delete collection in community.
- DELETE /communities/{communityId}/communities/{communityId2} - Delete subcommunity in community.

Collections

Collections in DSpace are containers of Items. (ex: Engineering Faculty Publications)

- GET /collections - Return all collections of DSpace in array.
- GET /collections/{collectionId} - Return collection with id.
- GET /collections/{collectionId}/items - Return all items of collection.
- POST /collections/{collectionId}/items - Create posted item in collection. You must post an Item
- POST /collections/find-collection - Find collection by passed name.
- PUT /collections/{collectionId} - Update collection. You must put Collection.
- DELETE /collections/{collectionId} - Delete collection from DSpace.
- DELETE /collections/{collectionId}/items/{itemId} - Delete item in collection.

Items

Items in DSpace represent a "work" and combine metadata and files, known as Bitstreams.

- GET /items - Return list of items.
- GET /items/{item id} - Return item.
- GET /items/{item id}/metadata - Return item metadata.
- GET /items/{item id}/bitstreams - Return item bitstreams.
- POST /items/find-by-metadata-field - Find items by metadata entry. You must post a MetadataEntry
- POST /items/{item id}/metadata - Add metadata to item. You must post an array of MetadataEntry
- POST /items/{item id}/bitstreams - Add bitstream to item. You must post a Bitstream
- PUT /items/{item id}/metadata - Update metadata in item. You must put a MetadataEntry
- DELETE /items/{item id} - Delete item.
- DELETE /items/{item id}/metadata - Clear item metadata.
- DELETE /items/{item id}/bitstreams/{bitstream id} - Delete item bitstream.

Bitstreams

Bitstreams are files. They have a filename, size (in bytes), and a file format. Typically in DSpace, the Bitstream will be the "full text" article, or some other media. Some files are the actual file that was uploaded (tagged with `bundleName:ORIGINAL`), others are DSpace-generated files that are derivatives or renditions, such as text-extraction, or thumbnails. You can download files/bitstreams. DSpace doesn't really limit the type of files that it takes in, so this could be PDF, JPG, audio, video, zip, or other. Also, the logo for a Collection or a Community, is also a Bitstream.

- GET /bitstreams - Return all bitstreams in DSpace.
- GET /bitstreams/{bitstream id} - Return bitstream.
- GET /bitstreams/{bitstream id}/policy - Return bitstream policies.
- GET /bitstreams/{bitstream id}/retrieve - Return data of bitstream.
- POST /bitstreams/{bitstream id}/policy - Add policy to item. You must post a ResourcePolicy
- PUT /bitstreams/{bitstream id}/data - Update data/file of bitstream. You must put the data
- PUT /bitstreams/{bitstream id} - Update metadata of bitstream. You must put a Bitstream, does not alter the file/data
- DELETE /bitstreams/{bitstream id} - Delete bitstream from DSpace.
- DELETE /bitstreams/{bitstream id}/policy/{policy_id} - Delete bitstream policy.

You can access the parent object of a Bitstream (normally an Item, but possibly a Collection or Community when it is its logo) through: `/bitstreams/:bitstreamID?expand=parent`

As the documentation may state "You must post a ResourcePolicy" or some other object type, this means that there is a structure of data types, that your XML or JSON must be of type, when it is posted in the body.

Model - Object data types

Here are all of the data types, not all fields are necessary or supported when posting/putting content, but the output contains this information:

Community Object

```
{ "id": 456, "name": "Reports Community", "handle": "10766/10213", "type": "community", "link": "/RESTapi/communities/456", "expand": ["parentCommunity", "collections", "subCommunities", "logo", "all"], "logo": null, "parentCommunity": null, "copyrightText": "", "introductoryText": "", "shortDescription": "Collection contains materials pertaining to the Able Family", "sidebarText": "", "countItems": 3, "subcommunities": [], "collections": [] }
```

Collection Object

```
{ "id": 730, "name": "Annual Reports Collection", "handle": "10766/10214", "type": "collection", "link": "/RESTapi/collections/730", "expand": ["parentCommunityList", "parentCommunity", "items", "license", "logo", "all"], "logo": null, "parentCommunity": null, "parentCommunityList": [], "items": [], "license": null, "copyrightText": "", "introductoryText": "", "shortDescription": "", "sidebarText": "", "numberItems": 3 }
```

Item Object

```
{ "id": 14301, "name": "2015 Annual Report", "handle": "123456789/13470", "type": "item", "link": "/RESTapi/items/14301", "expand": ["metadata", "parentCollection", "parentCollectionList", "parentCommunityList", "bitstreams", "all"], "lastModified": "2015-01-12 15:44:12.978", "parentCollection": null, "parentCollectionList": null, "parentCommunityList": null, "bitstreams": null, "archived": true, "withdrawn": false }
```

Bitstream Object

```
{ "id": 47166, "name": "appearance and physiology 100 percent copied from wikipedia.pdf", "handle": null, "type": "bitstream", "link": "/RESTapi/bitstreams/47166", "expand": ["parent", "policies", "all"], "bundleName": "ORIGINAL", "description": "", "format": "Adobe PDF", "mimeType": "application/pdf", "sizeBytes": 129112, "parentObject": null, "retrieveLink": "/bitstreams/47166/retrieve", "checksum": { "value": "62778292a3a6dccbe2662a2bfca3b86e", "checksumAlgorithm": "MD5"}, "sequenceId": 1, "policies": null }
```

ResourcePolicy Object

```
[ { "id": 317127, "action": "READ", "epersonId": -1, "groupId": 0, "resourceId": 47166, "resour
```

MetadataEntry Object

```
{ "key": "dc.description.abstract", "value": "This is the description abstract", "lang": null }
```

User Object

```
{ "email": "test@dspace.org", "password": "pass" }
```

Status Object

```
{ "okay": true, "authenticated": true, "email": "test@dspace.org", "fullName": "DSpace Test User", "token": "6d45daaa-7b02-4ae7-86de-a960838fae5c" }
```

6.2.2 Introduction to Jersey for developers

The REST API for DSpace is implemented using Jersey, the reference implementation of the Java standard for building RESTful Web Services (JAX-RS 1). That means this API should be easier to expand and maintain than other API approaches, as this approach has been widely adopted in the industry. If this client documentation does not fully answer about how an endpoint works, it is helpful to look directly at the [Java REST API code](#), to see how it is implemented. The code typically has required parameters, optional parameters, and indicates the type of data that will be responded.

There was no central ProviderRegistry that you have to declare your path. Instead, the code is driven by annotations, here is a list of annotations used in the code for CommunitiesResource.java:

- `@Path("/communities")`, which then allows it to be routed to <http://localhost:8080/communities>, this is then the base path for all the requests within this class.
- `@GET`, which indicates that this method responds to GET http requests
- `@POST`, which indicates that this method responds to POST http requests
- `@PUT`, which indicates that this method responds to PUT http requests
- `@DELETE`, which indicates that this method responds to DELETE http requests
- `@Path("/{community_id}")`, the path is appended to the class level `@Path` above, this one uses a variable `{community_id}`. The total endpoint would be <http://localhost:8080/rest/communities/123>, where 123 is the ID.
- `@Consumes({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML })`, this indicates that this request expects input of either JSON or XML. Another endpoint accepts HTML input.
- `@PathParam("community_id") Integer communityId`, this maps the path placeholder variable `{community_id}` to Java int `communityID`
- `@QueryParam("userIP") String user_ip`, this maps a query param like `?userIP=8.8.4.4` to Java String `user_id` variable, and `user_id == "8.8.4.4"`

6.2.3 Configuration for DSpace REST

Property	stats
Example Value	true
Informational Note	Boolean value indicates whether statistics should be recorded for access via the REST API; Defaults to 'false'.

6.2.4 Recording Proxy Access by Tools

For the purpose of more accurate statistics, a web-based tool may specify who is using it, by adding parameters to the request:

```
http://localhost:8080/rest/items/:ID?userIP=ip&userAgent=userAgent&xforwarderfor=xforwarderfor
```

If no parameters are given, the details of the HTTP request's sender are used in statistics. This enables tools to record the details of their user rather than themselves.

6.2.5 Deploying the DSpace REST API in your Servlet Container

The `dspace-rest` module is automatically configured to compile and build with DSpace 4.0, so a `mvn+ant` process will create the webapp. To make it work in your environment, you would just need to add a context entry for it in your servlet container. For example, in tomcat, one might alter `$CATALINA_HOME/conf/server.xml` and add:

```
<Context path="/rest" docBase="/dspace/webapps/rest"/>
```

6.2.6 Additional Information

Additional information can be found in the [README for dspace-rest](#), and in the GitHub [Pull Request for DSpace REST \(Jersey\)](#).

Usage examples can be found at: <https://github.com/BrunoNZ/dspace-rest-requests>

6.3 Curation System

As of release 1.7, DSpace supports running curation tasks, which are described in this section. DSpace includes several useful tasks out-of-the-box, but the system also is designed to allow new tasks to be added between releases, both general purpose tasks that come from the community, and locally written and deployed tasks.

- 1 [Changes in 1.8](#)
- 2 [Tasks](#)
- 3 [Activation](#)
- 4 [Writing your own tasks](#)
- 5 [Task Invocation](#)
 - 5.1 [On the command line](#)

- 5.2 [In the admin UI](#)
- 5.3 [In workflow](#)
- 5.4 [In arbitrary user code](#)
- 6 [Asynchronous \(Deferred\) Operation](#)
- 7 [Task Output and Reporting](#)
 - 7.1 [Status Code](#)
 - 7.2 [Result String](#)
 - 7.3 [Reporting Stream](#)
- 8 [Task Properties](#)
- 9 [Task Annotations](#)
- 10 [Scripted Tasks](#)
 - 10.1 [Interface](#)
 - 10.1.1 [performDso\(\) vs. performId\(\)](#)
- 11 [Bundled Tasks](#)
 - 11.1 [MetadataWebService Task](#)
 - 11.1.1 [ISSN to Publisher Name](#)
 - 11.1.2 [HTTP Headers](#)
 - 11.1.3 [Transformations](#)
 - 11.1.4 [Result String Programatic Use](#)
 - 11.1.5 [Limits and Use](#)
 - 11.2 [NoOp Curation Task](#)
 - 11.3 [Bitstream Format Profiler](#)
 - 11.4 [Required Metadata](#)
 - 11.5 [Virus Scan](#)
 - 11.5.1 [Setup the service from the ClamAV documentation.](#)
 - 11.5.2 [DSpace Configuration](#)
 - 11.5.3 [Task Operation from the Administrative user interface](#)
 - 11.5.4 [Task Operation from the Item Submission user interface](#)
 - 11.5.5 [Task Operation from the curation command line client](#)
 - 11.5.5.1 [Table 1 – Virus Scan Results Table](#)
 - 11.6 [Link Checkers](#)
 - 11.6.1 [Basic Link Checker](#)
 - 11.6.2 [Metadata Value Link Checker](#)
 - 11.7 [Microsoft Translator](#)
 - 11.7.1 [Configure Microsoft Translator](#)

6.3.1 Changes in 1.8

- *New package:* The default curation task package is now **org.dspace.ctask**. The tasks supplied with DSpace releases are now under **org.dspace.ctask.general**
- *New tasks in DSpace release:* Some additional curation tasks have been supplied with DSpace 1.8, including a link checker and a translator

- *UI task groups*: Ability to assign tasks to groups whose members display together in the Administrative UI
- *Task properties*: Support for a site-portable system for configuration and profiling of tasks using configuration files
- *New framework services*: Support for context management during curation operations
- *Scripted tasks*: New (experimental) support for authoring and executing tasks in languages other than Java

6.3.2 Tasks

The goal of the curation system ("CS") is to provide a simple, extensible way to manage routine content operations on a repository. These operations are known to CS as "tasks", and they can operate on any DSpaceObject (i.e. subclasses of DSpaceObject) - which means the entire Site, Communities, Collections, and Items - viz. core data model objects. Tasks may elect to work on only one type of DSpace object - typically an Item - and in this case they may simply ignore other data types (tasks have the ability to "skip" objects for any reason). The DSpace core distribution will provide a number of useful tasks, but the system is designed to encourage local extension - tasks can be written for any purpose, and placed in any java package. This gives DSpace sites the ability to customize the behavior of their repository without having to alter - and therefore manage synchronization with - the DSpace source code. What sorts of activities are appropriate for tasks?

Some examples:

- apply a virus scan to item bitstreams (this will be our example below)
- profile a collection based on format types - good for identifying format migrations
- ensure a given set of metadata fields are present in every item, or even that they have particular values
- call a network service to enhance/replace/normalize an item's metadata or content
- ensure all item bitstreams are readable and their checksums agree with the ingest values

Since tasks have access to, and can modify, DSpace content, performing tasks is considered an administrative function to be available only to knowledgeable collection editors, repository administrators, sysadmins, etc. No tasks are exposed in the public interfaces.

6.3.3 Activation

For CS to run a task, the code for the task must of course be included with other deployed code (to [dspace]/lib, WAR, etc) but it must also be declared and given a name. This is done via a configuration property in [dspace]/config/modules/curate.cfg as follows:

```
plugin.named.org.dspace.curate.CurationTask = \
org.dspace.ctask.general.NoOpCurationTask = noop, \
org.dspace.ctask.general.ProfileFormats = profileformats, \
org.dspace.ctask.general.RequiredMetadata = requiredmetadata, \
org.dspace.ctask.general.ClamScan = vscan, \
org.dspace.ctask.general.MicrosoftTranslator = translate, \
```

```
org.dspace.ctask.general.MetadataValueLinkChecker = checklinks
```

For each activated task, a key-value pair is added. The key is the fully qualified class name and the value is the *taskname* used elsewhere to configure the use of the task, as will be seen below. Note that the *curate.cfg* configuration file, while in the config directory, is located under "modules". The intent is that tasks, as well as any configuration they require, will be optional "add-ons" to the basic system configuration. Adding or removing tasks has no impact on *dspace.cfg*.

For many tasks, this activation configuration is all that will be required to use it. But for others, the task needs specific configuration itself. A concrete example is described below, but note that these task-specific configuration property files also reside in `[dspace]/config/modules`

6.3.4 Writing your own tasks

A task is just a java class that can contain arbitrary code, but it must have 2 properties:

First, it must provide a no argument constructor, so it can be loaded by the PluginManager. Thus, all tasks are 'named' plugins, with the taskname being the plugin name.

Second, it must implement the interface "org.dspace.curate.CurationTask"

The CurationTask interface is almost a "tagging" interface, and only requires a few very high-level methods be implemented. The most significant is:

```
int perform(DSpaceObject dso);
```

The return value should be a code describing one of 4 conditions:

- 0 : SUCCESS the task completed successfully
- 1 : FAIL the task failed (it is up to the task to decide what 'counts' as failure - an example might be that the virus scan finds an infected file)
- 2 : SKIPPED the task could not be performed on the object, perhaps because it was not applicable
- -1 : ERROR the task could not be completed due to an error

If a task extends the AbstractCurationTask class, that is the only method it needs to define.

6.3.5 Task Invocation

Tasks are invoked using CS framework classes that manage a few details (to be described below), and this invocation can occur wherever needed, but CS offers great versatility "out of the box":

On the command line

A simple tool "CurationCli" provides access to CS via the command line. This tool bears the name "curate" in the DSpace launcher. For example, to perform a virus check on collection "4":

```
[dspace]/bin/dspace curate -t vsan -i 123456789/4
```

The complete list of arguments:

```
-t taskname: name of task to perform
-T filename: name of file containing list of tasknames
-e epersonID: (email address) will be superuser if unspecified
-i identifier: Id of object to curate. May be (1) a handle (2) a workflow Id or (3) 'all' to
operate on the whole repository
-q queue: name of queue to process - -i and -q are mutually exclusive
-l limit: maximum number of objects in Context cache. If absent, unlimited objects may be added.
-s scope: declare a scope for database transactions. Scope must be: (1) 'open' (default value) (2)
'curation' or (3) 'object'
-v emit verbose output
-r - emit reporting to standard out
```

As with other command-line tools, these invocations could be placed in a cron table and run on a fixed schedule, or run on demand by an administrator.

In the admin UI

In the UI, there are several ways to execute configured Curation Tasks:

1. **From the "Curate" tab/button that appears on each "Edit Community/Collection/Item" page:** this tab allows an Administrator, Community Administrator or Collection Administrator to run a Curation Task on that particular Community, Collection or Item. When running a task on a Community or Collection, that task will also execute on all its child objects, unless the Task itself states otherwise (e.g. running a task on a Collection will also run it across all Items within that Collection).
 - NOTE: Community Administrators and Collection Administrators can only run Curation Tasks on the Community or Collection which they administer, along with any child objects of that Community or Collection. For example, a Collection Administrator can run a task on that specific Collection, or on any of the Items within that Collection.
2. **From the Administrator's "Curation Tasks" page:** This option is only available to DSpace Administrators, and appears in the Administrative side-menu. This page allows an Administrator to run a Curation Task across a single object, or all objects within the entire DSpace site.
 - In order to run a task from this interface, you must enter in the handle for the DSpace object. To run a task site-wide, you can use the handle: [your-handle-prefix]/0

Each of the above pages exposes a drop-down list of configured tasks, with a button to 'perform' the task, or queue it for later operation (see section below). Not all activated tasks need appear in the Curate tab - you filter them by means of a configuration property. This property also permits you to assign to the task a more user-friendly name than the PluginManager *taskname*. The property resides in `[dspace]/config/modules/curate.cfg`:

```
ui.tasknames = \  
  profileformats = Profile Bitstream Formats, \  
  requiredmetadata = Check for Required Metadata
```

When a task is selected from the drop-down list and performed, the tab displays both a phrase interpreting the "status code" of the task execution, and the "result" message if any has been defined. When the task has been queued, an acknowledgement appears instead. You may configure the words used for status codes in `curate.cfg` (for clarity, language localization, etc):

```
ui.statusmessages = \  
  -3 = Unknown Task, \  
  -2 = No Status Set, \  
  -1 = Error, \  
  0 = Success, \  
  1 = Fail, \  
  2 = Skip, \  
  other = Invalid Status
```

As the number of tasks configured for a system grows, a simple drop-down list of **all** tasks may become too cluttered or large. DSpace 1.8+ provides a way to address this issue, known as *task groups*. A task group is a simple collection of tasks that the Admin UI will display in a separate drop-down list. You may define as many or as few groups as you please. If no groups are defined, then all tasks that are listed in the `ui.tasknames` property will appear in a single drop-down list. If at least *one* group is defined, then the admin UI will display **two** drop-down lists. The first is the list of task groups, and the second is the list of task names associated with the selected group. A few key points to keep in mind when setting up task groups:

- a task can appear in more than one group if desired
- tasks that belong to no group are *invisible* to the admin UI (but of course available in other contexts of use)

The configuration of groups follows the same simple pattern as tasks, using properties in `[dspace]/config/modules/curate.cfg`. The group is assigned a simple logical name, but also a localizable name that appears in the UI. For example:

```
# ui.taskgroups contains the list of defined groups, together with a pretty name for UI display  
ui.taskgroups = \  
  replication = Backup and Restoration Tasks, \  
  integrity = Metadata Integrity Tasks, \  
  ...
```

```

.....
# each group membership list is a separate property, whose value is comma-separated list of logical
task names
ui.taskgroup.integrity = profileformats, requiredmetadata
.....

```

In workflow

CS provides the ability to attach any number of tasks to standard DSpace workflows. Using a configuration file [dspace]/config/workflow-curation.xml, you can declaratively (without coding) wire tasks to any step in a workflow. An example:

```

<taskset-map>
  <mapping collection-handle="default" taskset="cautious" />
</taskset-map>
<tasksets>
  <taskset name="cautious">
    <flowstep name="step1">
      <task name="vscan">
        <workflow>reject</workflow>
        <notify on="fail">${flowgroup}</notify>
        <notify on="fail">${colladmin}</notify>
        <notify on="error">${siteadmin}</notify>
      </task>
    </flowstep>
  </taskset>
</tasksets>

```

This markup would cause a virus scan to occur during step one of workflow for any collection, and automatically reject any submissions with infected files. It would further notify (via email) both the reviewers (step 1 group), and the collection administrators, if either of these are defined. If it could not perform the scan, the site administrator would be notified.

The notifications use the same procedures that other workflow notifications do - namely email. There is a new email template defined for curation task use: [dspace]/config/emails/flowtask_notify. This may be language-localized or otherwise modified like any other email template.

Tasks wired in this way are normally performed as soon as the workflow step is entered, and the outcome action (defined by the 'workflow' element) immediately follows. It is also possible to delay the performance of the task - which will ensure a responsive system - by queuing the task instead of directly performing it:

```

...
  <taskset name="cautious">
    <flowstep name="step1" queue="workflow">
  ...

```

This attribute (which must always follow the "name" attribute in the flowstep element), will cause all tasks associated with the step to be placed on the queue named "workflow" (or any queue you wish to use, of course), and further has the effect of **suspending** the workflow. When the queue is emptied (meaning all tasks in it performed), then the workflow is restarted. Each workflow step may be separately configured,

Like configurable submission, you can assign these task rules per collection, as well as having a default for any collection.

In arbitrary user code

If these pre-defined ways are not sufficient, you can of course manage curation directly in your code. You would use the CS helper classes. For example:

```
Collection coll = (Collection)HandleManager.resolveToObject(context, "123456789/4");
Curator curator = new Curator();
curator.addTask("vscan").curate(coll);
System.out.println("Result: " + curator.getResult("vscan"));
```

would do approximately what the command line invocation did. the method "curate" just performs all the tasks configured (you can add multiple tasks to a curator).

6.3.6 Asynchronous (Deferred) Operation

Because some tasks may consume a fair amount of time, it may not be desirable to run them in an interactive context. CS provides a simple API and means to defer task execution, by a queuing system. Thus, using the previous example:

```
Curator curator = new Curator();
curator.addTask("vscan").queue(context, "monthly", "123456789/4");
```

would place a request on a named queue "monthly" to virus scan the collection. To read (and process) the queue, we could for example:

```
[dspace]/bin/dspace curate -q monthly
```

use the command-line tool, but we could also read the queue programmatically. Any number of queues can be defined and used as needed.

In the administrative UI curation "widget", there is the ability to both perform a task, but also place it on a queue for later processing.

6.3.7 Task Output and Reporting

Few assumptions are made by CS about what the 'outcome' of a task may be (if any) - it could e.g. produce a report to a temporary file, it could modify DSpace content silently, etc. But the CS runtime does provide a few pieces of information whenever a task is performed:

Status Code

This was mentioned above. This is returned to CS whenever a task is called. The complete list of values:

```
-3 NOTASK - CS could not find the requested task
-2 UNSET  - task did not return a status code because it has not yet run
-1 ERROR  - task could not be performed
 0 SUCCESS - task performed successfully
 1 FAIL    - task performed, but failed
 2 SKIP    - task not performed due to object not being eligible
```

In the administrative UI, this code is translated into the word or phrase configured by the *ui.statusmessages* property (discussed above) for display.

Result String

The task may define a string indicating details of the outcome. This result is displayed, in the "curation widget" described above:

```
"Virus 12312 detected on Bitstream 4 of 1234567789/3"
```

CS does not interpret or assign result strings, the task does it. A task may not assign a result, but the "best practice" for tasks is to assign one whenever possible.

Reporting Stream

For very fine-grained information, a task may write to a *reporting* stream. This stream is sent to standard out, so is only available when running a task from the command line. Unlike the result string, there is no limit to the amount of data that may be pushed to this stream.

The status code, and the result string are accessed (or set) by methods on the Curation object:

```
Curator curator = new Curator();
curator.addTask("vscan").curate(coll);
int status = curator.getStatus("vscan");
String result = curator.getResult("vscan");
```

6.3.8 Task Properties

DSpace 1.8 introduces a new "idiom" for tasks that require configuration data. It is available to any task whose implementation extends `AbstractCurationTask`, but is completely optional. There are a number of problems that task properties are designed to solve, but to make the discussion concrete we will start with a particular one: the problem of hard-coded configuration file names. A task that relies on configuration data will typically encode a fixed reference to a configuration file name. For example, the virus scan task reads a file called "clamav.cfg", which lives in `[dspace]/config/modules`. And thus in the implementation one would find:

```
host = ConfigurationManager.getProperty("clamav", "service.host");
```

and similar. But tasks are supposed to be written by anyone in the community and shared around (without prior coordination), so if another task uses the same configuration file name, there is a name **collision** here that can't be easily fixed, since the reference is hard-coded in each task. In this case, if we wanted to use both at a given site, we would have to alter the source of one of them - which introduces needless code localization and maintenance.

Task properties gives us a simple solution. Here is how it works: suppose that both colliding tasks instead use this method provided by `AbstractCurationTask` in their task implementation code (e.g. in virus scanner):

```
host = taskProperty("service.host");
```

Note that there is no name of the configuration file even mentioned, just the property name whose value we want. At runtime, the curation system **resolves** this call to a configuration file, and it uses the *name the task has been configured as* as the name of the config file. So, for example, if both were installed (in `curate.cfg`) as:

```
org.dspace.ctask.general.ClamAv = vscan,  
org.community.ctask.ConflictTask = virusscan,  
....
```

then "taskProperty()" will resolve to `[dspace]/config/modules/vscan.cfg` when called from `ClamAv` task, but `[dspace]/config/modules/virusscan.cfg` when called from `ConflictTask`'s code. Note that the "vscan" etc are locally assigned names, so we can always prevent the "collisions" mentioned, and we make the tasks much more portable, since we remove the "hard-coding" of config names.

The entire "API" for task properties is:

```
public String taskProperty(String name);  
public int taskIntProperty(String name, int defaultValue);  
public long taskLongProperty(String name, long defaultValue);  
public boolean taskBooleanProperty(String name, boolean default);
```

Another use of task properties is to support multiple task profiles. Suppose we have a task that we want to operate in one of two modes. A good example would be a mediafilter task that produces a thumbnail. We can either create one if it doesn't exist, or run with "-force" which will create one regardless. Suppose this behavior was controlled by a property in a config file. If we configured the task as "thumbnail", then we would have in [dspace]/config/modules/thumbnail.cfg:

```
...other properties...
thumbnail.maxheight = 80
thumbnail.maxwidth = 80
forceupdate=false
```

Then, following the pattern above, the thumbnail generating task code would look like:

```
if (taskBooleanProperty("forceupdate")) {
    // do something
}
```

But an obvious use-case would be to want to run force mode **and** non-force mode from the admin UI on different occasions. To do this, one would have to stop Tomcat, change the property value in the config file, and restart, etc However, we can use task properties to elegantly rescue us here. All we need to do is go into the config/modules directory, and create a new file called: thumbnail.force.cfg. In this file, we put only **one** property:

```
forceupdate=true
```

Then we add a new task (really just a new name, no new code) in curate.cfg:

```
org.dspace.ctask.general.ThumbnailTask = thumbnail,
org.dspace.ctask.general.ThumbnailTask = thumbnail.force
```

Consider what happens: when we perform the task "thumbnail" (using taskProperties), it reads the config file thumbnail.cfg and operates in "non-force" profile (since the value is false), but when we run the task "thumbnail.force" the curation system **first** reads thumbnail.cfg, **then** reads thumbnail.force.cfg which **overrides** the value of the "forceupdate" property. Notice that we did all this via local configuration - we have not had to touch the source code at all to obtain as many "profiles" as we would like.

6.3.9 Task Annotations

CS looks for, and will use, certain java annotations in the task Class definition that can help it invoke tasks more intelligently. An example may explain best. Since tasks operate on DSOs that can either be simple (Items) or containers (Collections, and Communities), there is a fundamental problem or ambiguity in how a task is invoked

: if the DSO is a collection, should the CS invoke the task on each member of the collection, or does the task "know" how to do that itself? The decision is made by looking for the `@Distributive` annotation: if present, CS assumes that the task will manage the details, otherwise CS will walk the collection, and invoke the task on each member. The java class would be defined:

```
@Distributive
public class MyTask implements CurationTask
```

A related issue concerns how non-distributive tasks report their status and results: the status will normally reflect only the last invocation of the task in the container, so important outcomes could be lost. If a task declares itself `@Suspendable`, however, the CS will cease processing when it encounters a FAIL status. When used in the UI, for example, this would mean that if our virus scan is running over a collection, it would stop and return status (and result) to the scene on the first infected item it encounters. You can even tune `@Suspendable` tasks more precisely by annotating what invocations you want to suspend on. For example:

```
@Suspendable(invoked=Curator.Invoked.INTERACTIVE)
public class MyTask implements CurationTask
```

would mean that the task would suspend if invoked in the UI, but would run to completion if run on the command-line.

Only a few annotation types have been defined so far, but as the number of tasks grow, we can look for common behavior that can be signaled by annotation. For example, there is a `@Mutative` type: that tells CS that the task may alter (mutate) the object it is working on.

6.3.10 Scripted Tasks



The procedure to set up curation tasks in Jython is described on a separate page: [Curation tasks in Jython](#)

DSpace 1.8 includes limited (and somewhat experimental) support for deploying and running tasks written in languages other than Java. Since version 6, Java has provided a standard way (API) to invoke so-called scripting or dynamic language code that runs on the java virtual machine (JVM). Scripted tasks are those written in a language accessible from this API. The exact number of supported languages will vary over time, and the degree of maturity of each language, or suitability of the language for curation tasks will also vary significantly. However, preliminary work indicates that Ruby (using the JRuby runtime) and Groovy may prove viable task languages.

Support for scripted tasks does **not** include any DSpace pre-installation of the scripting language itself - this must be done according to the instructions provided by the language maintainers, and typically only requires a

few additional jars on the DSpace classpath. Once one or more languages have been installed into the DSpace deployment, task support is fairly straightforward. One new property must be defined in `[dspace]/config/modules/curate.cfg`:

```
script.dir = ${dspace.dir}/scripts
```

This merely defines the directory location (usually relative to the deployment base) where task script files should be kept. This directory will contain a "catalog" of scripted tasks named `task.catalog` that contains information needed to run scripted tasks. Each task has a 'descriptor' property with value syntax:

```
<engine> | <relFilePath> | <implClassCtor>
```

An example property for a link checking task written in Ruby might be:

```
linkchecker = ruby|rubytask.rb|LinkChecker.new
```

This descriptor means that a "ruby" script engine will be created, a script file named "rubytask.rb" in the directory `<script.dir>` will be loaded and the resolver will expect an evaluation of "LinkChecker.new" will provide a correct implementation object. Note that the task must be configured in all other ways just like java tasks (in `ui.tasknames`, `ui.taskgroups`, etc).

Script files may embed their descriptors to facilitate deployment. To accomplish this, a script must include the descriptor string with syntax:

`$td=<descriptor>` somewhere on a comment line. For example:

```
# My descriptor $td=ruby|rubytask.rb|LinkChecker.new
```

For reasons of portability, the `<relFilePath>` component may be omitted in this context. Thus, "`$td=ruby| | LinkChecker.new`" will be expanded to a descriptor with the name of the embedding file.

Interface

Scripted tasks must implement a slightly different interface than the [CurationTask](#) interface used for Java tasks. The appropriate interface for scripting tasks is [ScriptedTask](#) and has the following methods:

```
public void init(Curator curator, String taskId) throws IOException;
public int performDso(DSpaceObject dso) throws IOException;
public int performId(Context ctx, String id) throws IOException;
```

The difference is that `ScriptedTask` has separate perform methods for DSO and identifier. The reason for that is that some scripting languages (e.g. Ruby) don't support method overloading.

performDso() vs. performId()

You may have noticed that the `ScriptedTask` interface has both `performDso()` and `performId()` methods, but only `performDso` is ever called when curator is launched from command line.

There are a class of use-cases in which we want to construct or create new DSOs (`DspaceObject`) given an identifier in a task. In these cases, there may be no live DSO to pass to the task.

You actually **can** get curation system to call `performId()` if you queue a task then process the queue - when reading the queue all CLI has is the handle to pass to the task.

6.3.11 Bundled Tasks

DSpace bundles a small number of tasks of general applicability. Those that do not require configuration (or have usable default values) are activated to demonstrate the use of the curation system. They may be removed (deactivated by means of configuration) if desired without affecting system integrity. Those that require configuration may be enabled (activated) by means editing DSpace configuration files. Each task - current as of DSpace 4.0 - is briefly described below.

MetadataWebService Task

DSpace item metadata can contain any number of identifiers or other field values that participate in networked information systems. For example, an item may include a DOI which is a controlled identifier in the DOI registry. Many web services exist to leverage these values, by using them as 'keys' to retrieve other useful data. In the DOI case for example, CrossRef provides many services that given a DOI will return author lists, citations, etc. The `MetadataWebService` task enables the use of such services, and allows you to obtain and (optionally) add to DSpace metadata the results of any web service call to any service provider. You simply need to describe what service you want to call, and what to do with the results. Using the task code, you can create as many distinct tasks as you have services you want to call. Each description lives in a configuration file in 'config/modules', and is a simple properties file, like all other DSpace configuration files. The name of the configuration file is the task name you assign to it in `config/modules/curate.cfg`. There are a few required properties you must configure for any service, and for certain services, a few additional ones. An example will illustrate best.

ISSN to Publisher Name

Suppose items (holding journal articles) include 'dc.identifier.issn' when available. We might also want to catalog the publisher name (in 'dc.publisher'). The cataloger could look up the name given the ISSN in various sources, but this 'research' is tedious, costly and error-prone. There are many good quality, free web services that can furnish this information. So we will configure a `MetadataWebService` task to call a service, and then automatically assign the publisher name to the item metadata. As noted above, all that is needed is a description of the service, and what to do with the results. Create a new file in 'config/modules' called 'issn2pubname.cfg' (or whatever is mnemonically useful to you). The first property in this file describes the service in a 'template'. The template is just the URL to call the web service, with parameters to substitute values in. Here we will use the 'Sherpa/Romeo' service:

```
template=http://www.sherpa.ac.uk/romeo/api29.php?issn={dc.identifier.issn}
```

When the task runs, it will replace '{dc.identifier.issn}' with the value of that field in the item. If the field has multiple values, the first one will be used. As a web service, the call to the above URL will return an XML document containing information (including the publisher name) about that ISSN. We need to describe what to do with this response document, i.e. what elements we want to extract, and what to do with the extracted content. This description is encoded in a property called the 'datamap'. Using the example service above we might have:

```
datamap=//publisher/name=>dc.publisher, //romeocolor
```

Each separate instruction is separated by a comma, so there are 2 instructions in this map. The first instruction essentially says: find the XML element 'publisher name' and assign the value or values of this element to the 'dc.publisher' field of the item. The second instruction says: find the XML element 'romeocolor', but do not add it to the DSpace item metadata - simply add it to the task result string (so that it can be seen by the person running the task). You can have as many instructions as you like in a datamap, which means that you can retrieve multiple values from a single web service call. A little more formally, each instruction consists of one to three parts. The first (mandatory) part identifies the desired data in the response document. The syntax (here '//publisher/name') is an XPath 1.0 expression, which is the standard language for navigating XML trees. If the value is to be assigned to the DSpace item metadata, then 2 other parts are needed. The first is the 'mapping symbol' (here '=>'), which is used to determine how the assignment should be made. There are 3 possible mapping symbols, shown here with their meanings:

```
'->' mapping will add to any existing value(s) in the item field  
'=>' mapping will replace any existing value(s) in the item field  
'~>' mapping will add *only if* item field has no existing value(s)
```

The third part (here 'dc.publisher') is simply the name of the metadata field to be updated. These two mandatory properties (template and datamap) are sufficient to describe a large number of web services. All that is required to enable this task is to edit 'config/modules/curate.cfg', add 'issn2pubname' to the list of tasks:

```
plugin.named.org.dspace.curate.CurationTask = \  
... other defined tasks  
org.dspace.ctask.general.MetadataWebService = issn2pubname, \  
... other metadata web service tasks  
org.dspace.ctask.general.MetadataWebService = doi2crossref, \  
...
```

If you wish the task to be available in the Admin UI, see the [Invocation from the Admin UI](#) documentation (above) about how to configure it. The remaining sections describe some more specialized needs using the MetadataWebService task.

HTTP Headers

For some web services, protocol and other information is expressed not in the service URL, but in HTTP headers. Examples might be HTTP basic auth tokens, or requests for a particular media type response. In these cases, simply add a property to the configuration file (our example was 'issn2pubname.cfg') containing all headers you wish to transmit to the service:

```
headers=Accept: application/xml | Cache-Control: no-cache
```

You can specify any number of headers, just separate them with a 'double-pipe' ('|').

Transformations

One potential problem with the simple parameter substitutions performed by the task is that the service might expect a different format or expression of a value than the way it is stored in the item metadata. For example, a DOI service might expect a bare prefix/suffix notation ('10.000/12345'), whereas the DSpace metadata field might have a URI representation ('http://dx.doi.org/10.000/12345'). In these cases one can declare a 'transformation' of a value in the template. For example:

```
template=http://www.crossref.org/openurl/?id={doi:dc.relation.isversionof}&format=unixref
```

The 'doi:' prepended to the metadata field name declares that the value of the 'dc.relation.isversionof' field should be *transformed* before the substitution into the template using a transformation named 'doi'. The transformation is itself defined in the same configuration file as follows:

```
transform.doi=match 10. trunc 60
```

This would be read as: exclude the value string up to the occurrence of '10.', then truncate any characters after length 60. You may define as many transformations as you want in any task, although generally 1 or 2 will suffice. The keywords 'match', 'trunc', etc are names of 'functions' to be applied (in the order entered). The currently available functions are:

```
'cut' <number> = remove number leading characters  
'trunc' <number> = remove trailing characters after number length  
'match' <pattern> = start match at pattern  
'text' <characters> = append literal characters (enclose in ' ' when whitespace needed)
```

When the task is run, if the transformation results in an invalid state (e.g. cutting more characters than there are in the value), the un-transformed value will be used and the condition will be logged. Transformations may also be applied to values returned from the web service. That is, one can apply the transformation to a value before assigning it to a metadata field. In this case, the declaration occurs in the datamap property, not the template:


```
datamap=//publisher/name=>shorten:dc.publisher, //romeocolor
```

Here the task will apply the 'shorten' transformation (which must be defined in the same config file) before assigning the value to 'dc.publisher'.

Result String Programatic Use

Normally a task result string appears in a window in the admin UI after it has been invoked. The MetadataWebService task will concatenate all the values declared in the 'datamap' property and place them in the result string using the format: 'name:value name:value' for as many values as declared. In the example above we would get a string like 'publisher: Nature romeocolor: green'. This format is fine for simple display purposes, but can be tricky if the values contain spaces. You can override the space separator using an optional property 'separator' (put in the config file, with all other properties). If you use:

```
separator=|
```

for example, it becomes easy to parse the result string and preserve spaces in the values. This use of the result string can be very powerful, since you are essentially creating a map of returned values, which can then be used to populate a user interface, or any other way you wish to exploit the data (drive a workflow, etc).

Limits and Use

A few limitations should be noted. First, since the response parsing utilizes XPath, the service can only operate on XML, (not JSON) response documents. Most web services can provide either, so this should not be a major obstacle. The MetadataWebService can be used in many ways: showing an admin a value in the result string in a UI, run in a batch to update a set of items, etc. One excellent configuration is to wire these tasks into submission workflow, so that 'automatic cataloging' of many fields can be performed on ingest.

NoOp Curation Task

This task does absolutely nothing. It is intended as a starting point for developers and administrators wishing to learn more about the curation system.

Bitstream Format Profiler

The task with the taskname 'formatprofiler' (in the admin UI it is labeled "Profile Bitstream Formats") examines all the bitstreams in an item and produces a table ("profile") which is assigned to the result string. It is activated by default, and is configured to display in the administrative UI. The result string has the layout:

```
10 (K) Portable Network Graphics
5 (S) Plain Text
```

where the left column is the count of bitstreams of the named format and the letter in parentheses is an abbreviation of the repository-assigned support level for that format:

```
U  Unsupported
K  Known
S  Supported
```

The profiler will operate on any DSpace object. If the object is an item, then only that item's bitstreams are profiled; if a collection, all the bitstreams of all the items; if a community, all the items of all the collections of the community.

Required Metadata

The "requiredmetadata" task examines item metadata and determines whether fields that the web submission (`input-forms.xml`) marks as required are present. It sets the result string to indicate either that all required fields are present, or constructs a list of metadata elements that are required but missing. When the task is performed on an item, it will display the result for that item. When performed on a collection or community, the task is performed on each item, and will display the `/as/item` result. If all items in the community or collection have all required fields, that will be the last in the collection. If the task fails for any item (i.e. the item lacks all required fields), the process is halted. This way the results for the 'failed' items are not lost.

Virus Scan

The "vscan" task performs a virus scan on the bitstreams of items using the ClamAV software product. Clam AntiVirus is an open source (GPL) anti-virus toolkit for UNIX. A port for Windows is also available. The virus scanning curation task interacts with the ClamAV virus scanning service to scan the bitstreams contained in items, reporting on infection(s). Like other curation tasks, it can be run against a container or item, in the GUI or from the command line. It should be installed according to the documentation at <http://www.clamav.net>. It should not be installed in the dspace installation directory. You may install it on the same machine as your dspace installation, or on another machine which has been configured properly.

Setup the service from the ClamAV documentation.

This plugin requires a ClamAV daemon installed and configured for TCP sockets. Instructions for installing ClamAV ([http:// www.clamav.net/doc/latest/ clamdoc .pdf](http://www.clamav.net/doc/latest/clamdoc.pdf))

NOTICE: The following directions assume there is a properly installed and configured clamav daemon. Refer to links above for more information about ClamAV.

The Clam anti-virus database must be updated regularly to maintain the most current level of anti-virus protection. Please refer to the ClamAV documentation for instructions about maintaining the anti-virus database.

DSpace Configuration

In `[dspace]/config/modules/curate.cfg`, activate the task:

- Add the plugin to the comma separated list of curation tasks.

```
### Task Class implementations
plugin.named.org.dspace.curate.CurationTask = \
org.dspace.ctask.general.ProfileFormats = profileformats, \
org.dspace.ctask.general.RequiredMetadata = requiredmetadata, \
org.dspace.ctask.general.ClamScan = vscan
```

- Optionally, add the vscan friendly name to the configuration to enable it in the administrative it in the administrative user interface.

```
ui.tasknames = \
profileformats = Profile Bitstream Formats, \
requiredmetadata = Check for Required Metadata, \
vscan = Scan for Viruses
```

- In `[dspace]/config/modules`, edit configuration file `clamav.cfg`:

```
service.host = 127.0.0.1
Change if not running on the same host as your DSpace installation.
service.port = 3310
Change if not using standard ClamAV port
socket.timeout = 120
Change if longer timeout needed
scan.failfast = false
Change only if items have large numbers of bitstreams
```

- Finally, if desired virus scanning can be enabled as part of the submission process upload file step. In `[dspace]/config/modules`, edit configuration file `submission-curation.cfg`:

```
virus-scan = true
```

Task Operation from the Administrative user interface

Curation tasks can be run against container and item dspace objects by e-persons with administrative privileges. A curation tab will appear in the administrative ui after logging into DSpace:

1. Click on the curation tab.
2. Select the option configured in `ui.tasknames` above.
3. Select Perform.

Task Operation from the Item Submission user interface

If desired virus scanning can be enabled as part of the submission process upload file step. In `[dspace]/config/modules`, edit configuration file `submission-curation.cfg`:

```
virus-scan = true
```

Task Operation from the curation command line client

To output the results to the console:

```
[dSPACE]/bin/dSPACE curate -t vscan -i <handle of container or item dso> -r -
```

Or capture the results in a file:

```
[dSPACE]/bin/dSPACE curate -t vscan -i <handle of container or item dso> -r - > /<path...>/<name>
```

Table 1 – Virus Scan Results Table

GUI (Interactive Mode)	FailFast	Expectation
Container	T	Stop on 1 st Infected Bitstream
Container	F	Stop on 1 st Infected Item
Item	T	Stop on 1 st Infected Bitstream
Item	F	Scan all bitstreams
Command Line		
Container	T	Report on 1 st infected bitstream within an item/Scan all contained Items
Container	F	Report on all infected bitstreams/Scan all contained Items
Item	T	Report on 1 st infected bitstream
Item	F	Report on all infected bitstreams

Link Checkers

Two link checker tasks, BasicLinkChecker and MetadataValueLinkChecker can be used to check for broken or unresolvable links appearing in item metadata.

This task is intended as a prototype / example for developers and administrators who are new to the curation system.

These tasks are not configurable.

Basic Link Checker

BasicLinkChecker iterates over all metadata fields ending in "uri" (eg. dc.relation.uri, dc.identifier.uri, dc.source.uri ...), attempts a GET to the value of the field, and checks for a 200 OK response. Results are reported in a simple "one row per link" format.

Metadata Value Link Checker

MetadataValueLinkChecker parses all metadata fields for valid HTTP URLs, attempts a GET to those URLs, and checks for a 200 OK response. Results are reported in a simple "one row per link" format.

Microsoft Translator

Microsoft Translator uses the Microsoft Translate API to translate metadata values from one source language into one or more target languages.

This task can be configured to process particular fields, and use a default language if no authoritative language for an item can be found. Bing API v2 key is needed.

MicrosoftTranslator extends the more generic AbstractTranslator. This now seems wasteful, but a GoogleTranslator had also been written to extend AbstractTranslator. Unfortunately, Google has announced they are decommissioning free Translate API service, so this task hasn't been included in DSpace's general set of curation tasks.

Translated fields are added in addition to any existing fields, with the target language code in the 'language' column. This means that running a task multiple times over one item with the same configuration could result in duplicate metadata.

This task is intended as a prototype / example for developers and administrators who are new to the curation system.

Configure Microsoft Translator

An example configuration file can be found in [dspace]/config/modules/translator.cfg.

```
#-----#
#-----TRANSLATOR CURATION TASK CONFIGURATIONS-----#
#-----#
# Configuration properties used solely by MicrosoftTranslator #
# Curation Task (uses Microsoft Translation API v2) #
#-----#
## Translation field settings
##
## Authoritative language field
## This will be read to determine the original language an item was submitted in
## Default: dc.language
translate.field.language = dc.language
## Metadata fields you wish to have translated
```

```
#
translate.field.targets = dc.description.abstract, dc.title, dc.type
## Translation language settings
##
## If the language field configured in translate.field.language is not present
## in the record, set translate.language.default to a default source language
## or leave blank to use autodetection
#
translate.language.default = en
## Target languages for translation
#
translate.language.targets = de, fr
## Translation API settings
##
## Your Bing API v2 key and/or Google "Simple API Access" Key
## (note to Google users: your v1 API key will not work with Translate v2,
## you will need to visit https://code.google.com/apis/console and activate
## a Simple API Access key)
##
## You do not need to enter a key for both services.
#
translate.api.key.microsoft = YOUR_MICROSOFT_API_KEY_GOES_HERE
translate.api.key.google = YOUR_GOOGLE_API_KEY_GOES_HERE
```

6.3.12 Curation tasks in Jython

As mentioned in the "Scripted Tasks" chapter of [Curation System](#), you can write your curation tasks in several languages, including Jython (a flavour of Python running on JVM).

Setting up scripted tasks in Jython

1. Download the latest Jython installer jar (e.g. [jython-installer-2.5.3.jar](http://www.jython.org/downloads.html)) from <http://www.jython.org/downloads.html>
2. Get `jython.jar` and the `Lib` directory.

- a. either unzip the installer jar:

```
unzip -d [dspace]/lib/ jython-installer-2.5.3.jar jython.jar Lib/unzip -
d [dspace]/webapps/xmlui/WEB-INF/lib/ jython-installer-2.5.3.jar
jython.jar Lib/
```

- b. or use it to install Jython:

```
java -jar jython-installer-2.5.3.jar --console
```

Note: Installation location doesn't matter, this is not necessary for DSpace. You can safely delete it after you retrieve `jython.jar` and `Lib`.

3. Install Jython to DSpace classpaths (step 2a already did this for you):

- a. The goal is to put `jython.jar` and the `jython Lib/` directory into **every** DSpace classpath you intend to use, so it must be installed in **both** `[dspace]/lib` and the webapp that deploys to Tomcat (if you want to run from the UI) - `[dspace]/webapps/xmlui/WEB-INF/lib/`. There are no special maven/pom extensions - just copy in the jar and `Lib/`.
 - b. You **can** use symlinks if you wish as long as `allowLinking` ([Tomcat <=7](#), [Tomcat 8](#)) is set to true in that context's configuration. However, be warned that [Tomcat documentation lists `allowLinking="true"` as a possible security concern](#).
 - c. Note: Older versions of Jython mention the need for `jython-engine.jar` to implement JSR-223. Don't worry about that, new Jython versions, e.g. 2.5.3 don't require this.
4. Configure the curation framework to be aware of your new task(s):
- a. set up the location of scripted tasks in the curation system. This means simply adding a property to `[dspace]/config/modules/curate.cfg`:
`script.dir=${dspace.dir}/ctscripts`
 - b. in this directory, create a text file named `"task.catalog"`. This is a Java properties file where lines beginning with `'#'` are comments. Add a line for each task you write. The syntax is following:

```
# logical task name = script engine name|file name|constructor invocation
mytask=python|mytask.py|MyTask()
```

Notes:

- **don't** put spaces around the pipe character or you'll get an error similar to this one:
`ERROR org.dspace.curate.TaskResolver @ Script engine: 'python ' is not installed`
 - The "script engine name" is what ever name (or alias) jython registers in the JVM. You can use both "python" and "jython" as engine name (tested on jython 2.5.3).
 - The logical task name can't conflict with existing (java) task names, but otherwise any single-word token can be used.
 - The file name is just the script file name in the `script.dir` directory
 - "constructor invocation" is the language specific way to create an object that implements the task interface - it's `ClassName()` for Python
- c. If you want pretty names in the UI, configure other `curate.cfg` properties - see `"ui.tasknames"` (or groups etc)
5. Write your task.

In the directory configured above, create your task (with the name configured in `"task.catalog"`).

The basic requirement of any scripted task is that it implements the [ScriptedTask](#) Java interface.

So for our example, the `mytask.py` file might look like this:

```
from org.dspace.curate import ScriptedTask
class MyTask(ScriptedTask):
    def init(self, curator, taskName):
        print "initializing with Jython"
    def performDso(self, dso):
```

```
        print "perform on dso"  
        return 0  
def performId(self, context, id):  
    print "perform on id %s" % (id)  
    return 0
```

6. Invoke the task.

You can do this the same way you would invoke any task (from command line, in the admin UI, etc). The advantage of scripting is that you do not need to restart your servlet container to test changes; each task's source code is reloaded when you launch the task, so you can just put the updated script in place. Example of invocation from command line:

```
[dspace]/bin/dspace curate -t mytask -i 123456789/123 -r -
```

Note: "-r -" means that the script's standard output will be directed to the console. You can read more details in the "On the command line" chapter of the [Curation System](#) page.

See also

- [Curation System](#) page in the official documentation
- [Nailgun](#) - for speeding up repeated runs of a dspace command from the command line

6.4 Date parser tester

Some parts of DSpace use a custom date/time parser (`org.dspace.util.MultiFormatDateParser`) which is driven by a table of regular expressions, so it can match any of a variety of formats. The table is found in `config/spring/api/discovery-solr.xml`. To test new and altered rules, you can use the DSpace command line tool's `validate-date` command. You can simply pass it a date/time string on the command line (`dspace validate-date 01-01-2015`). You can pipe a stream of strings to be validated, one per line (`dspace validate-date < test.data`). Or you can have it prompt you for each string to be tested (`dspace validate-date`).

7 DSpace Reference


- [Configuration Reference](#)
- [Directories and Files](#)
- [Metadata and Bitstream Format Registries](#)
- [Architecture](#)
 - [Application Layer](#)
 - [Business Logic Layer](#)
 - [DSpace Services Framework](#)
 - [Storage Layer](#)
- [History](#)
 - [Changes in 5.x](#)
 - [Changes in 4.x](#)
 - [Changes in 3.x](#)
 - [Changes in 1.8.x](#)
 - [Changes in 1.7.x](#)
 - [Changes in 1.6.x](#)
 - [Changes in 1.5.x](#)
 - [Changes in 1.4.x](#)
 - [Changes in 1.3.x](#)
 - [Changes in 1.2.x](#)
 - [Changes in 1.1.x](#)
- [DSpace Item State Definitions](#)

7.1 Configuration Reference

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

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

- [General Configuration](#) - addresses general conventions used with configuring not only the `dspace.cfg` file, but other configuration files which use similar conventions.
- [The `build.properties` Configuration Properties File](#) - specifies the basic `build.properties` file settings (these basic settings are used when building/installing/upgrading DSpace)
- [The `dspace.cfg` Configuration Properties File](#) - specifies the basic `dspace.cfg` file settings (these settings are used when DSpace is actually running)
- [Optional or Advanced Configuration Settings](#) - contain other more advanced settings that are optional in the `dspace.cfg` configuration file.

 As of version 1.8 much of the DSpace configuration has been moved to discrete configuration files related to specific functionality and is documented in subsequent sections of this document.

The full table of contents follows:

- 1 [General Configuration](#)
 - 1.1 [Input Conventions](#)
 - 1.2 [Update Reminder](#)
- 2 [The build.properties Configuration Properties File](#)
- 3 [The dspace.cfg Configuration Properties File](#)
 - 3.1 [Main DSpace Configurations](#)
 - 3.2 [DSpace Database Configuration](#)
 - 3.3 [DSpace Email Settings](#)
 - 3.3.1 [Wording of E-mail Messages](#)
 - 3.4 [File Storage](#)
 - 3.5 [SRB \(Storage Resource Brokerage\) File Storage](#)
 - 3.6 [Logging Configuration](#)
 - 3.7 [General Plugin Configuration](#)
 - 3.8 [Configuring the Search Engine](#)
 - 3.9 [Delegation Administration: Authorization System Configuration](#)
 - 3.9.1 [Login as feature](#)
 - 3.10 [Restricted Item Visibility Settings](#)
 - 3.11 [Proxy Settings](#)
 - 3.12 [Configuring Media Filters](#)
 - 3.13 [Crosswalk and Packager Plugin Settings](#)
 - 3.13.1 [Configurable MODS Dissemination Crosswalk](#)
 - 3.13.2 [XSLT-based Crosswalks](#)
 - 3.13.2.1 [Testing XSLT Crosswalks](#)
 - 3.13.3 [Configurable Qualified Dublin Core \(QDC\) dissemination crosswalk](#)
 - 3.13.4 [Configuring Crosswalk Plugins](#)
 - 3.13.5 [Configuring Packager Plugins](#)
 - 3.14 [Event System Configuration](#)
 - 3.15 [Embargo](#)
 - 3.16 [Checksum Checker Settings](#)
 - 3.17 [Item Export and Download Settings](#)
 - 3.18 [Subscription Emails](#)
 - 3.19 [Hiding Metadata](#)
 - 3.20 [Settings for the Submission Process](#)
 - 3.21 [Configuring the Sherpa/RoMEO Publishers Policy Database Integration](#)
 - 3.22 [Configuring Creative Commons License](#)
 - 3.23 [WEB User Interface Configurations](#)

- 3.24 [Browse Index Configuration](#)
 - 3.24.1 [Defining the storage of the Browse Data](#)
 - 3.24.2 [Defining Sort Options](#)
 - 3.24.3 [Browse Index Normalization Rule Configuration](#)
 - 3.24.4 [Other Browse Options](#)
 - 3.24.5 [Browse Index Authority Control Configuration](#)
 - 3.24.6 [Tag cloud](#)
- 3.25 [Author \(Multiple metadata value\) Display](#)
- 3.26 [Links to Other Browse Contexts](#)
- 3.27 [Recent Submissions](#)
- 3.28 [Submission License Substitution Variables](#)
- 3.29 [Syndication Feed \(RSS\) Settings](#)
- 3.30 [OpenSearch Support](#)
- 3.31 [Content Inline Disposition Threshold](#)
- 3.32 [Multi-file HTML Document/Site Settings](#)
- 3.33 [Sitemap Settings](#)
- 3.34 [Authority Control Settings](#)
- 3.35 [JSPUI Upload File Settings](#)
- 3.36 [JSP Web Interface \(JSPUI\) Settings](#)
- 3.37 [JSPUI Configuring Multilingual Support](#)
 - 3.37.1 [Setting the Default Language for the Application](#)
 - 3.37.2 [Supporting More Than One Language](#)
 - 3.37.2.1 [Changes in dspace.cfg](#)
 - 3.37.2.2 [Related Files](#)
- 3.38 [JSPUI Item Mapper](#)
- 3.39 [Display of Group Membership](#)
- 3.40 [JSPUI / XMLUI SFX Server](#)
- 3.41 [JSPUI Item Recommendation Setting](#)
- 3.42 [Controlled Vocabulary Settings](#)
- 3.43 [XMLUI Specific Configuration](#)
- 4 [Optional or Advanced Configuration Settings](#)
 - 4.1 [The Metadata Format and Bitstream Format Registries](#)
 - 4.1.1 [Metadata Format Registries](#)
 - 4.1.2 [Bitstream Format Registry](#)
 - 4.2 [XPDF Filter](#)
 - 4.2.1 [Installation Overview](#)
 - 4.2.2 [Install XPDF Tools](#)
 - 4.2.3 [Fetch and install jai_imageio JAR](#)
 - 4.2.4 [Edit DSpace Configuration](#)
 - 4.2.5 [Build and Install](#)
 - 4.3 [Configuring Usage Instrumentation Plugins](#)
 - 4.3.1 [The Passive Plugin](#)
 - 4.3.2 [The Tab File Logger Plugin](#)

- [4.4 JSPUI: Per item visual indicators for browse and search results](#)

7.1.1 General Configuration

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

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

Input Conventions

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

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

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

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

Examples:

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

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

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

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

Update Reminder

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

It is important to remember that there are *multiple `dspace.cfg` files in several places after an installation of DSpace.* The only two you should notice are:

1. The "source" file that is found in `[dspace-source]/dspace/config/dspace.cfg`
2. The "runtime" file that is found in `[dspace]/config/dspace.cfg`

The runtime file is supposed to be the **copy** of the source file, which is considered the master version.

However, the DSpace server and command programs only look at the *runtime* configuration file, so when you are revising your configuration values, it is tempting to *only edit the runtime file*. **DO NOT** do this.

Always make the same changes to the source version of `dspace.cfg` in addition to the runtime file. The two files should always be identical, since the source `dspace.cfg` will be the basis of your next upgrade.

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

```
cd [dspace-source]/dspace/  
mvn package  
cd [dspace-source]/dspace/target/dspace-installer  
ant update_configs
```

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

Please note that there are in fact two options available, choose whichever you prefer :-

- "ant update_configs" ==> Moves existing configs in `[dspace]/config/` to `*.old` files and replaces them with what is in `[dspace-source]/dspace/config/`
- "ant -Doverwrite=false update_configs" ==> Leaves existing configs in `[dspace]/config/` intact. Just copies new configs from `[dspace-source]/dspace/config/` over to `*.new` files.

7.1.2 The build.properties Configuration Properties File

As of DSpace 3.0, we now provide a `[dspace-source]/build.properties` as an easy means of configuration a subset of properties before you build DSpace (by running "mvn package" or similar). Any properties set in this `build.properties` file will be automatically copied over to your final `dspace.cfg` file as part of the Maven build process.

Users/Developers may also choose to copy the `build.properties` under a different name for different environments (e.g. development, test & production), and choose which environment to build DSpace for by passing a "-Denv" (environment) flag to the Maven build process (e.g. "mvn package -Denv=test" would build DSpace using a custom "test.properties" file).

Here's a basic example of how `build.properties` (or any `*.properties`) file may be used to simplify installation & development:

1. A developer or user downloads a copy of DSpace to build & install
2. He/She can edit the `[dspace-source]/build.properties` to specify the very basic settings for building & installing DSpace
 - a. OR, alternatively he/she can copy/rename the "build.properties" to a different `*.properties` file & edit it. For example, you could choose to create a separate properties file for each environment (`dev.properties`, `test.properties`, `prod.properties`)
3. He/She can then build the DSpace Installation Package using the `*.properties` file they choose
 - a. Running simply "mvn package" will always use the default "build.properties" settings.
 - b. Passing in the "-Denv" (environment) flag, will cause the build process to use a custom properties file. Some examples:
 - i. "mvn package -Denv=test" would build DSpace using a custom file named `[dspace-source]/test.properties`
 - ii. "mvn package -Denv=local" would build DSpace using a custom file named `[dspace-source]/local.properties`
 - iii. "mvn package -Denv=john" would build DSpace using a custom file named `[dspace-source]/john.properties`
4. No matter which build options are used, the values in the enabled properties file will be automatically copied over to your `[dspace-source]/dspace/target/dspace-[version]-build/dspace.cfg` file in the DSpace Installation Package. That way they can be installed using the appropriate Apache Ant command (see [Installing DSpace](#) for all the details of the full install.)



build.properties file is only used with building/compiling DSpace

It is worth noting that the `[dspace-source]/build.properties` file (or custom properties file) is ONLY used in the act of building/installing/upgrading DSpace. During that build/install/upgrade process, any settings in your "build.properties" file (or custom properties file) are automatically copied over to the "dspace.cfg" file. Once DSpace is installed, it only uses the settings in your `[dspace]/config/dspace.cfg` file.



Do not remove or comment out settings in build.properties

When you edit the "build.properties" file (or a custom `*.properties` file), take care not to remove or comment out any settings. Doing so, may cause your final "dspace.cfg" file to be misconfigured with

regards to that particular setting. Instead, if you wish to remove/disable a particular setting, just clear out its value. For example, if you don't want to be notified of new user registrations, ensure the "mail.registration.notify" setting has no value, e.g.

```
mail.registration.notify=
```

As another example, if you are running the DSpace UI of your choice (XMLUI or JSPUI) directly under tomcat's root, leave "dspace.ui" empty but do not delete the setting, e.g.

```
dspace.ui=
```

You may add new settings to your build.properties or custom *.properties

Based on your institution's needs, you may wish to add settings to your own build.properties (or custom *.properties) file. This is actually a relatively easy process.

Any existing DSpace configuration (any config in dspace.cfg or in any configuration file under [dspace-src]/dspace/config/modules/*.cfg) can be "moved" into your local build.properties file via the following process:

1. First, copy the existing configuration from the *.cfg file into your local build.properties file. You can actually choose to rename this configuration in build.properties, if it makes more sense. Essentially, the name of the new configuration in build.properties is entirely up to you.
 - a. For example, if you want to copy the LDAP "provider_url" from [dspace-src]/dspace/config/modules/authentication-ldap.cfg to your build.properties, you may wish to rename it to "ldap.provider_url" within build.properties
 - b. You can also choose to keep the name of the configuration the same in build.properties. For example, if you wish to move the "xmlui.google.analytics.key" (from dspace.cfg) to your build.properties, you could keep the name the same.
2. Second, you will need to modify the corresponding configuration file (the config file you copied the setting from) so that it now references your newly added build.properties setting. This is achieved by using the "\${setting-in-build.properties}" placeholder.
 - a. For example, to reference a new "ldap.provider_url" setting in build.properties (mentioned in 1.a above), just modify the [dspace-src]/dspace/config/modules/authentication-ldap.cfg file to have a line that says `provider_url=${ldap.provider_url}` (The first part is the name of the actual config in authentication-ldap.cfg, and the second part is the name of the config in build.properties)
 - b. Another example: To reference a new "xmlui.google.analytics.key" setting in build.properties (mentioned in 1.b above), just modify the [dspace-src]/dspace/config/dspace.cfg file to have a line that says `xmlui.google.analytics.key=${xmlui.google.analytics.key}` (The first part is the name of the actual config in dspace.cfg, and the second part is the name of the config in build.properties)

3. Finally, rebuild DSpace (using Maven), and redeploy (using Ant). The new settings in your `build.properties` file will automatically be copied into your configuration file during the rebuild process.

7.1.3 The `dspace.cfg` Configuration Properties File

The `dspace.cfg` contains basic information about a DSpace installation, including system path information, network host information, and other like items. It is the primary configuration file for DSpace, used by DSpace when it is actively running.

In ordinary use, this file is assumed to be `[dspace]/config/dspace.cfg`. If you define a system property – `Ddspace.configuration=/some/path/to/a/file` then that file will be used instead.

Main DSpace Configurations

Property:	<code>dspace.dir</code>
Example Value:	<code>/dspace</code>
Informational Note:	<p>Root directory of DSpace installation. Omit the trailing slash '/'. Note that if you change this, there are several other parameters you will probably want to change to match, e.g. <code>assetstore.dir</code>.</p> <p><i>(On Windows be sure to use forward slashes for the directory path! For example: "C:/dspace" is a valid path for Windows.)</i></p>
Property:	<code>dspace.hostname</code>
Example Value:	<code>dspace.hostname = dspace.mysu.edu</code>
Informational Note:	Fully qualified hostname; do not include port number.
Property:	<code>dspace.baseUrl</code>
Example Value:	http://dspacetest.myu.edu:8080
Informational Note:	Main URL at which DSpace Web UI webapp is deployed. Include any port number, but do not include the trailing '/'.
Property:	<code>dspace.url</code>
Example Value:	<code>dspace.url = http://dspacetest.myu.edu:8080/xmlui</code>
Informational note	URL that determines whether JSPUI or XMLUI will be loaded by default. Include port number etc., but NOT trailing slash. Alternatively to the example, this url can have <code>/jspui</code> at the end if you are using <code>jspui</code> instead of <code>xmlui</code> . You can also opt to run your UI app as your servlet engine's "ROOT" webapp. In that case, ensure that you remove <code>/xmlui</code> or <code>/jspui</code> .
Property:	<code>dspace.oai.url</code>
Example Value:	<code>dspace.oai.url = \${dspace.baseUrl}/oai</code>
Informational note:	The base URL of the OAI webapp (do not include <code>/request</code>).

Property:	<code>dspace.name</code>
Example Value:	<code>dspace.name = DSpace at My University</code>
Informational Note:	Short and sweet site name, used throughout Web UI, e-mails and elsewhere (such as OAI protocol)

DSpace Database Configuration

Many of the database configurations are software-dependent. That is, it will be based on the choice of database software being used. Currently, DSpace properly supports PostgreSQL and Oracle.

Property:	<code>db.url</code>
Example Value:	<code>db.url = jdbc:postgresql://localhost:5432/dspace_services</code>
Informational Note:	The above value is the default value when configuring with PostgreSQL. When using Oracle, use this value: <code>jdbc.oracle.thin:@//host:port/dspace</code>
Property:	<code>db.username</code>
Example Value:	<code>db.username = dspace</code>
Informational Note:	In the installation directions, the administrator is instructed to create the user "dspace" who will own the database "dspace".
Property:	<code>db.password</code>
Example Value:	<code>db.password = dspace5</code>
Informational Note:	This is the password that was prompted during the installation process (cf. 3.2.3. Installation)
Property:	<code>db.schema</code>
Example Value:	<code>db.schema = vra</code>
Informational Note:	If your database contains multiple schemas, you can avoid problems with retrieving the definitions of duplicate objects by specifying the schema name here that is used for DSpace by uncommenting the entry. This property is optional.
Property:	

	<code>db.maxconnections</code>
Example Value:	<code>db.maxconnections = 30</code>
Informational Note:	Maximum number of Database connections in the connection pool
Property:	<code>db.maxwait</code>
Example Value:	<code>db.maxwait = 5000</code>
Informational Note:	Maximum time to wait before giving up if all connections in pool are busy (in milliseconds).
Property:	<code>db.maxidle</code>
Example Value:	<code>db.maxidle = -1</code>
Informational Note:	Maximum number of idle connections in pool. (-1 = unlimited)
Property:	<code>db.statementpool</code>
Example Value:	<code>db.statementpool = true</code>
Informational Note:	Determines if prepared statement should be cached. (Default is set to true)
Property:	<code>db.poolname</code>
Example Value:	<code>db.poolname = dspacepool</code>
Informational Note:	Specify a name for the connection pool. This is useful if you have multiple applications sharing Tomcat's database connection pool. If nothing is specified, it will default to 'dspacepool'
Property:	<code>db.jndi</code>
Example Value:	<code>db.jndi = jdbc/dspace</code>
Informational Note:	Specify the name of a configured connection pool to be fetched from a directory using JNDI. If this property is not configured or no such pool can be retrieved, then DSpace will fall back to creating its own pool using the other <code>db.*</code> properties.

DSpace Email Settings

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

DSpace will look up a `javax.mail.Session` object in JNDI and, if found, will use that to send email. Otherwise it will create a Session using some of the properties detailed below.

Property:	<code>mail.server</code>
Example Value:	<code>mail.server = smtp.my.edu</code>
Informational Note:	The address on which your outgoing SMTP email server can be reached.
Property:	<code>mail.server.username</code>
Example Value:	<code>mail.server.username = myusername</code>
Informational Note:	SMTP mail server authentication username, if required. This property is optional.
Property:	<code>mail.server.password</code>
Example Value:	<code>mail.server.password = mypassword</code>
Informational Note:	SMTP mail server authentication password, if required. This property is optional/
Property:	<code>mail.server.port</code>
Example Value:	<code>mail.server.port = 25</code>
Informational Note:	The port on which your SMTP mail server can be reached. By default, port 25 is used. Change this setting if your SMTP mailserver is running on another port. This property is optional.
Property:	<code>mail.from.address</code>
Example Value:	<code>mail.from.address = dspace-noreply@myu.edu</code>
Informational Note:	The "From" address for email. Change the 'myu.edu' to the site's host name.
Property:	<code>feedback.recipient</code>

Example Value:	<code>feedback.recipient = dspace-help@myu.edu</code>
Informational Note:	When a user clicks on the feedback link/feature, the information will be sent to the email address of choice. This configuration is currently limited to only one recipient. Since DSpace 4.0, this is also the email address displayed on the contacts page.
Property:	<code>mail.admin</code>
Example Value:	<code>mail.admin = dspace-help@myu.edu</code>
Informational Note:	Email address of the general site administrator (Webmaster)
Property:	<code>alert.recipient</code>
Example Value:	<code>alert.recipient = john.doe@myu.edu</code>
Informational Note:	Enter the recipient for server errors and alerts. This property is optional.
Property:	<code>registration.notify</code>
Example Value:	<code>registration.notify = mike.smith@myu.edu</code>
Informational Note:	Enter the recipient that will be notified when a new user registers on DSpace. This property is optional.
Property:	<code>mail.charset</code>
Example Value:	<code>mail.charset = UTF-8</code>
Informational Note:	Set the default mail character set. This may be over-ridden by providing a line inside the email template ' <i>charset: <encoding></i> ', otherwise this default is used.
Property:	<code>mail.allowed.referrers</code>
Example Value:	<code>mail.allowed.referrers = localhost</code>
Informational Note:	A comma separated list of hostnames that are allowed to refer browsers to email forms. Default behavior is to accept referrals only from <i>dspace.hostname</i> . This property is optional.
Property:	<code>mail.extraproperties</code>

Example Value:	<pre>mail.extraproperties = mail.smtp.socketFactory.port=465, \ mail.smtp.socketFactory.class=javax.net.ssl.SSLSocketFactory, \ mail.smtp.socketFactory.fallback=false</pre>
Informational Note:	If you need to pass extra settings to the Java mail library. Comma separated, equals sign between the key and the value. This property is optional.
Property:	<code>mail.server.disabled</code>
Example Value:	<code>mail.server.disabled = false</code>
Informational Note:	An option is added to disable the mailservier. By default, this property is set to 'false'. By setting value to 'true', DSpace will not send out emails. It will instead log the subject of the email which should have been sent. This is especially useful for development and test environments where production data is used when testing functionality. This property is optional.
Property:	<code>mail.session.name</code>
Example Value:	<code>mail.session.name = myDSpace</code>
Informational Note:	Specifies the name of a <code>javax.mail.Session</code> object stored in JNDI under <code>java:comp/env/mail</code> . The default value is "Session".
Property:	<code>default.language</code>
Example Value:	<code>default.language = en_US</code>
Informational Note:	If no other language is explicitly stated in the <i>input-forms.xml</i> , the default language will be attributed to the metadata values.

Wording of E-mail Messages

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

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

```
config/emails/change_password
config/emails/register
```

File Storage

DSpace supports two distinct options for storing your repository bitstreams (uploaded files). The files are not stored in the database in which Metadata, user information, ... are stored. An assetstore is a directory on your server, on which the bitstreams are stored and consulted afterwards. The usage of different assetstore directories is the default "technique" in DSpace. The parameters below define which assetstores are present, and which one should be used for newly incoming items. As an alternative, DSpace can also use SRB (Storage Resource Brokerage) as an alternative. See [SRB File Storage](#) for details regarding SRB.

Property:	<code>assetstore.dir</code>
Example Value:	<code>assetstore.dir = \${dspace.dir}/assetstore</code>
Informational Note:	This is Asset (bitstream) store number 0 (Zero). You need not place your assetstore under the <i>/dspace</i> directory, but may want to place it on a different logical volume on the server that DSpace resides. So, you might have something like this: <code>assetstore.dir = /storevgm/assestore .</code>
Property:	<pre>assetstore.dir.1 assetstore.dir.2</pre>
Example Value:	<pre>assetstore.dir.1 = /second/assetstore assetstore.dir.2 = /third/assetstore</pre>
Informational Note:	This property specifies extra asset stores like the one above, counting from one (1) upwards. This property is commented out (#) until it is needed.
Property:	<code>assetstore.incoming</code>
Example Value:	<code>assetstore.incoming = 1</code>
Informational Note:	

Specify the number of the store to use for new bitstreams with this property. The default is 0 [zero] which corresponds to the 'assestore.dir' above. As the asset store number is stored in the item metadata (in the database), always keep the assetstore numbering consistent and don't change the asset store number in the item metadata.

Be Careful

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

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

Please Note: When adding additional storage configuration, you will then need to uncomment and declare

```
assestore.incoming = 1
```

SRB (Storage Resource Brokerage) File Storage

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

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

Property:	<code>srb.hosts.1</code>
Example value:	<code>srb.hosts.1 = mysrbmcatzhost.myu.edu</code>
Property:	<code>srb.port.1</code>
Example value:	<code>srb.port.1 = 5544</code>
Property:	<code>srb.mcatzzone.1</code>

Example value:	<code>srb.mcatzone.1 = mysrbzone</code>
Informational Note:	Your SRB Metadata Catalog Zone. An SRB Zone (or zone for short) is a set of SRB servers 'brokered' or administered through a single MCAT. Hence a zone consists of one or more SRB servers along with one MCAT-enabled server. Any existing SRB system (version 2.x.x and below) can be viewed as an SRB zone. For more information on zones, please check http://www.sdsc.edu/srb/index.php/Zones .
Property:	<code>srb.mdasdomainname.1</code>
Example Value:	<code>srb.mdasdomainname.1 = mysrbdomain</code>
Informational Note:	Your SRB domain. This domain should be created under the same zone, specified in <code>srb.mcatzone</code> . Information on domains is included here http://www.sdsc.edu/srb/index.php/Zones .
Property:	<code>srb.defaultstorageresource.1</code>
Example Value:	<code>srb.defaultstorageresource.1 = mydefaultsrbresource</code>
Informational Note:	Your default SRB Storage resource.
Property:	<code>srb.username.1</code>
Example Value:	<code>srb.username.1 = mysrbuser</code>
Informational Note:	Your SRB Username.
Property:	<code>srb.password.1</code>
Example Value:	<code>srb.password.1 = mysrbpassword</code>
Informational Note:	Your SRB Password.
Property:	<code>srb.homedirectory.1</code>
Example Value:	<pre>srb.homedirectory.1 = /mysrbzone/home/ mysrbuser.mysrbdomain</pre>

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

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

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

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

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

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

Logging Configuration

Property:	<code>log.init.config</code>
Example Value:	<code>log.init.config = \${dspace.dir}/config/log4j.properties</code>
Informational Note:	<p>This is where your logging configuration file is located. You may override the default log4j configuration by providing your own. Existing alternatives are:</p> <pre style="border: 1px dashed gray; padding: 10px;">log.init.config = \${dspace.dir}/config/log4j.properties log.init.config = \${dspace.dir}/config/log4j-console.properties</pre>
Property:	<code>log.dir</code>
Example value:	<code>log.dir = \${dspace.dir}/log</code>
Informational Note:	This is where to put the logs. (This is used for initial configuration only)
Property:	<code>useProxies</code>
Example Value:	<code>useProxies = true</code>
Informational Note:	<p>If your DSpace instance is protected by a proxy server, in order for log4j to log the correct IP address of the user rather than of the proxy, it must be configured to look for the X-Forwarded-For header. This feature can be enabled by ensuring this setting is set to <i>true</i>. This also affects IPAuthentication, and should be enabled for that to work properly if your installation uses a proxy server.</p>


Previous releases of DSpace provided an example `${dspace.dir}/config/log4j.xml` as an alternative to `log4j.properties`. This caused some confusion and has been removed. log4j continues to support both Properties and XML forms of configuration, and you may continue (or begin) to use any form that log4j supports.

General Plugin Configuration

Property:	<code>plugin.classpath</code>
Example Value:	<code>/opt/dspace/plugins/aPlugin.jar:/opt/dspace/moreplugins</code>

Property:	<code>plugin.classpath</code>
Informational Note:	Search path for third-party plugin classes. This is a colon-separated list of directories and JAR files, each of which will be searched for plugin classes after looking in all the places where DSpace classes are found. In this way you can designate one or more locations for plugin files which will not be affected by DSpace upgrades.

Configuring the Search Engine

 Since DSpace 4.0 the advanced search module named Discovery (based on Apache SOLR) is the default search provider. It provides up-to-date features, such as filtering/faceting, hit highlighting, search snippets, etc.

A detailed documentation is available for customization, see [Discovery](#)

Please refer to [Legacy methods for re-indexing content](#) if you want re-enable and customize the "legacy" DSpace search engine (based on Apache Lucene).

Handle Server Configuration

The CNRI Handle system is a 3rd party service for maintaining persistent URL's. For a nominal fee, you can register a handle prefix for your repository. As a result, your repository items will be also available under the links <http://handle.net/<<handle prefix>>/<<item id>>>. As the base url of your repository might change or evolve, the persistent handle.net URL's secure the consistency of links to your repository items. For complete information regarding the Handle server, the user should consult [The Handle Server](#) section of Installing DSpace.

Property:	<code>handle.canonical.prefix</code>
Example Value	<code>handle.canonical.prefix = http://hdl.handle.net/</code> <code>handle.canonical.prefix = \${dspace.url}/handle/</code>
Informational Note:	Canonical Handle URL prefix. By default, DSpace is configured to use http://hdl.handle.net/ as the canonical URL prefix when generating <code>dc.identifier.uri</code> during submission, and in the 'identifier' displayed in item record pages. If you do not subscribe to CNRI's handle service, you can change this to match the persistent URL service you use, or you can force DSpace to use your site's URL, e.g. <code>handle.canonical.prefix = \${dspace.url}/handle/</code> . Note that this will not alter <code>dc.identifier.uri</code> metadata for existing items (only for subsequent submissions).

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

Delegation Administration: Authorization System Configuration

It is possible to delegate the administration of Communities and Collections. This functionality eliminates the need for an Administrator Superuser account for these purposes. An EPerson that will be attributed Delegate Admin rights for a certain community or collection will also "inherit" the rights for underlying collections and items . As a result, a community admin will also be collection admin for all underlying collections. Likewise, a collection admin will also gain admin rights for all the items owned by the collection.

Authorization to execute the functions that are allowed to user with WRITE permission on an object will be attributed to be the ADMIN of the object (e.g. community/collection/admin will be always allowed to edit metadata of the object). The default will be "*true*" for all the configurations.

Community Administration: Subcommunities and Collections	
Property:	<code>core.authorization.community-admin.create-subelement</code>
Example Value:	<code>core.authorization.community-admin.create-subelement = true</code>
Informational Note:	Authorization for a delegated community administrator to create subcommunities or collections.
Property:	<code>core.authorization.community-admin.delete-subelement</code>
Example Value:	<code>core.authorization.community-admin.delete-subelement = true</code>
Informational Note:	

	Authorization for a delegated community administrator to delete subcommunities or collections.
Community Administration: Policies and The group of administrators	
Property:	<code>core.authorization.community-admin.policies</code>
Example Value:	<code>core.authorization.community-admin.policies = true</code>
Informational Note:	Authorization for a delegated community administrator to administrate the community policies.
Property:	<code>core.authorization.community-admin.admin-group</code>
Example Value:	<code>core.authorization.community-admin.admin-group = true</code>
Informational Note:	Authorization for a delegated community administrator to edit the group of community admins.
Community Administration: Collections in the above Community	
Property:	<code>core.authorization.community-admin.collection.policies</code>
Example Value:	<code>core.authorization.community-admin.collection.policies = true</code>
Informational Note:	Authorization for a delegated community administrator to administrate the policies for underlying collections.
Property:	<code>core.authorization.community-admin.collection.template-item</code>
Example Value:	<code>core.authorization.community-admin.collection.template-item = true</code>
Informational Note:	Authorization for a delegated community administrator to administrate the item template for underlying collections.
Property:	<code>core.authorization.community-admin.collection.submitters</code>
Example Value:	<code>core.authorization.community-admin.collection.submitters = true</code>
Informational Note:	Authorization for a delegated community administrator to administrate the group of submitters for underlying collections.

Property:	<code>core.authorization.community-admin.collection.workflows</code>
Example Value:	<code>core.authorization.community-admin.collection.workflows = true</code>
Informational Note:	Authorization for a delegated community administrator to administrate the workflows for underlying collections.
Property:	<code>core.authorization.community-admin.collection.admin-group</code>
Example Value:	<code>core.authorization.community-admin.collection.admin-group = true</code>
Informational Note:	Authorization for a delegated community administrator to administrate the group of administrators for underlying collections.
Community Administration: Items Owned by Collections in the Above Community	
Property:	<code>core.authorization.community-admin.item.delete</code>
Example Value:	<code>core.authorization.community-admin.item.delete = true</code>
Informational Note:	Authorization for a delegated community administrator to delete items in underlying collections.
Property:	<code>core.authorization.community-admin.item.withdraw</code>
Example Value:	<code>core.authorization.community-admin.item.withdraw = true</code>
Informational Note:	Authorization for a delegated community administrator to withdraw items in underlying collections.
Property:	<code>core.authorization.community-admin.item.reinstate</code>
Example Value:	<code>core.authorization.community-admin.item.reinstate = true</code>
Informational Note:	Authorization for a delegated community administrator to reinstate items in underlying collections.
Property:	<code>core.authorization.community-admin.item.policies</code>
Example Value:	<code>core.authorization.community-admin.item.policies = true</code>
Informational Note:	Authorization for a delegated community administrator to administrate item policies in underlying collections.

Community Administration: Bundles of Bitstreams, related to items owned by collections in the above Community	
Property:	<code>core.authorization.community-admin.item.create-bitstream</code>
Example Value:	<code>core.authorization.community-admin.item.create-bitstream = true</code>
Informational Note:	Authorization for a delegated community administrator to create additional bitstreams in items in underlying collections.
Property:	<code>core.authorization.community-admin.item.delete-bitstream</code>
Example Value:	<code>core.authorization.community-admin.item.delete-bitstream = true</code>
Informational Note:	Authorization for a delegated community administrator to delete bitstreams from items in underlying collections.
Property:	<code>core.authorization.community-admin.item.cc-license</code>
Example Value:	<code>core.authorization.community-admin.item.cc-license = true</code>
Informational Note:	Authorization for a delegated community administrator to administer licenses from items in underlying collections.
Community Administration: The properties for collection administrators work similar to those of community administrators , with respect to collection administration.	<div style="border: 1px dashed gray; padding: 5px;"> <code>core.authorization.collection-admin.policies</code> <code>core.authorization.collection-admin.template-item</code> <code>core.authorization.collection-admin.submitters</code> <code>core.authorization.collection-admin.workflows</code> <code>core.authorization.collection-admin.admin-group</code> </div>
Collection Administration: Item owned by the above CollectionThe properties for collection administrators work similar to those of community administrators, with respect to	<div style="border: 1px dashed gray; padding: 5px;"> <code>core.authorization.collection-admin.item.delete</code> <code>core.authorization.collection-admin.item.withdraw</code> <code>core.authorization.collection-admin.item.reinstantiate</code> <code>core.authorization.collection-admin.item.policies</code> </div>

<p>administration of items in underlying collections.</p>	
<p>Collection Administration: Bundles of bitstreams, related to items owned by collections in the above Community. The properties for collection administrators work similar to those of community administrators, with respect to administration of bitstreams related to items in underlying collections.</p>	<pre>core.authorization.collection-admin.item.create-bitstream core.authorization.collection-admin.item.delete-bitstream core.authorization.collection-admin.item-admin.cc-license</pre>
<p>Item Administration. The properties for item administrators work similar to those of community and collection administrators, with respect to administration of items in underlying collections.</p>	<pre>core.authorization.item-admin.policies</pre>
<p>Item Administration: Bundles of bitstreams, related to items owned by collections in the above Community. The properties for item administrators work similar to those of community and collection administrators, with respect to administration of bitstreams related to items in underlying collections.</p>	<pre>core.authorization.item-admin.create-bitstream core.authorization.item-admin.delete-bitstream core.authorization.item-admin.cc-license</pre>

Login as feature

Property:	<code>webui.user.assumellogin</code>
Example Value:	<code>webui.user.assumellogin = true</code>
Informational Note:	<p>Determine if super administrators (those whom are in the Administrators group) can login as another user from the "edit eperson" page. This is useful for debugging problems in a running dspace instance, especially in the workflow process. The default value is false, i.e., no one may assume the login of another user.</p> <p>Please note that this configuration parameter has changed name in DSpace 4.0 from <code>xmlui.user.assumellogin</code> to <code>webui.user.assumellogin</code> as it is now supported also in the JSP UI</p>

Restricted Item Visibility Settings

By default RSS feeds and subscription emails will include ALL items regardless of permissions set on them. If you wish to only expose items through these channels where the ANONYMOUS user is granted READ permission, then set the following options to false.

Property:	<code>harvest.includerestricted.rss</code>
Example Value:	<code>harvest.includerestricted.rss = true</code>
Informational Note:	When set to 'true' (default), items that haven't got the READ permission for the ANONYMOUS user, will be included in RSS feeds anyway.
Property:	<code>harvest.includerestricted.subscription</code>
Example Value:	<code>harvest.includerestricted.subscription = true</code>
Informational Note:	When set to true (default), items that haven't got the READ permission for the ANONYMOUS user, will be included in Subscription emails anyway.

Proxy Settings

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

Property:	<code>http.proxy.host</code>
Example Value	<code>http.proxy.host = proxy.myu.edu</code>

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

Configuring Media Filters

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

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

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

Property:	<pre>filter.org.dspace.app.mediafilter.PDFFilter.inputFormats filter.org.dspace.app.mediafilter.HTMLFilter.inputFormats filter.org.dspace.app.mediafilter.WordFilter.inputFormats filter.org.dspace.app.mediafilter.JPEGFilter.inputFormats filter.org.dspace.app.mediafilter.BrandedPreviewJPEGFilter.inputFormats</pre>
Example Value:	<pre>filter.org.dspace.app.mediafilter.PDFFilter.inputFormats = Adobe PDF filter.org.dspace.app.mediafilter.HTMLFilter.inputFormats = HTML, Text filter.org.dspace.app.mediafilter.WordFilter.inputFormats = Microsoft Word filter.org.dspace.app.mediafilter.JPEGFilter.inputFormats = BMP, GIF, JPEG, \ image/png filter.org.dspace.app.mediafilter.BrandedPreviewJPEGFilter.inputFormats = BMP, \ GIF, JPEG, image/png</pre>
Informational Note:	Configure each filter's input format(s)
Property:	<code>pdffilter.largepdfs</code>
Example Value:	<code>pdffilter.largepdfs = true</code>
Informational Note:	It this value is set for "true", all PDF extractions are written to temp files as they are indexed. This is slower, but helps to ensure that PDFBox software DSpace uses does not eat up all your memory.
Property:	<code>pdffilter.skiponmemoryexception</code>
Example Value:	<code>pdffilter.skiponmemoryexception = true</code>
Informational Note:	If this value is set for "true", PDFs which still result in an "Out of Memory" error from PDFBox are skipped over. These problematic PDFs will never be indexed until memory usage can be decreased in the PDFBox software.

Names are assigned to each filter using the

`plugin.named.org.dspace.app.mediafilter.FormatFilter` field (e.g. by default the PDFFilter is named "PDF Text Extractor").

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

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

More Information on MediaFilters

For more information on Media/Format Filters, see the section on [Mediafilters for Transforming DSpace Content](#).

Crosswalk and Packager Plugin Settings

The subsections below give configuration details based on the types of crosswalks and packager plugins you need to implement.

More Information on Packers & Crosswalks

For more information on using Packers and Crosswalks, see the section on [Importing and Exporting Content via Packages](#).

Configurable MODS Dissemination Crosswalk

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

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

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

The MODS crosswalk properties file is a list of properties describing how DSpace metadata elements are to be turned into elements of the MODS XML output document. The property name is a concatenation of the metadata schema, element name, and optionally the qualifier. For example, the `contributor.author` element in the native Dublin Core schema would be: `dc.contributor.author`. The value of the property is a line containing two

segments separated by the vertical bar ("|_"): The first part is an XML fragment which is copied into the output document. The second is an XPath expression describing where in that fragment to put the value of the metadata element. For example, in this property:

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

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.

XSLT-based Crosswalks

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

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

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

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

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

1 2 3 4

crosswalk first part of the property key.

submission second part of the property key.

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

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

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

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

Example:

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

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

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

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

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

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

For example:

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



If you remove all XSLTDisseminationCrosswalks please disable the XSLTDisseminationCrosswalk in the list of selfnamed plugins. If no XSLTDisseminationCrosswalks are configured but the plugin is loaded the PluginManager will log an error message ("Self-named plugin class " `org.dspace.content.crosswalk.XSLTDisseminationCrosswalk` " returned null or empty name list!").

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 crosswalk by using a command-line utility. To test a dissemination crosswalk you have to run:

```
[dspace]/bin/dspace dsrun org.dspace.content.crosswalk.XSLTDisseminationCrosswalk <plugin name> <
handle> [output-file]
```

For example, you can test the marc plugin on the handle 123456789/3 with:

```
[dspace]/bin/dspace dsrun org.dspace.content.crosswalk.XSLTDisseminationCrosswalk marc 123456789/3
```

Informations from the script will be printed to stderr while the XML output of the dissemination crosswalk will be printed to stdout. You can give a third parameter containing a filename to write the output into a file, but be careful: the file will be overwritten if it exists.

Testing a submission crosswalk works quite the same way. Use the following command-line utility, it calls the crosswalk plugin to translate an XML document you submit, and displays the resulting intermediate XML (DIM). Invoke it with:

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

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

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

Configurable Qualified Dublin Core (QDC) dissemination crosswalk

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

The following is from *dspace.cfg* file:

Properties:	<code>crosswalk.qdc.namespace.qdc.dc</code>
	<code>crosswalk.qdc.namespace.qdc.dc = http://purl.org/dc/elements/1.1_</code>

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

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

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

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

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

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

For example, in this property:

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

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

```
<dcterms:temporal>Fall, 2005</dcterms:temporal>
```

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.

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.

Event System Configuration

If you are unfamiliar with the Event System in DSpace, and require additional information with terms like "Consumer" and "Dispatcher" please refer to: <http://wiki.dspace.org/index.php/EventSystemPrototype>

Property:	<code>event.dispatcher.default.class</code>
Example Value:	<code>event.dispatcher.default.class = org.dspace.event.BasicDispatcher</code>
Informational Note:	This is the default synchronous dispatcher (Same behavior as traditional DSpace).
Property:	<code>event.dispatcher.default.consumers</code>
Example Value:	<code>event.dispatcher.default.consumers = search, browse, eperson</code>
Informational Note:	This is the default synchronous dispatcher (Same behavior as traditional DSpace).
Property:	<code>event.dispatcher.noindex.class</code>
	<code>event.dispatcher.noindex.class = org.dspace.event.BasicDispatcher</code>

Example Value:	
Informational Note:	The noindex dispatcher will not create search or browse indexes (useful for batch item imports) .
Property:	<code>event.dispatcher.noindex.consumers</code>
Example Value:	<code>event.dispatcher.noindex.consumers = eperson</code>
Informational Note:	The noindex dispatcher will not create search or browse indexes (useful for batch item imports) .
Property:	<code>event.consumer.search.class</code>
Example Value:	<code>event.consumer.search.class = org.dspace.search.SearchConsumer</code>
Informational Note:	Consumer to maintain the search index.
Property:	<code>event.consumer.search.filters</code>
Example Value:	<code>{{event.consumer.search.filters = }} Community Collection Item Bundle+Add Create Modify Modify_Metadata Delete Remove</code>
Informational Note:	Consumer to maintain the search index.
Property:	<code>event.consumer.browse.class</code>
Example Value:	<code>event.consumer.browse.class = org.dspace.browse.BrowseConsumer</code>
Informational Note:	Consumer to maintain the browse index.
Property:	<code>event.consumer.browse.filters</code>
Example Value:	<code>event.consumer.browse.filters = Community Collection Item Bundle+Add Create Modify Modify_Metadata Delete Remove</code>
Informational Note:	Consumer to maintain the browse index.

Property:	<code>event.consumer.eperson.class</code>
Example Value:	<code>event.consumer.eperson.class = org.dspace.eperson.EPersonConsumer</code>
Informational Note:	Consumer related to EPerson changes
Property:	<code>event.consumer.eperson.filters</code>
Example Value:	<code>event.consumer.eperson.filters = EPerson+Create</code>
Informational Note:	Consumer related to EPerson changes
Property:	<code>event.consumer.test.class</code>
Example Value:	<code>event.consumer.test.class = org.dspace.event.TestConsumer</code>
Informational Note:	Test consumer for debugging and monitoring. Commented out by default.
Property:	<code>event.consumer.test.filters</code>
Example Value:	<code>event.consumer.test.filters = All+All</code>
Informational Note:	Test consumer for debugging and monitoring. Commented out by default.
Property:	<code>testConsumer.verbose</code>
Example Value:	<code>testConsumer.verbose = true</code>
Informational Note:	Set this to true to enable testConsumer messages to standard output. Commented out by default.

Embargo

DSpace embargoes utilize standard metadata fields to hold both the "terms" and the "lift date". Which fields you use are configurable, and no specific metadata element is dedicated or predefined for use in embargo. Rather, you specify exactly what field you want the embargo system to examine when it needs to find the terms or assign the lift date.

Property:	<code>embargo.field.terms</code>
Example Value:	<code>embargo.field.terms = SCHEMA.ELEMENT.QUALIFIER</code>
Informational Note:	Embargo terms will be stored in the item metadata. This property determines in which metadata field these terms will be stored. An example could be <code>dc.embargo.terms</code>
Property:	<code>embargo.field.lift</code>
Example Value:	<code>embargo.field.lift = SCHEMA.ELEMENT.QUALIFIER</code>
Informational Note:	The Embargo lift date will be stored in the item metadata. This property determines in which metadata field the computed embargo lift date will be stored. You may need to create a DC metadata field in your Metadata Format Registry if it does not already exist. An example could be <code>dc.embargo.liftdate</code>
Property:	<code>embargo.terms.open</code>
Example Value:	<code>embargo.terms.open = forever</code>
Informational Note:	You can determine your own values for the <code>embargo.field.terms</code> property (see above). This property determines what the string value will be for indefinite embargos. The string in terms field to indicate indefinite embargo.
Property:	<code>plugin.single.org.dspace.embargo.EmbargoSetter</code>
Example Value:	<code>plugin.single.org.dspace.embargo.EmbargoSetter = org.dspace.embargo.DefaultEmbargoSetter</code>
Informational Note:	To implement the business logic to set your embargos, you need to override the <code>EmbargoSetter</code> class. If you use the value <code>DefaultEmbargoSetter</code> , the default implementation will be used.
Property:	<code>plugin.single.org.dspace.embargo.EmbargoLifter</code>
Example Value:	<code>plugin.single.org.dspace.embargo.EmbargoLifter = org.dspace.embargo.DefaultEmbargoLifter</code>
Informational Note:	To implement the business logic to lift your embargos, you need to override the <code>EmbargoLifter</code> class. If you use the value <code>DefaultEmbargoLifter</code> , the default implementation will be used.

 **More Embargo Details**

More details on Embargo configuration, including specific examples can be found in the [Embargo](#) section of the documentation.

Checksum Checker Settings

DSpace now comes with a Checksum Checker script (`[dspace]/bin/dspace_checker`) which can be scheduled to verify the checksum of every item within DSpace. Since DSpace calculates and records the checksum of every file submitted to it, this script is able to determine whether or not a file has been changed (either manually or by some sort of corruption or virus). The idea being that the earlier you can identify a file has changed, the more likely you'd be able to recover it (assuming it was not a wanted change).

Property:	<code>plugin.single.org.dspace.checker.BitstreamDispatcher</code>
Example Value:	<code>plugin.single.org.dspace.checker.BitstreamDispatcher = org.dspace.checker.SimpleDispatcher</code>
Informational Note:	The Default dispatcher is case non is specified.
Property:	<code>checker.retention.default</code>
Example Value:	<code>checker.retention.default = 10y</code>
Informational Note:	This option specifies the default time frame after which all checksum checks are removed from the database (defaults to 10 years). This means that after 10 years, all successful or unsuccessful matches are removed from the database.
Property:	<code>checker.retention.CHECKSUM_MATCH</code>
Example Value:	<code>checker.retention.CHECKSUM_MATCH = 8w</code>
Informational Note:	This option specifies the time frame after which a successful match will be removed from your DSpace database (defaults to 8 weeks). This means that after 8 weeks, all successful matches are automatically deleted from your database (in order to keep that database table from growing too large).

More Checksum Checking Details

For more information on using DSpace's built-in Checksum verification system, see the section on [Validating CheckSums of Bitstreams](#).

Item Export and Download Settings

It is possible for an authorized user to request a complete export and download of a DSpace item in a compressed zip file. This zip file may contain the following:

dublin_core.xml

license.txt

contents (*listing of the contents*)

handle *file itself and the extract file if available*

The configuration settings control several aspects of this feature:

Property:	<code>org.dspace.app.itemexport.work.dir</code>
Example Value:	<code>org.dspace.app.itemexport.work.dir = \${dspace.dir}/exports</code>
Informational Note:	The directory where the exports will be done and compressed.
Property:	<code>org.dspace.app.itemexport.download.dir</code>
Example Value:	<code>org.dspace.app.itemexport.download.dir = \${dspace.dir}/exports/download</code>
Informational Note:	The directory where the compressed files will reside and be read by the downloader.
Property:	<code>org.dspace.app.itemexport.life.span.hours</code>
Example Value:	<code>org.dspace.app.itemexport.life.span.hours = 48</code>
Informational Note:	The length of time in hours each archive should live for. When new archives are created this entry is used to delete old ones.
Property:	<code>org.dspace.app.itemexport.max.size</code>
Example Value:	<code>org.dspace.app.itemexport.max.size = 200</code>
Informational Note:	The maximum size in Megabytes (Mb) that the export should be. This is enforced before the compression. Each bitstream's size in each item being exported is added up, if their cumulative sizes are more than this entry the export is not kicked off.

Subscription Emails

DSpace, through some advanced installation and setup, is able to send out an email to collections that a user has subscribed. The user who is subscribed to a collection is emailed each time an item id added or modified. The following property key controls whether or not a user should be notified of a modification.

Property:	<code>eperson.subscription.onlynew</code>
Example Value:	<code>eperson.subscription.onlynew = true</code>
Informational Note:	For backwards compatibility, the subscription emails by default include any modified items. The property key is COMMENTED OUT by default.

Hiding Metadata

It is now possible to hide metadata from public consumption that is only available to the Administrator.

Property:	<code>metadata.hide.dc.description.provenance</code>
Example Value:	<code>metadata.hide.dc.description.provenance = true</code>
Informational Note:	<p>Hides the metadata in the property key above except to the administrator. Fields named here are hidden in the following places UNLESS the logged-in user is an Administrator:</p> <ol style="list-style-type: none"> 1. XMLUI metadata XML view, and Item splash pages (long and short views). 2. JSPUI Item splash pages 3. OAI-PMH server, "oai_dc" format. (Note: Other formats are *not* affected.)To designate a field as hidden, add a property here in the form: <code>metadata.hide.SCHEMA.ELEMENT.QUALIFIER = true</code>. This default configuration hides the <code>dc.description.provenance</code> field, since that usually contains email addresses which ought to be kept private and is mainly of interest to administrators.

Settings for the Submission Process

These settings control three aspects of the submission process: thesis submission permission, whether or not a bitstream file is required when submitting to a collection and whether to show a progress bar during the file upload.

Property:	<code>webui.submit.blocktheses</code>
-----------	---------------------------------------

Example Value:	<code>webui.submit.blocktheses = false</code>
Informational Note:	Controls whether or not the UI blocks a submission which is marked as a thesis.
Property:	<code>webui.submit.upload.required</code>
Example Value:	<code>webui.submit.upload.required = true</code>
Informational Note:	Whether or not a file is required to be uploaded during the "Upload" step in the submission process. The default is true. If set to "false", then the submitter (human being) has the option to skip the uploading of a file.
Property:	<code>webui.submit.upload.html5</code>
Example Value:	<code>webui.submit.upload.html5 = true</code>
Informational Note:	If the browser supports it, JSPUI uses html5 File API to enhance file upload. If this property is set to false the enhanced file upload is not used even if the browser would support it.
Property:	<code>webui.submit.upload.progressbar</code> (new in DSpace 4.0)
Example Value:	<code>webui.submit.upload.progressbar = true</code>
Informational Note:	<p>Whether to show a progress bar during file upload. Please note that to work this feature requires a JSON endpoint (<code>json/uploadProgress</code>) that is enabled by default. See the named plugin for the interface <code>org.dspace.app.webui.json.JSONRequest</code></p> <pre>org.dspace.app.webui.json.UploadProgressJSON = uploadProgress</pre> <p>This property is actually supported only by the JSPUI. The XMLUI doesn't yet provide a progress bar indicator for file upload.</p>

Configuring the Sherpa/RoMEO Publishers Policy Database Integration

DSpace 4.0 introduced integration with the Sherpa/RoMEO Publishers Policy Database in order to allow displaying the publisher policy in the submission upload step. The submission step interface is available in JSPUI (since DSpace 4.0) and in XMLUI (since DSpace 5.0) and enabled by default, however to use it in production (over 500 requests per day), you must register for a free API key (see below for details).

Property:	<code>webui.submission.sherparomeo-policy-enabled</code>
Example Value:	<code>webui.submission.sherparomeo-policy-enabled = true</code>
Informational Note:	Controls whether or not the UI submission should try to use the Sherpa/RoMEO Publishers Policy Database Integration (default true)
Property:	<code>sherpa.romeo.url</code>
	<code>sherpa.romeo.url = http://www.sherpa.ac.uk/romeo/api29.php</code>

Example Value:	
Informational Note:	The Sherpa/RoMEO endpoint. Shared with the authority control feature for Journal Title autocomplete see AuthorityControlSettings
Property:	sherpa.romeo.apikey
Example Value:	sherpa.romeo.apikey = YOUR-API-KEY
Informational Note:	Allow to use a specific API key to raise the usage limit (500 calls/day for unregistered user). You can register for a free api access key at http://www.sherpa.ac.uk/news/romeoapikey.htm

The functionality rely on understanding to which Journal (ISSN) is related the submitting item. This is done out of box looking to some item metadata but a different strategy can be used as for example look to a metadata authority in the case that the Sherpa/RoMEO autocomplete for Journal is used (see [AuthorityControlSettings](#))

The strategy used to discover the Journal related to the submission item is defined in the spring file **/config/spring/api/sherpa.xml**

```
<bean class="org.dspace.app.sherpa.submit.SHERPASubmitConfigurationService"
      id="org.dspace.app.sherpa.submit.SHERPASubmitConfigurationService">
  <property name="issnItemExtractors">
    <list>
      <bean class="org.dspace.app.sherpa.submit.MetadataValueISSNExtractor">
        <property name="metadataList">
          <list>
            <value>dc.identifier.issn</value>
          </list>
        </property>
      </bean>
      <!-- Use the follow if you have the SHERPARoMEOJournalTitle enabled
      <bean class="org.dspace.app.sherpa.submit.MetadataAuthorityISSNExtractor">
        <property name="metadataList">
          <list>
            <value>dc.title.alternative</value>
          </list>
        </property>
      </bean> -->
    </list>
  </property>
</bean>
```

Configuring Creative Commons License

The following configurations are for the Creative Commons license step in the submission process. 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 further details, refer to the Creative Commons website at <http://creativecommons.org>.

Creative Commons licensing is optionally available and may be configured for any given collection that has a defined submission sequence, or be part of the "default" submission process. This process is described in the [Submission User Interface](#) section of this manual. There is a Creative Commons step already defined (step 5), but it is commented out, so enabling Creative Commons licensing is typically just a matter of uncommenting the CC License step.

Creative Commons licensing is captured slightly differently in each UI:

- In the JSPUI, an "iframe" is opened to the Creative Commons site. When a Creative Commons license is selected from that site, information about the CC license is stored in a series of internal bitstreams:
 - The URL of the CC License is stored in a bitstream named "license_url" in the CC-LICENSE bundle
 - The full (HTML) text of the CC License is stored in a bitstream named "license_txt" in the CC-LICENSE bundle
 - The RDF version of the CC License is stored in a bitstream named "license_rdf" in the CC-LICENSE bundle
- In the XMLUI, the Create Commons REST API is utilized. This allows the XMLUI to store metadata references to the selected CC license, while also storing the CC License as a bitstream. In the XMLUI, the following CC License information is captured:
 - The URL of the CC License is stored in the "dc.rights.uri" metadata field (or whatever field is configured in the "cc.license.uri" setting below)
 - The name of the CC License is stored in the "dc.rights" metadata field (or whatever field is configured in the "cc.license.name" setting below). This only occurs if "cc.submit.setname=true" (default value)
 - The RDF version of the CC License is stored in a bitstream named "license_rdf" in the CC-LICENSE bundle (as long as "cc.submit.addbitstream=true", which is the default value)

The following configurations (in `dspace.cfg`) relate to the XMLUI Creative Commons license process ONLY:

Property:	<code>cc.api.rooturl</code>
Example Value:	<code>cc.api.rooturl = http://api.creativecommons.org/rest/1.5</code>

Informational Note:	Generally will never have to assign a different value - this is the base URL of the Creative Commons service API.
Property:	<code>cc.license.uri</code>
Example Value:	<code>cc.license.uri = dc.rights.uri</code>
Informational Note:	The field that holds the Creative Commons license URI. If you change from the default value (<code>dc.rights.uri</code>), you will have to reconfigure the XMLUI for proper display of license data
Property:	<code>cc.license.name</code>
Example Value:	<code>cc.license.name = dc.rights</code>
Informational Note:	The field that holds the Creative Commons license Name. If you change from the default value (<code>dc.rights</code>), you will have to reconfigure the XMLUI for proper display of license data
Property:	<code>cc.submit.setname</code>
Example Value:	<code>cc.submit.setname = true</code>
Informational Note:	If true, the license assignment will add the field configured with the "cc.license.name" with the name of the CC license; if false, only "cc.license.uri" field is added.
Property:	<code>cc.submit.addbitstream</code>
Example Value:	<code>cc.submit.addbitstream = true</code>
Informational Note:	If true, the license assignment will add a bitstream with the CC license RDF; if false, only metadata field(s) are added.
Property:	<code>cc.license.classfilter</code>
Example Value:	<code>cc.license.classfilter = recombo,mark</code>
Informational Note:	This list defines the values that will be excluded from the license (class) selection list, as defined by the web service at the URL: http://api.creativecommons.org/rest/1.5/classes
Property:	<code>cc.license.jurisdiction</code>

Example Value:	<code>cc.license.jurisdiction = nz</code>
Informational Note:	Should a jurisdiction be used? If so, which one? See http://creativecommons.org/international/ for a list of possible codes (e.g. nz = New Zealand, uk = England and Wales, jp = Japan)

WEB User Interface Configurations

General Web User Interface Configurations

In this section of Configuration, we address the agnostic WEB User Interface that is used for JSPUI and XMLUI. Some of the configurations will give information towards customization or refer you to the appropriate documentation.

Property:	<code>webui.licence_bundle.show</code>
Example Value:	<code>webui.licence_bundle.show = false</code>
Informational Note:	Sets whether to display the contents of the license bundle (often just the deposit license in the standard DSpace installation).
Property:	<code>webui.browse.thumbnail.show</code>
Example Value:	<code>webui.browse.thumbnail.show = true</code>
Informational Note:	Controls whether to display thumbnails on browse and search result pages. If you have customized the Browse columnlist, then you must also include a "thumbnail" column in your configuration. _(This configuration property key is not used by XMLUI. To show thumbnails using XMLUI, you need to create a theme which displays them)._
Property:	<code>webui.browse.thumbnail.maxheight</code>
Example Value:	<code>webui.browse.thumbnail.maxheight = 80</code>
Informational Note:	This property determines the maximum height of the browse/search thumbnails in pixels (px). This only needs to be set if the thumbnails are required to be smaller than the dimensions of thumbnails generated by MediaFilter.
Property:	<code>webui.browse.thumbnail.maxwidth</code>
Example Value:	<code>webui.browse.thumbnail.maxwidth = 80</code>
Informational Note:	

	This determines the maximum width of the browse/search thumbnails in pixels (px). This only needs to be set if the thumbnails are required to be smaller than the dimensions of thumbnails generated by MediaFilter.
Property:	<code>webui.item.thumbnail.show</code>
Example Value:	<code>webui.item.thumbnail.show = true</code>
Informational Note:	This determines whether or not to display the thumbnail against each bitstream. (<i>This configuration property key is not used by XMLUI. To show thumbnails using XMLUI, you need to create a theme which displays them.</i>)
Property:	<code>webui.browse.thumbnail.linkbehavior</code>
Example Value:	<code>webui.browse.thumbnail.linkbehavior = item</code>
Informational Note:	This determines where clicks on the thumbnail in browse and search screens should lead. The only values currently supported are "item" or "bitstream", which will either take the user to the item page, or directly download the bitstream.
Property:	<code>thumbnail.maxwidth</code>
Example Value:	<code>thumbnail.maxwidth = 80</code>
Informational Note:	This property sets the maximum width of generated thumbnails that are being displayed on item pages.
Property:	<code>thumbnail.maxheight</code>
Example Value:	<code>thumbnail.maxheight = 80</code>
Informational Note:	This property sets the maximum height of generated thumbnails that are being displayed on item pages.
Property:	<code>webui.preview.enabled</code>
Example Value:	<code>webui.preview.enabled = false</code>
Informational Note:	Whether or not the user can "preview" the image.
Property:	<code>webui.preview.maxwidth</code>

Example Value:	<code>webui.preview.maxwidth = 600</code>
Informational Note:	This property sets the maximum width for the preview image.
Property:	<code>webui.preview.maxheight</code>
Example Value:	<code>webui.preview.maxheight = 600</code>
Informational Note:	This property sets the maximum height for the preview image.
Property:	<code>webui.preview.brand</code>
Example Value:	<code>webui.preview.brand = My Institution Name</code>
Informational Note:	This is the brand text that will appear with the image.
Property:	<code>webui.preview.brand.abbrev</code>
Example Value:	<code>webui.preview.brand.abbrev = MyOrg</code>
Informational Note:	An abbreviated form of the full Branded Name. This will be used when the preview image cannot fit the normal text.
Property:	<code>webui.preview.brand.height</code>
Example Value:	<code>webui.preview.brand.height = 20</code>
Informational Note:	The height (in px) of the brand.
Property:	<code>webui.preview.brand.font</code>
Example Value:	<code>webui.preview.brand.font = SansSerif</code>
Informational Note:	This property sets the font for your Brand text that appears with the image.
Property:	<code>webui.preview.brand.fontpoint</code>
	<code>webui.preview.brand.fontpoint = 12</code>

Example Value:	
Informational Note:	This property sets the font point (size) for your Brand text that appears with the image.
Property:	<code>webui.preview.dc</code>
Example Value:	<code>webui.preview.dc = rights</code>
Informational Note:	The Dublin Core field that will display along with the preview. This field is optional.
Property:	<code>webui.strengths.show</code>
Example Value:	<code>webui.strengths.show = false</code>
Informational Note:	Determines if communities and collections should display item counts when listed. The default behavior if omitted, is true. <i>(This configuration property key is not used by XMLUI. To show strengths using XMLUI, you need to create a theme which displays them).</i>
Property:	<code>webui.strengths.cache</code>
Example Value:	<code>webui.strengths.cache = false</code>
Informational Note:	When showing the strengths, should they be counted in real time, or fetched from the cache. Counts fetched in real time will perform an actual count of the database contents every time a page with this feature is requested, which will not scale. If you set the property key is set to cache ("true") you must run the following command periodically to update the count: <code>/[dspace]/bin/dspace itemcounter</code> . The default is to count in real time (set to "false").


Browse Index Configuration

The browse indexes for DSpace can be extensively configured. This section of the configuration allows you to take control of the indexes you wish to browse, and how you wish to present the results. The configuration is broken into several parts: defining the indexes, defining the fields upon which users can sort results, defining truncation for potentially long fields (e.g. authors), setting cross-links between different browse contexts (e.g. from an author's name to a complete list of their items), how many recent submissions to display, and configuration for item mapping browse.

Property:	<code>webui.browse.index.<n></code>
	<code>{{webui.browse.index.1 = dateissued:metadata:dc.date.issued:date:full }}</code>

Example Value:	
Informational Note:	This is an example of how one "Defines the Indexes". See Defining the Indexes in the next sub-section.
Property:	<code>webui.itemlist.sort-option.<n></code>
Example Value:	<code>webui.itemlist.sort-option.1 = title:dc.title:title</code>
Informational Note:	This is an example of how one "Defines the Sort Options". See Defining Sort Options in the following sub-section.

Defining the storage of the Browse Data

 Starting from DSpace 3.0 you can configure which implementation use for the Browse DAOs both for create/update operations and for read operations. This allows you to customize which browse engine is utilized in your DSpace. Options include:


- SOLR Browse Engine (SOLR DAOs), **default since DSpace 4.0** - This enables Apache Solr to be utilized as a backend for all browsing of DSpace. This option requires that you have [Discovery](#) (Solr search/browse engine) enabled in your DSpace.
- PostgreSQL Browse Engine (PostgreSQL DAOs) - This enables all browsing to be done via PostgreSQL database tables. (This is the traditional browsing option for users who have PostgreSQL installed.)
- Oracle Browse Engine (Oracle DAOs) - This enables all browsing to be done via Oracle database tables. (This is the traditional browsing option for users who have Oracles installed.)

Property:	<code>browseDAO.class</code>
Example Value:	<code>browseDAO.class = org.dspace.browse.SolrBrowseDAO</code>
Informational Note:	This property configures the Java class that is used for READ operations by the Browse System. You need to have Discovery enabled (this is the default since DSpace 4.0) to use the Solr Browse DAOs
Property:	<code>browseCreateDAO.class</code>
Example Value:	<code>browseCreateDAO.class = org.dspace.browse.SolrBrowseCreateDAO</code>

Informational Note:	This property configures the java class that is used for WRITE operations by the Browse System. You need to have Discovery enabled (this is the default since DSpace 4.0) to use the Solr Browse DAOs
---------------------	--

If you want to re-enable the legacy DBMS Browse Engine please refer to [Legacy methods for re-indexing content](#)

Defining the Indexes

 If you make changes in this section be sure to update your SOLR indexes running the Discovery Maintenance Script, see [Discovery](#)

DSpace comes with four default indexes pre-defined: author, title, date issued, and subjects. Users may also define additional indexes or re-configure the current indexes for different levels of specificity. For example, the default entries that appear in the *dspace.cfg* as default installation:

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

The format of each entry is `webui.browse.index.<n> = <index name>:<metadata>:<schema prefix>.<element>.<qualifier>:<data-type field>:<sort option>`. Please notice that the punctuation is paramount in typing this property key in the *dspace.cfg* file. The following table explains each element:

Element	Definition and Options (if available)
<code>webui.browse.index . < n ></code>	<i>n</i> is the index number. The index numbers must start from 1 and increment continuously by 1 thereafter. Deviation from this will cause an error during install or a configuration update. So anytime you add a new browse index, remember to increase the number. (Commented out index numbers may be used over again).
<code><index name></code>	The name by which the index will be identified. You will need to update your Messages.properties file to match this field. (The form used in the Messages.properties file is: <code>browse.type.metadata.<index name> .</code>
<code><metadata></code>	Only two options are available: "metadata" or "item"
<code><schema prefix></code>	The schema used for the field to be index. The default is dc (for Dublin Core).

Element	Definition and Options (if available)
<element>	The schema element. In Dublin Core, for example, the author element is referred to as "Contributor". The user should consult the default Dublin Core Metadata Registry table in Appendix A.
<qualifier>	This is the qualifier to the <element> component. The user has two choices: an asterisk "*" or a proper qualifier of the element. The asterisk is a wildcard and causes DSpace to index all types of the schema element. For example, if you have the element "contributor" and the qualifier "*" then you would index all contributor data regardless of the qualifier. Another example, you have the element "subject" and the qualifier "lcsh" would cause the indexing of only those fields that have the qualifier "lcsh". (This means you would only index Library of Congress Subject Headings and not all data elements that are subjects.
<datatype field>	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
<index display>	Choose full or 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 <code>webui.itemlist.columns</code> ; "single" will be a single list of only the indexed term.

If you are customizing this list beyond the default, you will need to insert the text you wish to appear in the navigation and on link and buttons. You need to edit the `Messages.properties` file. The form of the parameter(s) in the file:

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

Defining Sort Options



If you make changes in this section be sure to update your SOLR indexes running the Discovery Maintenance Script, see [Discovery](#)

Sort options will be available when browsing a list of items (i.e. only in "full" mode, not "single" mode). You can define an arbitrary number of fields to sort on, irrespective of which fields you display using `web.itemlist.columns`. For example, the default entries that appear in the `dspace.cfg` as default installation:


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

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

The format of each entry is `web.browse.sort-option.<n> = <option name>:<schema prefix>.<element>.<qualifier>:<datatype>`. Please notice the punctuation used between the different elements. The following table explains the each element:

Element	Definition and Options (if available)
<code>webui.browse.index . n</code>	<i>n</i> is an arbitrary number you choose.
<code><option name></code>	The name by which the sort option will be identified. This may be used in later configuration or to locate the message key (found in <i>Messages.properties</i> file) for this index.
<code><schema prefix></code>	The schema used for the field to be index. The default is dc (for Dublin Core).
<code><element></code>	The schema element. In Dublin Core, for example, the author element is referred to as "Contributor". The user should consult the default Dublin Core Metadata Registry table in Appendix A.
<code><qualifier></code>	This is the qualifier to the <code><element></code> component. The user has two choices: an asterisk "*" or a proper qualifier of the element.
<code><datatype field></code>	This refers to the datatype of the field: date the sort type will be treated as a date object text the sort type will be treated as plain text.

Browse Index Normalization Rule Configuration

 If you make changes in this section be sure to update your SOLR indexes running the Discovery Maintenance Script, see [Discovery](#)

Normalization Rules are those rules that make it possible for the indexes to intermix entries without regard to case sensitivity. By default, the display of metadata in the browse indexes are case-sensitive. In the example below, you retrieve separate entries:

Twain, Marktwain, markTWAIN, MARK

However, clicking through from either of these will result in the same set of items (i.e., any item that contains either representation in the correct field).

Property:	<code>webui.browse.metadata.case-insensitive</code>
	<code>webui.browse.metadata.case-insensitive = true</code>

Example Value:	
Informational Note:	This controls the normalization of the index entry. Uncommenting the option (which is commented out by default) will make the metadata items case-insensitive. This will result in a single entry in the example above. However, the value displayed may be any one of the above, depending on what representation was present in the first item indexed.

At the present time, you would need to edit your metadata to clean up the index presentation.

Other Browse Options

We set other browse values in the following section.

Property:	<code>webui.browse.metadata.show-freq.< n ></code>
Example Value:	<code>webui.browse.metadata.show-freq.1 = false</code>
Informational Note:	This enable/disable the show of frequencies (count) in metadata browse < n > refers to the browse configuration. As default frequencies are shown for all metadata browse
Property:	<code>webui.browse.value_columns.max</code>
Example Value:	<code>webui.browse.value_columns.max = 500</code>
Informational Note:	This sets the options for the size (number of characters) of the fields stored in the database. The default is 0, which is unlimited size for fields holding indexed data. Some database implementations (e.g. Oracle) will enforce their own limit on this field size. Reducing the field size will decrease the potential size of your database and increase the speed of the browse, but it will also increase the chance of mis-ordering of similar fields. The values are commented out, but proposed values for reasonably performance versus result quality. This affects the size of field for the browse value (this will affect display, and value sorting)
Property:	<code>webui.browse.sort_columns.max</code>
Example Value:	<code>webui.browse.sort_columns.max = 200</code>
Informational Note:	Size of field for hidden sort columns (this will affect only sorting, not display). Commented out as default.
Property:	<code>webui.browse.value_columns.omission_mark</code>
Example Value:	<code>webui.browse.value_columns.omission_mark = ...</code>

Informational Note:	Omission mark to be placed after truncated strings in display. The default is "...".
Property:	<code>plugin.named.org.dspace.sort.OrderFormatDelegate</code>
Example Value:	<pre>plugin.named.org.dspace.sort.OrderFormatDelegate = \ org.dspace.sort.OrderFormatTitleMarc21=title</pre>
Informational Note:	<p>This sets the option for how the indexes are sorted. All sort normalizations are carried out by the <code>OrderFormatDelegate</code>. The plugin manager can be used to specify your own delegates for each datatype. The default datatypes (and delegates) are:</p> <pre>author = org.dspace.sort.OrderFormatAuthor title = org.dspace.sort.OrderFormatTitle text = org.dspace.sort.OrderFormatText</pre> <p>If you redefine a default datatype here, the configuration will be used in preferences to the default. However, if you do not explicitly redefine a datatype, then the default will still be used in addition to the datatypes you do specify. As of DSpace release 1.5.2, the multi-lingual MARC21 title ordering is configured as default, as shown in the example above. To use the previous title ordering (before release 1.5.2), comment out the configuration in your <code>dspace.cfg</code> file.</p>

Browse Index Authority Control Configuration

Property:	<code>webui.browse.index.<n></code>
Example Value :	<code>webui.browse.index.5 = lcAuthor:metadataAuthority: dc.contributor.author:authority</code>
Informational Note:	

Tag cloud

Apart from the single (type=metadata) and full (type=item) browse pages, tag cloud is a new way to display the unique values of a metadata field.

To enable "tag cloud" browsing for a specific index you need to declare it in the `dspace.cfg` configuration file using the following option:

Property:	<code>webui.browse.index.tagcloud.<n></code>
-----------	--

Example Value:	webui.browse.index.tagcloud.1 = true
Informational Note:	<p>Enable/Disable tag cloud in browsing for a specific index. 'n' is the index number of the specific index which needs to be of type 'metadata'.</p> <p>Possible values: true, false</p> <p>Default value is false.</p> <p>If no option exists for a specific index, it is assumed to be false.</p> <p>You do not have to re-index discovery when you change this configuration</p>

Tag cloud configuration

The appearance configuration for the tag cloud is located in the Discovery xml configuration file (*dspace/config/spring/api/discovery.xml*). Without configuring the appearance, the default one will be applied to the tag cloud

In this file, there must be a bean named "*browseTagCloudConfiguration*" of class "*org.dspace.discovery.configuration.TagCloudConfiguration*". This bean can have any of the following properties. If some is missing, the default value will be applied.

displayScore	Should display the score of each tag next to it? Default: false
shouldCenter	Should display the tag as center aligned in the page or left aligned? Possible values: true false. Default: true
totalTags	How many tags will be shown. Value -1 means all of them. Default: -1
cloudCase	<p>The letter case of the tags.</p> <p>Possible values: Case.LOWER Case.UPPER Case.CAPITALIZATION Case.PRESERVE_CASE Case.CASE_SENSITIVE</p> <p>Default: Case.PRESERVE_CASE</p>
randomColors	If the 3 css classes of the tag cloud should be independent of score (random=yes) or based on the score. Possible values: true false . Default: true
fontFrom	The font size (in em) for the tag with the lowest score. Possible values: any decimal. Default: 1.1
fontTo	The font size (in em) for the tag with the lowest score. Possible values: any decimal. Default: 3.2
cuttingLevel	The score that tags with lower than that will not appear in the rag cloud. Possible values: any integer from 1 to infinity. Default: 0

ordering	<p>The ordering of the tags (based either on the name or the score of the tag)</p> <p>Possible values: Tag.NameComparatorAsc Tag.NameComparatorDesc Tag.ScoreComparatorAsc Tag.ScoreComparatorDesc</p> <p>Default: Tag.GreekNameComparatorAsc</p>
-----------------	---

When tagCloud is rendered there are some CSS classes that you can change in order to change the tagcloud appearance.

Class	Note
tagcloud	General class for the whole tagcloud
tagcloud_1	Specific tag class for tag of type 1 (baed on score)
tagcloud_2	Specific tag class for tag of type 2 (baed on score)
tagcloud_3	Specific tag class for tag of type 3 (baed on score)

Author (Multiple metadata value) Display

This section actually applies to any field with multiple values, but authors are the define case and example here.

Property:	<code>webui.browse.author-field</code>
Example Value:	<code>webui.browse.author-field = dc.contributor.*</code>
Informational Note:	This defines which field is the author/editor, etc. listing.

Replace `dc.contributor.*` with another field if appropriate. The field should be listed in the configuration for `webui.itemlist.columns`, otherwise you will not see its effect. It must also be defined in `webui.itemlist.columns` as being of the datatype `text` otherwise the functionality will be overridden by the specific data type feature. (This setting is not used by the XMLUI as it is controlled by your theme).

Now that we know which field is our author or other multiple metadata value field we can provide the option to truncate the number of values displayed by default. We replace the remaining list of values with "et al" or the language pack specific alternative. Note that this is just for the default, and users will have the option of changing the number displayed when they browse the results. See the following table:

Property:	<code>webui.browse.author-limit</code>
Example Value:	<code>webui.browse.author-limit = < n ></code>
:	

Informational Note:	Where <code><n></code> is an integer number of values to be displayed. Use <code>-1</code> for unlimited (the default value).
---------------------	---

Links to Other Browse Contexts

We can define which fields link to other browse listings. This is useful, for example, to link an author's name to a list of just that author's items. The effect this has is to create links to browse views for the item clicked on. If it is a "single" type, it will link to a view of all the items which share that metadata element in common (i.e. all the papers by a single author). If it is a "full" type, it will link to a view of the standard full browse page, starting with the value of the link clicked on.

Property:	<code>webui.browse.link.<n></code>
Example Value:	<code>webui.browse.link.1 = author:dc.contributor.*</code>
Informational Note:	This is used to configure which fields should link to other browse listings. This should be associated with the name of one of the browse indexes (<code>webui.browse.index.n</code>) with a metadata field listed in <code>webui.itemlist.columns</code> above. If this condition is not fulfilled, cross-linking will not work. Note also that crosslinking only works for metadata fields not tagged as <code>title</code> in <code>webui.itemlist.columns</code> .

The format of the property key is `webui.browse.link.<n> = <index name>:<display column metadata>` Please notice the punctuation used between the elements.

Element	Definition and Options (if available)
<code>webui.browse.link.n</code>	{{ <i>n</i> is an arbitrary number you choose
<code><index name></code>	This need to match your entry for the index name from <code>webui.browse.index</code> property key.
<code><display column metadata></code>	Use the DC element (and qualifier)

Examples of some browse links used in a real DSpace installation instance:

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

Creates a link for all types of contributors (authors, editors, illustrators, others, etc.)


```
webui.browse.link.2 = subject:dc.subject.lcsh
```

Creates a link to subjects that are Library of Congress only. In this case, you have a browse index that contains only LC Subject Headings

```
webui.browse.link.3 = series:dc.relation.ispartofseries
```

Creates a link for the browse index "Series". Please note this is again, a customized browse index and not part of the DSpace distributed release.

Recent Submissions

 Since DSpace 4.0 this will apply by default only to JSPUI. XML UI will use a new way to configure the recent submissions that does not rely on the Browse System. See [Discovery](#)

This allows us to define which index to base Recent Submission display on, and how many we should show at any one time. This uses the PluginManager to automatically load the relevant plugin for the Community and Collection home pages. Values given in examples are the defaults supplied in *dspace.cfg*

Property:	<code>recent.submission.sort-option</code>
Example Value:	<code>recent.submission.sort-option = dateaccessioned</code>
Informational Note:	Define the sort name (from <i>webui.browse.sort-options</i>) to use for displaying recent submissions.
Property:	<code>recent.submissions.count</code>
Example Value:	<code>recent.submissions.count = 5</code>
Informational Note:	Defines how many recent submissions should be displayed at any one time.

There will be the need to set up the processors that the PluginManager will load to actually perform the recent submissions query on the relevant pages. This is already configured by default *dspace.cfg* so there should be no need for the administrator/programmer to worry about this.

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

Submission License Substitution Variables

Property:	<pre>plugin.named.org.dspace.content.license. LicenseArgumentFormatter</pre>
	(property key broken up for display purposes only)
Example Value:	<pre>plugin.named.org.dspace.content.license.LicenseArgumentFormatter = \ org.dspace.content.license.SimpleDSpaceObjectLicenseFormatter = collection, \ org.dspace.content.license.SimpleDSpaceObjectLicenseFormatter = item, \ org.dspace.content.license.SimpleDSpaceObjectLicenseFormatter = eperson</pre>
Informational Note:	It is possible include contextual information in the submission license using substitution variables. The text substitution is driven by a plugin implementation.

Syndication Feed (RSS) Settings

This will enable syndication feeds, links display on community and collection home pages. This setting is not used by the XMLUI, as you enable feeds in your theme.

Property:	<code>webui.feed.enable</code>
Example Value:	<code>webui.feed.enable = true</code>
Informational Note:	By default, RSS feeds are set to true (on) . Change key to "false" to disable.
Property:	<code>webui.feed.items</code>
Example Value:	<code>webui.feed.items = 4</code>
Informational Note:	Defines the number of DSpace items per feed (the most recent submissions)
Property:	<code>webui.feed.cache.size</code>
Example Value:	<code>webui.feed.cache.size = 100</code>

Informational Note:	Defines the maximum number of feeds in memory cache. Value of "0" will disable caching.
Property:	<code>webui.feed.cache.age</code>
Example Value:	<code>webui.feed.cache.age = 48</code>
Informational Note:	Defines the number of hours to keep cached feeds before checking currency. The value of "0" will force a check with each request.
Property:	<code>webui.feed.formats</code>
Example Value:	<code>webui.feed.formats = rss_1.0,rss_2.0,atom_1.0</code>
Informational Note:	Defines which syndication formats to offer. You can use more than one; use a comma-separated list. The following list are the available values: <code>rss_0.90</code> , <code>rss_0.91</code> , <code>rss_0.92</code> , <code>rss_0.93</code> , <code>rss_0.94</code> , <code>rss_1.0</code> , <code>rss_2.0</code> , <code>atom_1.0</code> .
Property:	<code>webui.feed.localresolve</code>
Example Value:	<code>webui.feed.localresolve = false</code>
Informational Note:	By default, (set to false), URLs returned by the feed will point at the global handle resolver (e.g. http://hdl.handle.net/123456789/1). If set to <i>true</i> the local server URLs are used (e.g. http://myserver.myorg/handle/123456789/1).
Property:	<code>webui.feed.item.title</code>
Example Value:	<code>webui.feed.item.title = dc.title</code>
Informational Note:	This property customizes each single-value field displayed in the feed information for each item . Each of the fields takes a single metadata field. The form of the key is <code><scheme prefix>.<element>.<qualifier></code> In place of the qualifier, one may leave it blank to exclude any qualifiers or use the wildcard "*" to include all qualifiers for a particular element.
Property:	<code>webui.feed.item.date</code>
Example Value:	<code>webui.feed.item.date = dc.date.issued</code>
Informational Note:	This property customizes each single-value field displayed in the feed information for each item . Each of the fields takes a single metadata field. The form of the key is <code><scheme prefix>.<element>.<qualifier></code> In place of the qualifier, one may leave it blank to exclude any qualifiers or use the wildcard "*" to include all qualifiers for a particular element.

Property:	<code>webui.feed.item.description</code>
Example Value:	<pre>webui.feed.item.description = dc.title, dc.contributor.author, \ dc.contributor.editor, dc.description.abstract, \ dc.description</pre>
Informational Note:	<p>One can customize the metadata fields to show in the feed for each item's description. Elements are displayed in the order they are specified in <i>dspace.cfg</i>. Like other property keys, the format of this property key is: <i>webui.feed.item.description = <scheme prefix>.<element>.<qualifier></i>. In place of the qualifier, one may leave it blank to exclude any qualifiers or use the wildcard "*" to include all qualifiers for a particular element.</p>
Property:	<code>webui.feed.item.author</code>
Example Value:	<code>webui.feed.item.author = dc.contributor.author</code>
Informational Note:	The name of field to use for authors (Atom only); repeatable.
Property:	<code>webui.feed.logo.url</code>
Example Value:	<code>webui.feed.logo.url = \${dspace.url}/themes/mysite/images/mysite-logo.png</code>
Informational Note:	Customize the image icon included with the site-wide feeds. This must be an absolute URL.
Property:	<code>webui.feed.item.dc.creator</code>
Example Value:	<code>webui.feed.item.dc.creator = dc.contributor.author</code>
Informational Note:	This optional property adds <i>structured</i> DC elements as XML elements to the feed description. They are not the same thing as, for example, <i>webui.feed.item.description</i> . Useful when a program or stylesheet will be transforming a feed and wants separate author, description, date, etc.
Property:	<code>webui.feed.item.dc.date</code>
Example Value:	<code>webui.feed.item.dc.date = dc.date.issued</code>
Informational Note:	This optional property adds <i>structured</i> DC elements as XML elements to the feed description. They are not the same thing as, for example, <i>webui.feed.item.description</i> . Useful when a

	program or stylesheet will be transforming a feed and wants separate author, description, date, etc.
Property:	<code>webui.feed.item.dc.description</code>
Example Value:	<code>webui.feed.item.dc.description = dc.description.abstract</code>
Informational Note:	This optional property adds <i>structured</i> DC elements as XML elements to the feed description. They are not the same thing as, for example, <i>webui.feed.item.description</i> . Useful when a program or stylesheet will be transforming a feed and wants separate author, description, date, etc.
Property:	<code>webui.feed.podcast.collections</code>
Example Value:	<code>webui.feed.podcast.collections = 1811/45183,1811/47223</code>
Informational Note:	This optional property enables Podcast Support on the RSS feed for the specified collection handles. The podcast is iTunes compatible and will expose the bitstreams in the items for viewing and download by the podcast reader. Multiple values are separated by commas. For more on using/enabling Media RSS Feeds to share content via iTunesU, see: Enable Media RSS Feeds
Property:	<code>webui.feed.podcast.communities</code>
Example Value:	<code>webui.feed.podcast.communities = 1811/47223</code>
Informational Note:	This optional property enables Podcast Support on the RSS feed for the specified community handles. The podcast is iTunes compatible and will expose the bitstreams in the items for viewing and download by the podcast reader. Multiple values are separated by commas. For more on using/enabling Media RSS Feeds to share content via iTunesU, see: Enable Media RSS Feeds
Property:	<code>webui.feed.podcast.mimetypes</code>
Example Value:	<code>webui.feed.podcast.mimetypes = audio/x-mpeg,application/pdf</code>
Informational Note:	This optional property for Podcast Support, allows you to choose which MIME types of bitstreams are to be enclosed in the podcast feed. Multiple values are separated by commas. For more on using/enabling Media RSS Feeds to share content via iTunesU, see: Enable Media RSS Feeds
Property:	<code>webui.feed.podcast.sourceuri</code>

Example Value:	<code>webui.feed.podcast.sourceuri = dc.source.uri</code>
Informational Note:	This optional property for the Podcast Support will allow you to use a value for a metadata field as a replacement for actual bitstreams to be enclosed in the RSS feed. A use case for specifying the external sourceuri would be if you have a non-DSpace media streaming server that has a copy of your media file that you would prefer to have the media streamed from. For more on using/enabling Media RSS Feeds to share content via iTunesU, see: Enable Media RSS Feeds

OpenSearch Support

OpenSearch is a small set of conventions and documents for describing and using "search engines", meaning any service that returns a set of results for a query. See extensive description in the *Business Layer section* of the documentation.

Please note that for result data formatting, OpenSearch uses Syndication Feed Settings (RSS). So, even if Syndication Feeds **are not** enable, they **must** be configured to enable OpenSearch. OpenSearch uses all the configuration properties for DSpace RSS to determine the mapping of metadata fields to feed fields. Note that a new field for authors has been added (used in Atom format only).

Property:	<code>websvc.opensearch.enable</code>
Example Value:	<code>websvc.opensearch.enable = false</code>
Informational Note:	Whether or not OpenSearch is enabled. By default, the feature is disabled. Change the property key to "true" to enable.
Property:	<code>websvc.opensearch.uicontext</code>
Example Value:	<code>websvc.opensearch.uicontext = simple-search</code>
Informational Note:	Context for HTML request URLs. Change only for non-standard servlet mapping. IMPORTANT: If you are using XMLUI and have Discovery enabled, this property's value should be changed to <i>discover</i> .
Property:	<code>websvc.opensearch.svccontext</code>
Example Value:	<code>websvc.opensearch.svccontext = open-search/</code>
Informational Note:	Context for RSS/Atom request URLs. Change only for non-standard servlet mapping. IMPORTANT: If you are using XMLUI and have Discovery enabled, this property's value should be changed to <code>open-search/</code> <i>discover</i> .

Property:	<code>websvc.opensearch.autolink</code>
Example Value:	<code>websvc.opensearch.autolink = true</code>
Informational Note:	Present autodiscovery link in every page head.
Property:	<code>websvc.opensearch.validity</code>
Example Value:	<code>websvc.opensearch.validity = 48</code>
Informational Note:	Number of hours to retain results before recalculating. This applies to the Manakin interface only.
Property:	<code>websvc.opensearch.shortname</code>
Example Value:	<code>websvc.opensearch.shortname = DSpace</code>
Informational Note:	A short name used in browsers for search service. It should be sixteen (16) or fewer characters .
Property:	<code>websvc.opensearch.longname</code>
Example Value:	<code>websvc.opensearch.longname = \${dspace.name}</code>
Informational Note:	A longer name up to 48 characters.
Property:	<code>websvc.opensearch.description</code>
Example Value:	<code>websvc.opensearch.description = \${dspace.name} DSpace repository</code>
Informational Note:	Brief service description
Property:	<code>websvc.opensearch.faviconurl</code>
Example Value:	<code>_websvc.opensearch.faviconurl = http://www.dspace.org/images/favicon.ico_</code>
Informational Note:	Location of favicon for service, if any. They must by 16 x 16 pixels. You can provide your own local favicon instead of the default.
Property:	<code>websvc.opensearch.samplequery</code>

Example Value:	<code>websvc.opensearch.samplequery = photosynthesis</code>
Informational Note:	Sample query. This should return results. You can replace the sample query with search terms that should actually yield results in your repository.
Property:	<code>websvc.opensearch.tags</code>
Example Value:	<code>websvc.opensearch.tags = IR DSpace</code>
Informational Note:	Tags used to describe search service.
Property:	<code>websvc.opensearch.formats</code>
Example Value:	<code>websvc.opensearch.formats = html,atom,rss</code>
Informational Note:	Result formats offered. Use one or more comma-separated from the list: html, atom, rss. Please note that html is required for auto discovery in browsers to function, and must be the first in the list if present.

Content Inline Disposition Threshold

The following configuration is used to change the disposition behavior of the browser. That is, when the browser will attempt to open the file or download it to the user-specified location. For example, the default size is 8MB. When an item being viewed is larger than 8MB, the browser will download the file to the desktop (or wherever you have it set to download) and the user will have to open it manually.

Property:	<code>webui.content_disposition_threshold</code>
Example value:	<code>webui.content_disposition_threshold = 8388608</code>
Informational Note:	The default value is set to 8MB. This property key applies to the JSPUI interface.
Property:	<code>xmlui.content_disposition_threshold</code>
Example Value:	<code>xmlui.content_disposition_threshold = 8388608</code>
Informational Note:	The default value is set to 8MB. This property key applies to the XMLUI (Manakin) interface.

Other values are possible:

4 MB = 4194304 MB = 838860816 MB = 16777216

Multi-file HTML Document/Site Settings

The setting is used to configure the "depth" of request for html documents bearing the same name.

Property:	<code>webui.html.max-depth-guess</code>
Example Value:	<code>webui.html.max-depth-guess = 3</code>
Informational Note:	When serving up composite HTML items in the JSP UI, how deep can the request be for us to serve up a file with the same name? For example, if one receives a request for " <i>foo/bar/index.htm</i> " and one has a bitstream called just " <i>index.htm</i> ", DSpace will serve up the former bitstream (<i>foo/bar/index.htm</i>) for the request if <i>webui.html.max-depth-guess</i> is 2 or greater. If <i>webui.html.max-depth-guess</i> is 1 or less, then DSpace would not serve that bitstream, as the depth of the file is greater. If <i>webui.html.max-depth-guess</i> is zero, the request filename and path must always exactly match the bitstream name. The default is set to 3.
Property:	<code>xmlui.html.max-depth-guess</code>
Example Value:	<code>xmlui.html.max-depth-guess = 3</code>
Informational Note:	When serving up composite HTML items in the XMLUI, how deep can the request be for us to serve up a file with the same name? For example, if one receives a request for " <i>foo/bar/index.htm</i> " and one has a bitstream called just " <i>index.htm</i> ", DSpace will serve up the former bitstream (<i>foo/bar/index.htm</i>) for the request if <i>webui.html.max-depth-guess</i> is 2 or greater. If <i>xmlui.html.max-depth-guess</i> is 1 or less, then DSpace would not serve that bitstream, as the depth of the file is greater. If <i>_webui.html.max-depth-guess_</i> is zero, the request filename and path must always exactly match the bitstream name. The default is set to 3.

Sitemap Settings

To aid web crawlers index the content within your repository, you can make use of sitemaps.

Property:	<code>sitemap.dir</code>
Example Value:	<code>sitemap.dir = \${dspace.dir}/sitemaps</code>
Informational Note:	The directory where the generate sitemaps are stored.
Property:	<code>sitemap.engineurls</code>
Example Value:	<code>_sitemap.engineurls = http://www.google.com/webmasters/sitemaps/ping?sitemap=_</code>

Informational Note:	Comma-separated list of search engine URLs to "ping" when a new Sitemap has been created. Include everything except the Sitemap UL itself (which will be URL-encoded and appended to form the actual URL "pinged"). Add the following to the above parameter if you have an application ID with Yahoo: http://search.yahooapis.com/SiteExplorerService/V1/updateNotification?appid=REPLACE_ME?url=_ . (Replace the component <code>REPLACE_ME</code> with your application ID). There is no known "ping" URL for MSN/Live search.
---------------------	--

Authority Control Settings

Two features fall under the header of Authority Control: Choice Management and Authority Control of Item ("DC") metadata values. Authority control is a fully optional feature in DSpace 1.6. Implemented out of the box are the Library of Congress Names service, and the Sherpa Romeo authority plugin.

For an in-depth description of this feature, please consult: [Authority Control of Metadata Values](#)

Property:	<code>plugin.named.org.dspace.content.authority.ChoiceAuthority</code>
Example Value:	<pre>plugin.named.org.dspace.content.authority.ChoiceAuthority = \ org.dspace.content.authority.SampleAuthority = Sample, \ org.dspace.content.authority.LCNameAuthority = LCNameAuthority, \ org.dspace.content.authority.SHERPAROMEOPublisher = SRPublisher, \ org.dspace.content.authority.SHERPAROMEJournalTitle = SRJournalTitle</pre>
Informational Note:	--
Property:	<code>plugin.selfnamed.org.dspace.content.authority.ChoiceAuthority</code>
Example Value:	<pre>plugin.selfnamed.org.dspace.content.authority.ChoiceAuthority = \ org.dspace.content.authority.DCInputAuthority</pre>
Property:	<code>lcname.url</code>
Example Value:	<code>lcname.url = http://alcme.oclc.org/srw/search/lcnaf_</code>
Informational Note:	Location (URL) of the Library of Congress Name Service
Property:	<code>sherpa.romeo.url / sherpa.romeo.apikey</code>

Informational Note:	Please refers to the Sherpa/RoMEO Publishers Policy Database Integration section for details about such properties. See Configuring the Sherpa/RoMEO Publishers Policy Database Integration
Property:	<code>authority.minconfidence</code>
Example Value:	<code>authority.minconfidence = ambiguous</code>
Informational Note:	This sets the default lowest confidence level at which a metadata value is included in an authority-controlled browse (and search) index. It is a symbolic keyword, one of the following values (listed in descending order): accepted, uncertain, ambiguous, notfound, failed, rejected, novalue, unset. See <code>org.dspace.content.authority.Choices</code> source for descriptions.
Property:	<code>xmlui.lookup.select.size</code>
Example Value:	<code>xmlui.lookup.select.size = 12</code>
Informational Note:	This property sets the number of selectable choices in the Choices lookup popup

JSPUI Upload File Settings

To alter these properties for the XMLUI, please consult the Cocoon specific configuration at `/WEB-INF/cocoon/properties/core.properties`.

Property:	<code>upload.temp.dir</code>
Example Value:	<code>upload.temp.dir = \${dspace.dir}/upload</code>
Informational Note:	This property sets where DSpace temporarily stores uploaded files.
Property:	<code>upload.max</code>
Example Value:	<code>upload.max = 536870912</code>
Informational Note:	Maximum size of uploaded files in bytes. A negative setting will result in no limit being set. The default is set for 512Mb.

JSP Web Interface (JSPUI) Settings

The following section is limited to JSPUI. If the user wishes to use XMLUI settings, please refer to Chapter 7: XMLUI Configuration and Customization.

Property:	<code>webui.itemdisplay.default</code>
Example Value:	<pre>webui.itemdisplay.default = dc.title, dc.title.alternative, \ dc.contributor.*, dc.subject, dc.data.issued(date), \ dc.publisher, dc.identifier.citation, \ dc.relation.ispartofseries, dc.description.abstract, \ dc.description, dc.identifier.govdoc, \ dc.identifier.uri(link), dc.identifier.isbn, \ dc.identifier.issn, dc.identifier.ismn, dc.identifier</pre>
Informational Note:	<p>This is used to customize the DC metadata fields that display in the item display (the brief display) when pulling up a record. The format is: <code><schema>.<element>.<_optional_qualifier></code>. In place of the qualifier, one can use the wildcard "*" to include all fields of the same element, or, leave it blank for unqualified elements. Additionally, two additional options are available for behavior/rendering: (date) and (link). See the following examples:</p> <pre>dc.title = Dublin Core element "title" (unqualified) dc.title.alternative = DC element "title", qualifier "alternative" dc.title.* = All fields with Dublin Core element 'title' (any or no qualifier) dc.identifier.uri(link) = DC identifier.uri, rendered as a link dc.date.issued(date) = DC date.issued, rendered as a date</pre> <p>The <code>Messages.properties</code> file controls how the fields defined above will display to the user. If the field is missing from the <code>Messages.properties</code> file, it will not be displayed. Look in <code>Messages.properties</code> under the <code>metadata.dc.<field></code>. Example:</p> <pre>metadata.dc.contributor.other = Authors metadata.dc.contributor.author = Authors metadata.dc.title.* = Title</pre> <p>Please note: The order in which you place the values to the property key control the order in which they will display to the user on the outside world. (See the Example Value above).</p>
Property:	<pre>webui.resolver.1.urn webui.resolver.1.baseurl webui.resolver.2.urn webui.resolver.2.baseurl</pre>

Example Value:	<pre>webui.resolver.1.urn = doi webui.resolver.1.baseurl = http://dx.doi.org/ webui.resolver.2.urn = hdl webui.resolver.2.baseurl = http://hdl.handle.net/</pre>
Informational Note:	<p>When using "resolver" in <i>webui.itemdisplay</i> to render identifiers as resolvable links, the base URL is taken from <code><code>webui.resolver.<n>.baseurl</code></code> where <code><code>webui.resolver.<n>.baseurl</code></code> matches the urn specified in the metadata value. The value is appended to the "baseurl" as is, so the baseurl needs to end with the forward slash almost in any case. If no urn is specified in the value it will be displayed as simple text. For the doi and hdl urn defaults values are provided, respectively http://dc.doi.org and http://hdl.handle.net are used. If a metadata value with style "doi", "handle" or "resolver" matches a URL already, it is simply rendered as a link with no other manipulation.</p>
Property:	<pre>plugin.single.org.dspace.app.webui.util.StyleSelection</pre>
Example Value:	<pre>plugin.single.org.dspace.app.webui.util.StyleSelection = \ org.dspace.app.web.util.CollectionStyleSelection #org.dspace.app.web.util.MetadataStyleSelection</pre>
Informational Note:	<p>Specify which strategy to use for select the style for an item.</p>
Property:	<pre>webui.itemdisplay.thesis.collections</pre>
Example Value:	<pre>webui.itemdisplay.thesis.collections = 123456789/24, 123456789/35</pre>
Informational Note:	<p>Specify which collections use which views by Handle.</p>
Property:	<pre>webui.itemdisplay.metadata-style webui.itemdisplay.metadata-style</pre>
Example Value:	<pre>webui.itemdisplay.metadata-style = schema.element[.qualifier . *] webui.itemdisplay.metadata-style = dc.type</pre>

Informational Note:	Specify which metadata to use as name of the style
Property:	<code>webui.itemlist.columns</code>
Example Value:	<pre>webui.itemlist.columns = thumbnail, dc.date.issued(date), dc.title, \ dc.contributor.*</pre>
Informational Note:	<p>Customize the DC fields to use in the item listing page. Elements will be displayed left to right in the order they are specified here. The form is <code><schema prefix>.<element>[.<qualifier> .*][(date)], ...</code></p> <p>Although not a requirement, it would make sense to include among the listed fields at least the date and title fields as specified by the <code>webui.browse.index</code> configuration options in the next section mentioned. (cf.)</p> <p>If you have enabled thumbnails (<code>webui.browse.thumbnail.show</code>), you must also include a 'thumbnail' entry in your columns, this is where the thumbnail will be displayed.</p>
Property:	<code>webui.itemlist.width</code>
Example Value:	<code>webui.itemlist.width = *, 130, 60%, 40%</code>
Informational Note:	<p>You can customize the width of each column with the following line--you can have numbers (pixels) or percentages. For the 'thumbnail' column, a setting of '*' will use the max width specified for browse thumbnails (cf. <code>webui.browse.thumbnail.maxwidth</code>, <code>thumbnail.maxwidth</code>)</p>
Property:	<pre>webui.itemlist.browse.<index name>.sort.<sort name>.columns webui.itemlist.sort.<sort name>.columns webui.itemlist.browse.<browse name>.columns webui.itemlist.<sort or index name>.columns</pre>
Example Value:	
Informational Note:	<p>You can override the DC fields used on the listing page for a given browse index and/or sort option. As a sort option or index may be defined on a field that isn't normally included in the list , this allows you to display the fields that have been indexed/sorted on. There are a number of forms the configuration can take, and the order in which they are listed below is the priority in which they will be used (so a combination of an index name and sort name will take precedence over just the browse name).In the last case, a sort option name will always take</p>

	precedence over a browse index name. Note also, that for any additional columns you list, you will need to ensure there is an <i>itemlist.<field name></i> entry in the messages file.
Property:	<code>webui.itemlist.dateaccessioned.columns</code>
Example Value:	<code>webui.itemlist.dateaccessioned.columns = thumbnail, dc.date.accessioned(date), dc.title, dc.contributor.*</code>
Informational Note:	This would display the date of the accession in place of the issue date whenever the <code>dateaccessioned</code> browsed index or sort option is selected. Just like <i>webui.itemlist.columns</i> , you will need to include a 'thumbnail' entry to display the thumbnails in the item list.
Property:	<code>webui.itemlist.dateaccessioned.widths</code>
Example Value:	<code>webui.itemlist.dateaccessioned.widths = *, 130, 60%, 40%</code>
Informational Note:	As in the aforementioned property key, you can customize the width of the columns for each configured column list, substituting ".widths" for ".columns" in the property name. See the setting for <i>webui.itemlist.widths</i> for more information.
Property:	<code>webui.itemlist.tablewidth</code>
Example Value:	<code>webui.itemlist.tablewidth = 100%</code>
Informational Note:	You can also set the overall size of the item list table with the following setting. It can lead to faster table rendering when used with the column widths above, but not generally recommended.
Property:	<code>webui.session.invalidate</code>
Example Value:	<code>webui.session.invalidate = true</code>
Informational Note:	Enable or disable session invalidation upon login or logout. This feature is enabled by default to help prevent session hijacking but may cause problems for shibboleth, etc. If omitted, the default value is "true". [Only used for JSPUI authentication].
Property:	<code>jspui.google.analytics.key</code>
Example Value:	<code>jspui.google.analytics.key = UA-XXXXXX-X</code>
Informational Note:	If you would like to use Google Analytics to track general website statistics then use the following parameter to provide your Analytics key.

JSPUI Configuring Multilingual Support

[i18n – Locales]

Setting the Default Language for the Application

Property:	<code>default.locale</code>
Example Value:	<code>default.locale = en</code>
Informational Note:	The default language for the application is set with this property key. This is a locale according to i18n and might consist of country, country_language or country_language_variant. If no default locale is defined, then the server default locale will be used. The format of a local specifier is described here: http://java.sun.com/j2se/1.4.2/docs/api/java/util/Locale.html

Supporting More Than One Language

Changes in `dspace.cfg`

Property:	<code>webui.supported.locales</code>
Example Value:	<code>webui.supported.locales = en, de</code>
or perhaps	<code>webui.supported.locales = en, en_ca, de</code>
Informational Note:	All the locales that are supported by this instance of DSpace. Comma separated list.

The table above, if needed and is used will result in:

- a language switch in the default header
- the user will be enabled to choose his/her preferred language, this will be part of his/her profile
- wording of emails
 - mails to registered users, e.g. alerting service will use the preferred language of the user
 - mails to unregistered users, e.g. suggest an item will use the language of the session
- according to the language selected for the session, using `dspace-admin Edit News` will edit the news file of the language according to session

Related Files

If you set `webui.supported.locales` make sure that all the related additional files for each language are available. *LOCALE* should correspond to the locale set in `webui.supported.locales`, e. g.: for `webui.supported.locales = en, de, fr`, there should be:

- `[dspace-source]/dspace/modules/jspui/src/main/resources/Messages.properties`
- `[dspace-source]/dspace/modules/jspui/src/main/resources/Messages_en.properties`

- [dspace-source]/dspace/modules/jspui/src/main/resources/Messages_de.properties
- [dspace-source]/dspace/modules/jspui/src/main/resources/Messages_fr.properties
- Files to be localized:
- [dspace-source]/dspace/modules/jspui/src/main/resources/Messages_LOCALE.properties
- [dspace-source]/dspace/config/input-forms_LOCALE.xml
- [dspace-source]/dspace/config/default_LOCALE.license - should be pure ASCII
- [dspace-source]/dspace/config/news-top_LOCALE.html
- [dspace-source]/dspace/config/news-side_LOCALE.html
- [dspace-source]/dspace/config/emails/change_password_LOCALE
- [dspace-source]/dspace/config/emails/feedback_LOCALE
- [dspace-source]/dspace/config/emails/internal_error_LOCALE
- [dspace-source]/dspace/config/emails/register_LOCALE
- [dspace-source]/dspace/config/emails/submit_archive_LOCALE
- [dspace-source]/dspace/config/emails/submit_reject_LOCALE
- [dspace-source]/dspace/config/emails/submit_task_LOCALE
- [dspace-source]/dspace/config/emails/subscription_LOCALE
- [dspace-source]/dspace/config/emails/suggest_LOCALE
- [dspace]/webapps/jspui/help/collection-admin_LOCALE.html - in html keep the jump link as original; must be copied to [dspace-source]/dspace/modules/jspui/src/main/webapp/help
- [dspace]/webapps/jspui/help/index_LOCALE.html - must be copied to [dspace-source]/dspace/modules/jspui/src/main/webapp/help
- [dspace]/webapps/jspui/help/site-admin_LOCALE.html - must be copied to [dspace-source]/dspace/modules/jspui/src/main/webapp/help

JSPUI Item Mapper

Because the item mapper requires a primitive implementation of the browse system to be present, we simply need to tell that system which of our indexes defines the author browse (or equivalent) so that the mapper can list authors' items for mapping

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

Property:	itemmap.author.index
Example Value:	itemmap.author.index = author
Informational Note:	If you change the name of your author browse field, you will also need to update this property key.

Display of Group Membership

Property:	<code>webui.mydspace.showgroupmembership</code>
Example Value:	<code>webui.mydspace.showgroupmembership = false</code>
Informational Note:	To display group membership set to "true". If omitted, the default behavior is false.

JSPUI / XMLUI SFX Server

SFX Server is an OpenURL Resolver.

Property:	<code>sfx.server.url</code>
Example Value	<code>sfx.server.url = http://sfx.myu.edu:8888/sfx?</code>
:	
	<code>sfx.server.url = http://worldcatlibraries.org/registry/gateway?</code>
Informational Note:	SFX query is appended to this URL. If this property is commented out or omitted, SFX support is switched off.

All the parameters mapping are defined in `[dspace]/config/sfx.xml` file. The program will check the parameters in `sfx.xml` and retrieve the correct metadata of the item. It will then parse the string to your resolver.

For the following example, the program will search the first query-pair which is DOI of the item. If there is a DOI for that item, your retrieval results will be, for example:

`http://researchspace.auckland.ac.nz/handle/2292/5763`

Example. For setting DOI in `sfx.xml`

```
<query-pairs>
  <field>
    <querystring>rft_id=info:doi/</querystring>
    <dc-schema>dc</dc-schema>
    <dc-element>identifier</dc-element>
    <dc-qualifier>doi</dc-qualifier>
  </field>
</query-pairs>
```

If there is no DOI for that item, it will search next query-pair based on the `[dspace]/config/sfx.xml` and then so on.

Example of using ISSN, volume, issue for item without DOI

[<http://researchspace.auckland.ac.nz/handle/2292/4947>]

For parameter passing to the `<querystring>`

```
<querystring>rft_id=info:doi/</querystring>
```

Please refer to these:

[<http://ocoins.info/cobgbook.html>]

[<http://ocoins.info/cobg.html>]

Program assume won't get empty string for the item, as there will at least author, title for the item to pass to the resolver.

For contributor author, program maintains original DSpace SFX function of extracting author's first and last name.

```
<field>
  <querystring>rft.aulast=</querystring>
  <dc-schema>dc</dc-schema>
  <dc-element>contributor</dc-element>
  <dc-qualifier>author</dc-qualifier>
</field>
<field>
  <querystring>rft.aufirst=</querystring>
  <dc-schema>dc</dc-schema>
  <dc-element>contributor</dc-element>
  <dc-qualifier>author</dc-qualifier>
</field>
```

JSPUI Item Recommendation Setting

Property:	<code>webui.suggest.enable</code>
Example Value:	<code>webui.suggest.enable = true</code>
Informational Note :	Show a link to the item recommendation page from item display page.
Property:	<code>webui.suggest.loggedinusers.only</code>
Example Value:	<code>webui.suggest.loggedinusers.only = true</code>
Informational Note :	Enable only if the user is logged in. If this key commented out, the default value is false.

Controlled Vocabulary Settings

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

Property:	<code>webui.controlledvocabulary.enable</code>
Example Value:	<code>webui.controlledvocabulary.enable = true</code>
Informational Note:	Enable or disable the controlled vocabulary add-on. WARNING: This feature is not compatible with WAI (it requires JavaScript to function).

The need for a limited set of keywords is important since it eliminates the ambiguity of a free description system, consequently simplifying the task of finding specific items of information.

The controlled vocabulary add-on allows the user to choose from a defined set of keywords organized in a tree (taxonomy) and then use these keywords to describe items while they are being submitted.

We have also developed a small search engine that displays the classification tree (or taxonomy) allowing the user to select the branches that best describe the information that he/she seeks.

The taxonomies are described in XML following this (very simple) structure:

```

<node id="acmccs98" label="ACMCCS98">
  <isComposedBy>
    <node id="A." label="General Literature">
      <isComposedBy>
        <node id="A.0" label="GENERAL"/>
        <node id="A.1" label="INTRODUCTORY AND SURVEY"/>
      </isComposedBy>
    </node>
  </isComposedBy>
</node>

```

You are free to use any application you want to create your controlled vocabularies. A simple text editor should be enough for small projects. Bigger projects will require more complex tools. You may use Protegé to create your taxonomies, save them as OWL and then use a XML Stylesheet (XSLT) to transform your documents to the appropriate format. Future enhancements to this add-on should make it compatible with standard schemas such as OWL or RDF.

In order to make DSpace compatible with WAI 2.0, the add-on is **turned off** by default (the add-on relies strongly on JavaScript to function). It can be activated by setting the following property in `dspace.cfg`:

```
webui.controlledvocabulary.enable = true
```

New vocabularies should be placed in `[dspace]/config/controlled-vocabularies/` and must be according to the structure described. A validation XML Schema (named `controlledvocabulary.xsd`) is also available in that directory.

Vocabularies need to be associated with the correspondent DC metadata fields. Edit the file `[dspace]/config/input-forms.xml` and place a `"vocabulary"` tag under the `"field"` element that you want to control. Set value of the `"vocabulary"` element to the name of the file that contains the vocabulary, leaving out the extension (the add-on will only load files with extension `"*.xml"`). For example:

```
<field>
  <dc-schema>dc</dc-schema>
  <dc-element>subject</dc-element>
  <dc-qualifier></dc-qualifier>
  <!-- An input-type of twobox MUST be marked as repeatable -->
  <repeatable>true</repeatable>
  <label>Subject Keywords</label>
  <input-type>twobox</input-type>
  <hint> Enter appropriate subject keywords or phrases below. </hint>
  <required></required>
  <vocabulary [closed="false"]>nsi</vocabulary>
</field>
```

The vocabulary element has an optional boolean attribute **closed** that can be used to force input only with the JavaScript of controlled-vocabulary add-on. The default behavior (i.e. without this attribute) is as set **closed="false"**. This allow the user also to enter the value in free way.

The following vocabularies are currently available by default:

- **nsi** - *nsi.xml* - The Norwegian Science Index
- **srsc** - *srsc.xml* - Swedish Research Subject Categories

3. JSPUI Session Invalidation

Property:	<code>webui.session.invalidate</code>
Example Value:	<code>webui.session.invalidate = true</code>
Informational Note:	Enable or disable session invalidation upon login or logout. This feature is enabled by default to help prevent session hijacking but may cause problems for shibboleth, etc. If omitted, the default value is 'true'.

XMLUI Specific Configuration

The DSpace digital repository supports two user interfaces: one based upon JSP technologies and the other based upon the Apache Cocoon framework. This section describes those configurations settings which are

specific to the XMLUI interface based upon the Cocoon framework. (*Prior to DSpace Release 1.5.1 XMLUI was referred to Manakin. You may still see references to "Manakin"*)

Property:	<code>xmlui.force.ssl</code>
Example Value:	<code>xmlui.force.ssl = true</code>
Informational Note:	Force all authenticated connections to use SSL, only non-authenticated connections are allowed over plain http. If set to true, then you need to ensure that the " <i>dspace.hostname</i> " parameter is set correctly.
Property:	<code>xmlui.user.registration</code>
Example Value:	<code>xmlui.user.registration = true</code>
Informational Note:	Determine if new users should be allowed to register. This parameter is useful in conjunction with Shibboleth where you want to disallow registration because Shibboleth will automatically register the user. Default value is true.
Property:	<code>xmlui.user.editmetadata</code>
Example Value:	<code>xmlui.user.editmetadata = true</code>
Informational Note:	Determines if users should be able to edit their own metadata. This parameter is useful in conjunction with Shibboleth where you want to disable the user's ability to edit their metadata because it came from Shibboleth. Default value is true.
Property:	<code>xmlui.session.ipcheck</code>
Example Value:	<code>xmlui.session.ipcheck = true</code>
Informational Note:	Check if the user has a consistent ip address from the start of the login process to the end of the login process. Disabling this check is not recommended unless absolutely necessary as the ip check can be helpful for preventing session hijacking. Possible reasons to set this to false: many-to-many wireless networks that prevent consistent ip addresses or complex proxying of requests. The default value is true.
Property:	<code>xmlui.user.loginredirect</code>
Example Value:	<code>xmlui.user.loginredirect = /profile</code>

Informational Note:	After a user has logged into the system, which url should they be directed? Leave this parameter blank or undefined to direct users to the homepage, or <code>/profile</code> for the user's profile, or another reasonable choice is <code>/submissions</code> to see if the user has any tasks awaiting their attention. The default is the repository home page.
Property:	<code>xmlui.theme.allowoverrides</code>
Example Value:	<code>xmlui.theme.allowoverrides = false</code>
Informational Note:	Allow the user to override which theme is used to display a particular page. When submitting a request add the HTTP parameter "themepath" which corresponds to a particular theme, that specified theme will be used instead of the any other configured theme. Note that this is a potential security hole allowing execution of unintended code on the server, this option is only for development and debugging it should be turned off for any production repository. The default value unless otherwise specified is "false".
Property:	<code>xmlui.theme.enableConcatenation</code>
Example Value:	<code>xmlui.theme.enableConcatenation = false</code>
Informational Note:	Enabling this property will concatenate CSS, JS and JSON files where possible. CSS files can be concatenated if multiple CSS files with the same media attribute are used in the same page. Links to the CSS files are automatically referring to the concatenated resulting CSS file. The theme sitemap should be updated to use the ConcatenationReader for all js, css and json files before enabling this property.
Property:	<code>xmlui.theme.enableMinification</code>
Example Value:	<code>xmlui.theme.enableMinification = false</code>
Informational Note:	Enabling this property will minify CSS, JS and JSON files where possible. The theme sitemap should be updated to use the ConcatenationReader for all js, css and json files before enabling this property.
Property:	<code>xmlui.theme.mirage.item-list.emphasis</code>
Example Value:	<code>xmlui.theme.mirage.item-list.emphasis = file</code>
Informational Note:	When set to "file" the item listings in your repository will include the generated thumbnails of uploaded files. Alternatively, you can set this parameter to metadata to put more emphasis on the metadata and effectively hide the thumbnails. The default value is "metadata".

Property:	mirage2.item-view.bitstream.href.label.1 mirage2.item-view.bitstream.href.label.2
Example Value:	mirage2.item-view.bitstream.href.label.1 = label mirage2.item-view.bitstream.href.label.2 = title
Informational Note:	Mirage 2 theme ONLY Determines if the bitstream filename (title) or description (label) is being used as the display label on the hyperlinks to download the actual files. By default, the file description (label) will be shown. If this value is empty, the filename (title) will be used as a fallback. More information and screenshots.
Property:	xmlui.bundle.upload
Example Value:	xmlui.bundle.upload = ORIGINAL, METADATA, THUMBNAIL, LICENSE, CC_LICENSE
Informational Note:	Determine which bundles administrators and collection administrators may upload into an existing item through the administrative interface. If the user does not have the appropriate privileges (add and write) on the bundle then that bundle will not be shown to the user as an option.
Property:	xmlui.community-list.render.full
Example Value:	xmlui.community-list.render.full = true
Informational Note:	On the community-list page should all the metadata about a community/collection be available to the theme. This parameter defaults to true, but if you are experiencing performance problems on the community-list page you should experiment with turning this option off.
Property:	xmlui.community-list.cache
Example Value:	xmlui.community-list.cache = 12 hours
Informational Note:	Normally, the XMLUI will fully verify any cache pages before using a cache copy. This means that when the community-list page is viewed the database is queried for each community/collection to see if their metadata has been modified. This can be expensive for repositories with a large community tree. To help solve this problem you can set the cache to be assumed valued for a specific set of time. The downside of this is that new or editing communities/collections may not show up the website for a period of time.
Property:	xmlui.bitstream.mods
	xmlui.bitstream.mods = true

Example Value:	
Informational Note:	Optionally, you may configure XMLUI to take advantage of metadata stored as a bitstream. The MODS metadata file must be inside the "METADATA" bundle and named MODS.xml. If this option is set to 'true' and the bitstream is present then it is made available to the theme for display.
Property:	<code>xmlui.bitstream.mets</code>
Example Value:	<code>xmlui.bitstream.mets = true</code>
Informational Note:	Optionally, you may configure Manakin to take advantage of metadata stored as a bitstream. The METS metadata file must be inside the "METADATA" bundle and named METS.xml. If this option is set to "true" and the bitstream is present then it is made available to the theme for display.
Property:	<code>xmlui.google.analytics.key</code>
Example Value:	<code>xmlui.google.analytics.key = UA-XXXXXX-X</code>
Informational Note:	If you would like to use Google Analytics to track general website statistics then use the following parameter to provide your analytics key. First sign up for an account at http://analytics.google.com , then create an entry for your repositories website. Google Analytics will give you a snippet of javascript code to place on your site, inside that snip it is your Google Analytics key usually found in the line: <code>_uacct = "UA-XXXXXX-X"</code> Take this key (just the UA-XXXXXX-X part) and place it here in this parameter.
Property:	<code>xmlui.controlpanel.activity.max</code>
Example Value:	<code>xmlui.controlpanel.activity.max = 250</code>
Informational Note:	Assign how many page views will be recorded and displayed in the control panel's activity viewer. The activity tab allows an administrator to debug problems in a running DSpace by understanding who and how their dspace is currently being used. The default value is 250.
Property:	<code>xmlui.controlpanel.activity.ipheader</code>
Example Value:	<code>xmlui.controlpanel.activity.ipheader = X-Forward-For</code>
Informational Note:	Determine where the control panel's activity viewer receives an events IP address from. If your DSpace is in a load balanced environment or otherwise behind a context-switch then you will need to set the parameter to the HTTP parameter that records the original IP address.

7.1.4 Optional or Advanced Configuration Settings

The following section explains how to configure either optional features or advanced features that are not necessary to make DSpace "out-of-the-box"

The Metadata Format and Bitstream Format Registries

The `[dspace]/config/registries` directory contains three XML files. These are used to load the *initial* contents of the Dublin Core Metadata registry and Bitstream Format registry and SWORD metadata registry. After the initial loading (performed by *ant fresh_install* above), the registries reside in the database; the XML files are not updated.

In order to change the registries, you may adjust the XML files before the first installation of DSpace. On an already running instance it is recommended to change bitstream registries via DSpace admin UI, but the metadata registries can be loaded again at any time from the XML files without difficult. The changes made via admin UI are not reflected in the XML files.

Metadata Format Registries

The default metadata schema is Dublin Core, so DSpace is distributed with a default Dublin Core Metadata Registry. Currently, the system requires that every item have a Dublin Core record.

There is a set of Dublin Core Elements, which is used by the system and should not be removed or moved to another schema, see Appendix: Default Dublin Core Metadata registry.

Note: altering a Metadata Registry has no effect on corresponding parts, e.g. item submission interface, item display, item import and vice versa. Every metadata element used in submission interface or item import must be registered before using it.

Note also that deleting a metadata element will delete all its corresponding values.

If you wish to add more metadata elements, you can do this in one of two ways. Via the DSpace admin UI you may define new metadata elements in the different available schemas. But you may also modify the XML file (or provide an additional one), and re-import the data as follows:

```
[dspace]/bin/dspace dsrun org.dspace.administer.MetadataImporter -f [xml file]
```

The XML file should be structured as follows:

```
<dspace-dc-types>
  <dc-type>
    <schema>dc</schema>
    <element>contributor</element>
    <qualifier>advisor</qualifier>
```

```
<scope_note>Use primarily for thesis advisor.</scope_note>
</dc-type>
</dspace-dc-types>
```

Bitstream Format Registry

The bitstream formats recognized by the system and levels of support are similarly stored in the bitstream format registry. This can also be edited at install-time via `[dspace]/config/registries/bitstream-formats.xml` or by the administration Web UI. The contents of the bitstream format registry are entirely up to you, though the system requires that the following two formats are present:

- *Unknown*
- *License*

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

XPDF Filter

This is an alternative suite of MediaFilter plugins that offers faster and more reliable text extraction from PDF Bitstreams, as well as thumbnail image generation. It replaces the built-in default PDF MediaFilter.

If this filter is so much better, why isn't it the default? The answer is that it relies on external executable programs which must be obtained and installed for your server platform. This would add too much complexity to the installation process, so it left out as an optional "extra" step.

Installation Overview

Here are the steps required to install and configure the filters:

1. Install the xpdf tools for your platform, from the downloads at <http://www.foolabs.com/xpdf>
2. Acquire the Sun Java Advanced Imaging Tools and create a local Maven package.
3. Edit DSpace configuration properties to add location of xpdf executables, reconfigure MediaFilter plugins.
4. Build and install DSpace, adding `-Pxpdf-mediafilter-support` to Maven invocation.

Install XPDF Tools

First, download the XPDF suite found at: <http://www.foolabs.com/xpdf> and install it on your server. The executables can be located anywhere, but make a note of the full path to each command.

You may be able to download a binary distribution for your platform, which simplifies installation. Xpdf is readily available for Linux, Solaris, MacOSX, Windows, NetBSD, HP-UX, AIX, and OpenVMS, and is reported to work on AIX, OS/2, and many other systems.

The only tools you *really* need are:

- *pdfinfo* - displays properties and Info dict
- *pdftotext* - extracts text from PDF

- *pdftoppm* - images PDF for thumbnails

Fetch and install *jai_imageio* JAR

Fetch and install the Java Advanced Imaging Image I/O Tools.

For AIX, Sun support has the following: "JAI has native acceleration for the above but it also works in pure Java mode. So as long as you have an appropriate JDK for AIX (1.3 or later, I believe), you should be able to use it. You can download any of them, extract just the jars, and put those in your \$CLASSPATH."

Download the *jai_imageio* library version 1.0_01 or 1.1 found at: https://jai-imageio.dev.java.net/binary-builds.html#Stable_builds .

For these filters you do NOT have to worry about the native code, just the JAR, so choose a download for any platform.

```
curl -O http://download.java.net/media/jai-imageio/builds/release/1.1/jai_imageio-1_1-  
lib-linux-i586.tar.gz  
tar xzf jai_imageio-1_1-lib-linux-i586.tar.gz
```

The preceding example leaves the JAR in *jai_imageio-1_1/lib/jai_imageio.jar* . Now install it in your local Maven repository, e.g.: (changing the path after *file=* if necessary)

```
mvn install:install-file \\  
-Dfile=jai_imageio-1_1/lib/jai_imageio.jar \\  
-DgroupId=com.sun.media \\  
-DartifactId=jai_imageio \\  
-Dversion=1.0_01 \\  
-Dpackaging=jar \\  
-DgeneratePom=true
```

You may have to repeat this procedure for the *jai_core.jar* library, as well, if it is not available in any of the public Maven repositories. Once acquired, this command installs it locally:

```
mvn install:install-file -Dfile=jai_core-1.1.2_01.jar \\  
-DgroupId=javax.media -DartifactId=jai_core -Dversion=1.1.2_01 -Dpackaging=jar -DgeneratePom=tr  
ue
```

Edit DSpace Configuration

First, be sure there is a value for *thumbnail.maxwidth* and that it corresponds to the size you want for preview images for the UI, e.g.: (*NOTE*: this code doesn't pay any attention to *thumbnail.maxheight* but it's best to set it too so the other thumbnail filters make square images.)

```
# maximum width and height of generated thumbnails  
thumbnail.maxwidth= 80
```

```
thumbnail.maxheight = 80
```

Now, add the absolute paths to the XPDF tools you installed. In this example they are installed under */usr/local/bin* (a logical place on Linux and MacOSX), but they may be anywhere.

```
xpdf.path.pdftotext = /usr/local/bin/pdftotext
xpdf.path.pdftoppm = /usr/local/bin/pdftoppm
xpdf.path.pdfinfo = /usr/local/bin/pdfinfo
```

Change the MediaFilter plugin configuration to remove the old *org.dspace.app.mediafilter.PDFFilter* and add the new filters, e.g: (New sections are in bold)

```
filter.plugins = \
    PDF Text Extractor, \
    PDF Thumbnail, \
    HTML Text Extractor, \
    Word Text Extractor, \
    JPEG Thumbnail
    plugin.named.org.dspace.app.mediafilter.FormatFilter = \
    org.dspace.app.mediafilter.XPDF2Text = PDF Text Extractor, \
    org.dspace.app.mediafilter.XPDF2Thumbnail = PDF Thumbnail, \
    org.dspace.app.mediafilter.HTMLFilter = HTML Text Extractor, \
    org.dspace.app.mediafilter.WordFilter = Word Text Extractor, \
    org.dspace.app.mediafilter.JPEGFilter = JPEG Thumbnail, \
    org.dspace.app.mediafilter.BrandedPreviewJPEGFilter = Branded Preview JPEG
```

Then add the input format configuration properties for each of the new filters, e.g.:

```
filter.org.dspace.app.mediafilter.XPDF2Thumbnail.inputFormats = Adobe PDF
filter.org.dspace.app.mediafilter.XPDF2Text.inputFormats = Adobe PDF
```

Finally, if you want PDF thumbnail images, don't forget to add that filter name to the *filter.plugins* property, e.g.:

```
filter.plugins = PDF Thumbnail, PDF Text Extractor, ...
```

Build and Install

Follow your usual DSpace installation/update procedure, only add *-Pxpdf-mediafilter-support* to the Maven invocation:

```
mvn -Pxpdf-mediafilter-support package
ant -Dconfig=[dspace\]/config/dspace.cfg update
```

Configuring Usage Instrumentation Plugins

A usage instrumentation plugin is configured as a singleton plugin for the abstract class `org.dspace.app.statistics.AbstractUsageEvent`.

The Passive Plugin

The Passive plugin is provided as the class `org.dspace.app.statistics.PassiveUsageEvent`. It absorbs events without effect. Use the Passive plugin when you have no use for usage event postings. This is the default if no plugin is configured.

The Tab File Logger Plugin

The Tab File Logger plugin is provided as the class `org.dspace.app.statistics.UsageEventTabFileLogger`. It writes event records to a file in tab-separated column format. If left unconfigured, an error will be noted in the DSpace log and no file will be produced. To specify the file path, provide an absolute path as the value for `usageEvent.tabFileLogger.file` in `dspace.cfg`.

Property:

```
webui.submit.upload.required
```

JSPUI: Per item visual indicators for browse and search results

Visual indicators per item allow users to mark items in browse and search results. This could be useful in many scenarios, some of them follow:

1. If your repository contains items of different type (articles, book chapters, pictures) you can mark the type of each item using an icon.
2. If your repository has items with bitstreams but also has items with no bitstream, you could indicate this fact to the users using the visual indicators
3. If you have applied copyright licences in the bitstreams or items, you could notify users about that in the browse or result list
4. If you want your users to spot some items out of the list easily or if you want to differentiate some items from the others you could use the visual indicators

The visual indicators extension has the following specs:

1. Multiple marks can be added per item (i.e. mark the type of the item and the availability of the bitstreams)
2. Easy configuration of the strategy of what mark to display in every item
3. Marks based on images or a generic class (i.e. a glyphicon icon for bootstrap)
4. Display tooltip when hovering the mark + localization of the tooltip
5. Easy addition of new strategies for any type of mark the user desires
6. Add css styles for the user to configure the position of the marks in the list row

Some theory:

A mark is an instance of the class: **org.dspace.app.itemmarking.ItemMarkingInfo**.

Each mark can have the following properties:

- *imageName*: a path to the image that will be displayed for the specific mark
- *classInfo*: the css class to be applied in the mark (useful if you do not want to add an image but just an icon from the bootstrap glyph icons)
- *link*: the link to be applied in the mark (optional)
- *tooltip*: the tooltip to be shown when hovering over the mark (optional)

When you need to add a mark in an Item then you need to create a strategy that determined what mark to display per item. Strategy classes need to implement the interface:

```
org.dspace.app.itemmarking.ItemMarkingExtractor
```

Your strategy class just needs to implement the following method from the above Interface:

```
public ItemMarkingInfo getItemMarkingInfo(Context context, Item item) throws SQLException;
```

Which is, given an item, return the Mark info to display.

Currently, there are three Strategies included by default:

ItemMarkingMetadataStrategy

This strategy decides the mark to display per item based on a value of a metadata field (i.e. dc:type)

It accepts two properties:

- *metadataField*: the metadata field to be used for searching the value in the form “`schema.element.qualifier`”
- *mapping*: a Java Map of **Strings** to **ItemMarkingInfos**

If the String (key of the map) is found as a value in the metadataField field, then the mark denoted by the value of the map will be displayed.

ItemMarkingCollectionStrategy

This strategy decides the mark to display per item based on the collection this item belongs to.

It accepts one property:

- *mapping*: a Java Map of **Strings** to **ItemMarkingInfos**

The String (key of the map) is the collection handle (i.e.: 123456789/1) and if an items belongs to this collection, the mark denoted by the object of the map will be displayed

ItemMarkingAvailabilityBitStreamStrategy

This strategy decides the mark to display per item based on the availability (exists or not) of a bitstream within the item.

It accepts to properties:

- *nonAvailableImageName*: the image to display for the mark if no bitstreams exist for the item
- *nonAvailableImageName*: the image to display for the mark if at least one bistream exist for the item

Moreover, this strategy add a link in the mark (in case there are bitstreams in the item) to the first bitstream of the item

How to:

In order to enable a mark for the result or browse list you need to change the option:

```
webui.itemlist.columns
```

of the **dspace.cfg** file.

You need to include a 'mark_[value]' key in any column order you like. Do not add the brackets and you can replace the "value" with any word has a meaning for your marking type. You may add multiple marks (i.e.: one in the first column and one at the last)

For example, the following line is a valid option value:

```
webui.itemlist.columns = mark_type, dc.date.issued(date), dc.title, dc.contributor.*,  
mark_availability
```

In the aforementioned case, you just added two marks, one in the first column for the type of the item and one in the last item for the availability.

Now it's time to declare what "**mark_type**" and "**mark_availability**" means. This is done in the Spring configuration file **config/sping/api/item-marking.xml**, via the dependency injection feature.

In this file, for each “**mark_[value]**” key you add in the *dspace.cfg* file, you need to add a Spring bean with **id=org.dspace.app.itemmarking.ItemMarkingExtractor.[value]**. The class of this bean must be an implementation of **org.dspace.app.itemmarking.ItemMarkingExtractor**.

That’s all!

For our example, we need to declare two beans (one for “**mark_type**” and one for “**mark_availability**”).

```

<!-- Enable this strategy in order to mark item based on the value of a metadata field -->
<bean class="org.dspace.app.itemmarking.ItemMarkingMetadataStrategy" id="
org.dspace.app.itemmarking.ItemMarkingExtractor.type">
  <property name="metadataField" value="dc.type" />
  <property name="mapping" ref="typeMap"/>
</bean>
<!-- Enable this strategy in order to mark items based on the availability of their bitstreams -->
<bean class="org.dspace.app.itemmarking.ItemMarkingAvailabilityBitstreamStrategy" id="
org.dspace.app.itemmarking.ItemMarkingExtractor.availability">
  <property name="availableImageName" value="image/available.png" />
  <property name="nonAvailableImageName" value="image/nonavailable.png" />
</bean>

```

For the “**mark_type**”, we have declared the strategy to be **ItemMarkingMetadataStrategy** which means that the value of a metadata field (dc.type in our case) will determine the mark of each item. Here is the mapping:

```

<bean class="java.util.HashMap" id="typeMap">
  <constructor-arg>
    <map>
      <entry>
        <key>
          <value>image</value>
        </key>
        <ref bean="type1MarkingInfo"/>
      </entry>
      <entry>
        <key>
          <value>video</value>
        </key>
        <ref bean="type2MarkingInfo"/>
      </entry>
    </map>
  </constructor-arg>
</bean>

```

Thus, if the value of **dc.type** field is equal to image the “**type1MarkingInfo**” bean will be used for the marking, if it is equal to video the “**type2MarkingInfo**” bean will be used, otherwise, no mark will be displayed.

```

<bean class="org.dspace.app.itemmarking.ItemMarkingInfo" id="type1MarkingInfo">
  <property name="classInfo" value="glyphicon glyphicon-picture"/>

```

```
<property name="tooltip" value="itemlist.mark.type1MarkingInfo"/>
</bean>
<bean class="org.dspace.app.itemmarking.ItemMarkingInfo" id="type2MarkingInfo">
  <property name="imageName" value="image/type2.png"/>
  <property name="tooltip" value="itemlist.mark.type2MarkingInfo"/>
</bean>
```

Tooltip property contains the localized key to display.

Keep in mind that the Strategy that you may write can have its own logic on how to create the **ItemMarkingInfo** per item. The only requirement of the feature is to add in the Spring configuration file the initial beans one for each mark you have declared in the `dspace.cfg` file.

Styling:

The title for the column of each mark is titled based on the localized key “**itemlist.mark_[value]**”, so you just need to add the specific keys in the `messages.properties` files.

Moreover, the following CSS styles are applied to the various aspects of the mark:

- **mark_[value]_th**: a style applied to the column header
- **mark_[value]_tr**: a style applied to the each row

Add these classes to the `css` file and apply any style you like (like centering the text or the image)

7.2 Directories and Files

- 1 [Overview](#)
- 2 [Source Directory Layout](#)
- 3 [Installed Directory Layout](#)
- 4 [Contents of JSPUI Web Application](#)
- 5 [Contents of XMLUI Web Application \(aka Manakin\)](#)
- 6 [Log Files](#)
 - 6.1 [log4j.properties File](#).

7.2.1 Overview

A complete DSpace installation consists of three separate directory trees:

- **The source directory::** This is where (surprise!) the source code lives. Note that the config files here are used only during the initial install process. After the install, config files should be changed in the install directory. It is referred to in this document as *[dspace-source]*.

- **The install directory::** This directory is populated during the install process and also by DSpace as it runs. It contains config files, command-line tools (and the libraries necessary to run them), and usually -- although not necessarily -- the contents of the DSpace archive (depending on how DSpace is configured) . After the initial build and install, changes to config files should be made in this directory. It is referred to in this document as *[dspace]*.
- **The web deployment directory::** This directory is generated by the web server the first time it finds a *dspace.war* file in its *webapps* directory. It contains the unpacked contents of *dspace.war*, i.e. the JSPs and java classes and libraries necessary to run DSpace. Files in this directory should never be edited directly; if you wish to modify your DSpace installation, you should edit files in the source directory and then rebuild. The contents of this directory aren't listed here since its creation is completely automatic. It is usually referred to in this document as *[tomcat]/webapps/dspace*.

7.2.2 Source Directory Layout

- *[dspace-source]*
 - *LICENSE* - DSpace source code license.
 - *README* - Obligatory basic information file.
 - *build.properties* - The basic settings necessary to actually build/install DSpace for the first time
 - *dspace/* - Directory which contains all build and configuration information for DSpace
 - *bin/* - Some shell and Perl scripts for running DSpace command-line tasks. Primary among them is the '*dspace*' [commandline utility](#)
 - *config/* - Configuration files:
 - *controlled-vocabularies/* - Fixed, limited vocabularies used in metadata entry
 - *crosswalks/* - Metadata crosswalks - property files or XSL stylesheets
 - *emails/* - Text and layout templates for emails sent out by the system.
 - *modules/* - Configurations for modules / individual features within DSpace
 - *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.
 - *dspace.cfg* - The Main [DSpace configuration](#) file
 - *dc2mods.cfg* - Mappings from Dublin Core metadata to [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* , *item-submission.xml* - [Submission UI configuration files](#)
 - *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 JSPUI.
 - *news-xmlui.xml* - Text of the front-page news, only used in XMLUI
 - *etc/* - This directory contains administrative files.
 - *postgres/* - Administrative scripts for PostgreSQL
 - *oracle/* - Administrative scripts for Oracle.
 - *modules/* - The Web UI modules "overlay" directory. DSpace uses Maven to automatically look here for any customizations you wish to make to DSpace Web interfaces.

- *jspui* - Contains all customizations for the JSP User Interface.
 - *src/main/resources/* - The overlay for JSPUI *Resources*. This is the location to place any custom Messages.properties files. (*Previously this file had been stored at: `_[dspace-source]/config/language-packs/Messages.properties_`*)
 - *src/main/webapp/* - The overlay for JSPUI Web Application. This is the location to place any custom JSPs to be used by DSpace.
 - *Ini* - Contains all customizations for the Lightweight Network Interface.
 - *oai* - Contains all customizations for the OAI-PMH Interface.
 - *sword* - Contains all customizations for the SWORD (Simple Web-service Offering Repository Deposit) Interface.
 - *xmlui* - Contains all customizations for the XML User Interface (aka Manakin).
 - *src/main/webapp/* - The overlay for XMLUI Web Application. This is the location to place custom Themes or Configurations.
 - *i18n/* - The location to place a custom version of the XMLUI's messages.xml (You have to manually create this folder)
 - *themes/* - The location to place custom Themes for the XMLUI (You have to manually create this folder).
 - *solr/* - Solr configuration files for all Solr indexes used by DSpace.
 - *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*)
- The Source Release contains the following additional directories :-
 - *dspace-api/* - Java API source module
 - *dspace-jspui/* - JSP-UI source module
 - *dspace-Ini* - [Lightweight Network Interface](#) source module (*deprecated as of 5.0*)
 - *dspace-oai* - [OAI-PMH](#) source module
 - *dspace-rdf* - [RDF](#) source module
 - *dspace-rest* - [REST API](#) source module
 - *dspace-services* - Common Services module
 - *dspace-sword* - [SWORD](#) (Simple Web-serve Offering Repository Deposit) deposit service source module
 - *dspace-swordv2* - [SWORDv2](#) source module
 - *dspace-xmlui* - [XML-UI](#) (Manakin) source module
 - *dspace-xmlui-mirage2* - [Mirage 2 theme](#) for the XMLUI
 - *pom.xml* - DSpace Parent Project definition

7.2.3 Installed Directory Layout

Below is the basic layout of a DSpace installation using the default configuration. These paths can be configured if necessary.

- *[dspace]*
 - *assetstore/*- assetstore files. This is where all the files uploaded into DSpace are stored by default. See [Storage Layer](#).
 - *bin/*- shell scripts for DSpace command-line tasks. Primary among them is the '[dspace commandline utility](#)'
 - *config/*- configuration, with sub-directories as above
 - *etc/*- Administrative and database management files
 - *exports/*- temporary storage for any export packages
 - *handle-server/*- Handles server files and configuration
 - *imports/*- temporary storage for any import packages
 - *lib/*- JARs, including *dspace-api.jar*, containing the DSpace classes
 - *log/*- Log files
 - *reports/*- Reports generated by statistical report generator
 - *search/*- Lucene search index files (when Lucene is enabled)
 - *solr/*- Solr search/browse indexes
 - *triplestore/*- RDF triple store index files (when enabled)
 - *upload/*- temporary directory used during file uploads etc.
 - *webapps/*- location where DSpace installs all Web Applications

7.2.4 Contents of JSPUI Web Application

DSpace's Ant build file creates a *dspace-jspui-webapp/* directory with the following structure:

- (top level dir)
 - The JSPs
 - *WEB-INF/*
 - *web.xml*- DSpace JSPUI Web Application configuration and Servlet mappings
 - *dspace-tags.tld*- DSpace custom tag descriptor
 - *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

7.2.5 Contents of XMLUI Web Application (aka Manakin)

DSpace's Ant build file creates a *dspace-xmlui-webapp/* directory with the following structure:

- (top level dir)
 - *aspects/*- Contains overarching Aspect Generator config and Prototype DRI (Digital Repository Interface) document for Manakin.
 - *i18n/*- Internationalization / Multilingual support. Contains the *messages.xml*/English language pack by default.
 - *themes/*- Contains all out-of-the-box Manakin themes
 - *Classic/*- The classic theme, which makes the XMLUI look like classic DSpace
 - *Kubrick/*- The Kubrick theme
 - *Mirage/*- The Mirage theme (see [Mirage Configuration and Customization](#))
 - *Reference/*- The default reference theme for XMLUI
 - *dri2xhtml/*- The base theme template, which converts XMLUI DRI (Digital Repository Interface) format into XHTML for display. See [XMLUI Base Theme Templates \(dri2xhtml\)](#) for more details.
 - *dri2xhtml-alt/*- The alternative theme template (used by Mirage Theme), which also converts XMLUI DRI (Digital Repository Interface) format into XHTML for display. See [XMLUI Base Theme Templates \(dri2xhtml\)](#) for more details.
 - *template/*- An empty 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

7.2.6 Log Files

The first source of potential confusion is the log files. Since DSpace uses a number of third-party tools, problems can occur in a variety of places. Below is a table listing the main log files used in a typical DSpace setup. The locations given are defaults, and might be different for your system depending on where you installed DSpace and the third-party tools. The ordering of the list is roughly the recommended order for searching them for the details about a particular problem or error.

Log File	What's In It
<i>[dspace]/log/ dspace.log.yyyy-mm-dd</i>	Main DSpace log file. This is where the DSpace code writes a simple log of events and errors that occur within the DSpace code. You can control the verbosity of this by editing the <i>[dspace-source]/config/templates/log4j.properties</i> file and then running " <i>ant init_configs</i> ".
<i>[dspace]/log/ cocoon.log.yyyy-mm-dd</i>	Apache Cocoon log file for the XMLUI. This is where the DSpace XMLUI logs all of its events and errors.
<i>[tomcat]/logs/catalina.out</i>	This is where Tomcat's standard output is written. Many errors that occur within the Tomcat code are logged here. For example, if Tomcat can't find the DSpace code (<i>dspace.jar</i>), it would be logged in <i>catalina.out</i> .
<i>[tomcat]/logs/ hostname_log.yyyy-mm-dd.txt</i>	If you're running Tomcat stand-alone (without Apache), it logs some information and errors for specific Web applications to this log file. <i>hostname</i> will be your host name (e.g. <i>dspace.myu.edu</i>) and <i>yyyy-mm-dd</i> will be the date.
<i>[tomcat]/logs/ apache_log.yyyy-mm-dd.txt</i>	If you're using Apache, Tomcat logs information about Web applications running through Apache (<i>mod_webapp</i>) in this log file (<i>yyyy-mm-dd</i> being the date.)
<i>[apache]/error_log</i>	Apache logs to this file. If there is a problem with getting <i>mod_webapp</i> working, this is a good place to look for clues. Apache also writes to several other log files, though <i>error_log</i> tends to contain the most useful information for tracking down problems.
<i>[dspace]/log/handle-plug.log</i>	The Handle server runs as a separate process from the DSpace Web UI (which runs under Tomcat's JVM). Due to a limitation of log4j's 'rolling file appenders', the DSpace code running in the Handle server's JVM must use a separate log file. The DSpace code that is run as part of a Handle resolution request writes log information to this file. You can control the verbosity of this by editing <i>[dspace-source]/config/templates/log4j-handle-plugin.properties</i> .
<i>[dspace]/log/ handle-server.log</i>	This is the log file for CNRI's Handle server code. If a problem occurs within the Handle server code, before DSpace's plug-in is invoked, this is where it may be logged.
<i>[dspace]/handle-server/ error.log</i>	On the other hand, a problem with CNRI's Handle server code might be logged here.
<i>PostgreSQL log</i>	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*.

log4j.properties File.

the file *[dspace]/config/log4j.properties* controls how and where log files are created. There are three sets of configurations in that file, called A1, A2, and A3. These are used to control the logs for DSpace, the checksum checker, and the XMLUI respectively. The important settings in this file are:

<pre>log4j.rootCategory=INFO,A log4j.logger.org.dspace=INFO,A1</pre>	<p>These lines control what level of logging takes place. Normally they should be set to INFO, but if you need to see more information in the logs, set them to DEBUG and restart your web server</p>
<pre>log4j.appender.A1= org.dspace.app.util.DailyFileAppender</pre>	<p>This is the name of the log file creation method used. The DailyFileAppender creates a new date-stamped file every day or month.</p>
<pre>log4j.appender.A1.File=\${log.dir}/ dspace.log</pre>	<p>This sets the filename and location of where the log file will be stored. It will have a date stamp appended to the file name.</p>
<pre>log4j.appender.A1.DatePattern= yyy-MM-DD</pre>	<p>This defines the format for the date stamp that is appended to the log file names. If you wish to have log files created monthly instead of daily, change this to <i>yyyy-MM</i></p>
<pre>log4j.appender.A1.MaxLogs=0</pre>	<p>This defines how many log files will be created. You may wish to define a retention period for log files. If you set this to 365, logs older than a year will be deleted. By default this is set to 0 so that no logs are ever deleted. Ensure that you monitor the disk space used by the logs to make sure that you have enough space for them. It is often important to keep the log files for a long time in case you want to rebuild your statistics.</p>

7.3 Metadata and Bitstream Format Registries

- 1 [Default Dublin Core Metadata Registry \(DC\)](#)
- 2 [Dublin Core Terms Registry \(DCTERMS\)](#)
- 3 [Default Bitstream Format Registry](#)

7.3.1 Default Dublin Core Metadata Registry (DC)

The default DSpace Dublin Core Metadata Registry was originally derived from the 15 Dublin Core elements. This registry initializes the default schema, where **dc** is used to identify the namespace.

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	author ¹	Author(s) of the work (used by default)
contributor	editor	
contributor	illustrator	
contributor	other	
coverage	spatial	Spatial characteristics of content.
coverage	temporal	Temporal characteristics of content.
creator		May be used as an alternative to "contributor.author"
date		Use qualified form if possible.
date	accessioned ¹	Date DSpace takes possession of item.
date	available ¹	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	issued ¹	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	citation ²	Human-readable, standard bibliographic citation of non-DSpace format of this item

element	qualifier	scope note
identifier	govdoc ²	A government document number
identifier	isbn ²	International Standard Book Number
identifier	issn ²	International Standard Serial Number
identifier	sici	Serial Item and Contribution Identifier
identifier	ismn ²	International Standard Music Number
identifier	other ²	A known identifier type common to a local collection.
identifier	uri ¹	Uniform Resource Identifier
description ¹		Catch-all for any description not defined by qualifiers.
description	abstract ¹	Abstract or summary.
description	provenance ¹	The history of custody of the item since its creation, including any changes successive custodians made to it.
description	sponsorship ²	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 ²		Catch-all for any format information not defined by qualifiers.
format	extent ²	Size or duration.
format	medium ²	Physical medium.
format	mimetype ²	Registered MIME type identifiers.
language		Catch-all for non-ISO forms of the language of the item, accommodating harvested values.
language	iso ²	Current ISO standard for language of intellectual content, including country codes (e.g. "en_US").
publisher ²		Entity responsible for publication, distribution, or imprint.
relation		Catch-all for references to other related items.

element	qualifier	scope note
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 ²		Uncontrolled index term.
subject	classification	Catch-all for value from local classification system. Global classification systems will receive specific qualifier
subject	ddc	Dewey Decimal Classification Number
subject	lcc	Library of Congress Classification Number
subject	lcsb	Library of Congress Subject Headings
subject	mesh	MEdical Subject Headings
subject	other	Local controlled vocabulary; global vocabularies will receive specific qualifier.
title ¹		Title statement/title proper.

element	qualifier	scope note
title	alternative ²	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 several functional areas of DSpace. **DO NOT REMOVE WITHOUT INVESTIGATING THE CONSEQUENCES**

² This field is included in the default DSpace [Submission User Interface](#). Removing this field from your registry will break the default DSpace submission form.

7.3.2 Dublin Core Terms Registry (DCTERMS)

The Dublin Core Terms (DCTERMS) registry was introduced in DSpace 4. This registry initializes an optional metadata schema, where **dcterms** is used to identify the namespace. In DSpace 4, none of these fields are used by any of the system functionality out of the box. The registry and schema were added as a first step to facilitate the future migration of the DSpace specific DC schema, to this schema that complies to current Dublin Core standards.

The main advantage of the DCTERMS schema is that no field name details gets lost during harvesting, as opposed to harvesting of so called "simple" dublin core, where the qualifiers from the above schema are omitted during harvesting.

term	scope note
abstract	A summary of the resource.
accessRights	Information about who can access the resource or an indication of its security status. May include information regarding access or restrictions based on privacy, security, or other policies.
accrualMethod	The method by which items are added to a collection.
accrualPeriodicity	The frequency with which items are added to a collection.
accrualPolicy	The policy governing the addition of items to a collection.
alternative	An alternative name for the resource.
audience	A class of entity for whom the resource is intended or useful.
available	Date (often a range) that the resource became or will become available.
bibliographicCitation	

term	scope note
	Recommended practice is to include sufficient bibliographic detail to identify the resource as unambiguously as possible.
conformsTo	An established standard to which the described resource conforms.
contributor	An entity responsible for making contributions to the resource. Examples of a Contributor include a person, an organization, or a service.
coverage	The spatial or temporal topic of the resource, the spatial applicability of the resource, or the jurisdiction under which the resource is relevant.
created	Date of creation of the resource.
creator	An entity primarily responsible for making the resource.
date	A point or period of time associated with an event in the lifecycle of the resource.
dateAccepted	Date of acceptance of the resource.
dateCopyrighted	Date of copyright.
dateSubmitted	Date of submission of the resource.
description	An account of the resource.
educationLevel	A class of entity, defined in terms of progression through an educational or training context, for which the described resource is intended.
extent	The size or duration of the resource.
format	The file format, physical medium, or dimensions of the resource.
hasFormat	A related resource that is substantially the same as the pre-existing described resource, but in another format.
hasPart	A related resource that is included either physically or logically in the described resource.
hasVersion	A related resource that is a version, edition, or adaptation of the described resource.
identifier	An unambiguous reference to the resource within a given context.
instructionalMethod	A process, used to engender knowledge, attitudes and skills, that the described resource is designed to support.
isFormatOf	A related resource that is substantially the same as the described resource, but in another format.

term	scope note
isPartOf	A related resource in which the described resource is physically or logically included.
isReferencedBy	A related resource that references, cites, or otherwise points to the described resource.
isReplacedBy	A related resource that supplants, displaces, or supersedes the described resource.
isRequiredBy	A related resource that requires the described resource to support its function, delivery, or coherence.
issued	Date of formal issuance (e.g., publication) of the resource.
isVersionOf	A related resource of which the described resource is a version, edition, or adaptation.
language	A language of the resource.
license	A legal document giving official permission to do something with the resource.
mediator	An entity that mediates access to the resource and for whom the resource is intended or useful.
medium	The material or physical carrier of the resource.
modified	Date on which the resource was changed.
provenance	A statement of any changes in ownership and custody of the resource since its creation that are significant for its authenticity, integrity, and interpretation.
publisher	An entity responsible for making the resource available.
references	A related resource that is referenced, cited, or otherwise pointed to by the described resource.
relation	A related resource.
replaces	A related resource that is supplanted, displaced, or superseded by the described resource.
requires	A related resource that is required by the described resource to support its function, delivery, or coherence.
rights	Information about rights held in and over the resource.
rightsHolder	A person or organization owning or managing rights over the resource.
source	A related resource from which the described resource is derived.
spatial	Spatial characteristics of the resource.
subject	The topic of the resource.

term	scope note
tableOfContents	A list of subunits of the resource.
temporal	Temporal characteristics of the resource.
title	A name given to the resource.
type	The nature or genre of the resource.
valid	Date (often a range) of validity of a resource.

7.3.3 Default Bitstream Format Registry

Mimetype	Short Description	Description	Support Level	Internal	Extensions
application/octet-stream¹	Unknown	Unknown data format	Unknown	false	
text/plain¹	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	Broadcast 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

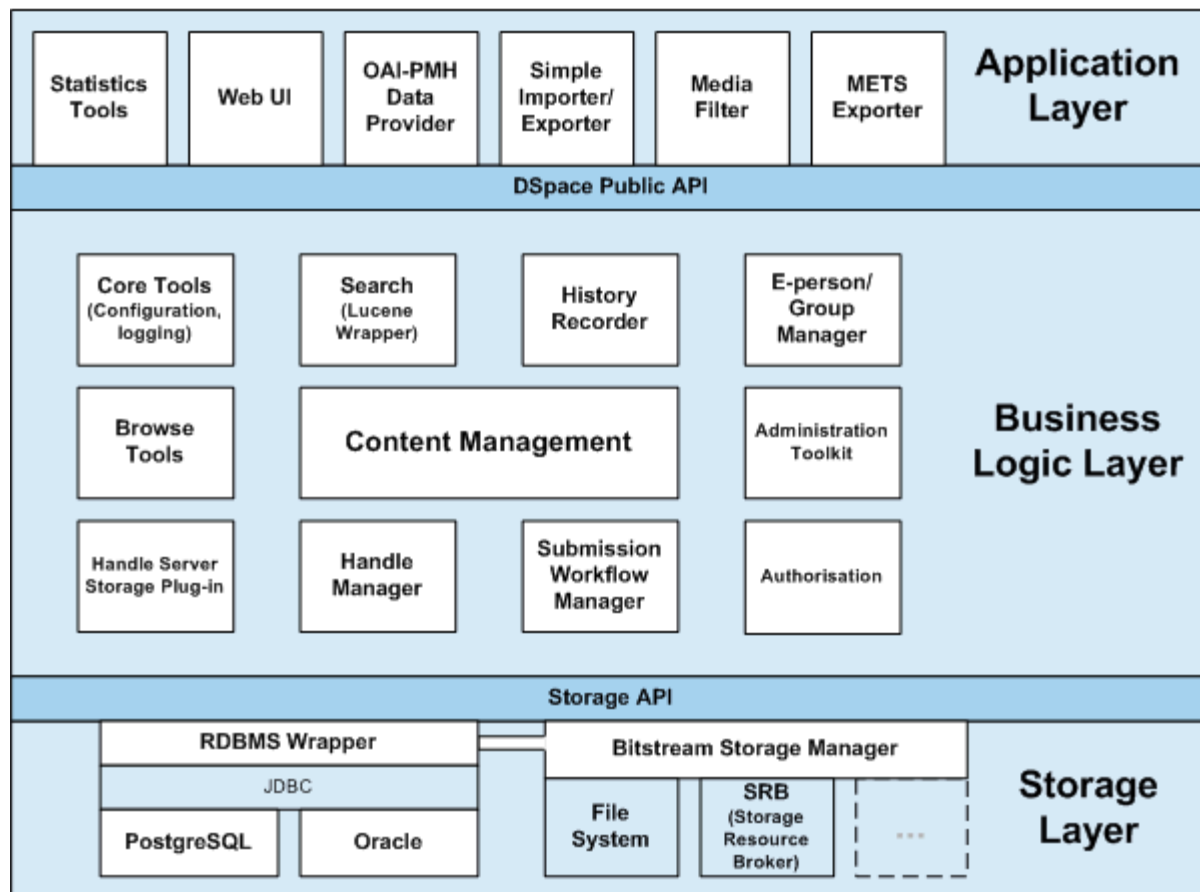
¹ Used by several functional areas of DSpace. **DO NOT REMOVE WITHOUT INVESTIGATING THE CONSEQUENCES**

7.4 Architecture

- 1 [Overview](#)
 - 1.1 [DSpace System Architecture](#)

7.4.1 Overview

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



DSpace System Architecture

The storage layer is responsible for physical storage of metadata and content. The business logic layer deals with managing the content of the archive, users of the archive (e-people), authorization, and workflow. The application layer contains components that communicate with the world outside of the individual DSpace installation, for example the Web user interface and the [Open Archives Initiative](#) protocol for metadata harvesting service.

Each layer only invokes the layer below it; the application layer may not use the storage layer directly, for example. Each component in the storage and business logic layers has a defined public API. The union of the APIs of those components are referred to as the Storage API (in the case of the storage layer) and the DSpace Public API (in the case of the business logic layer). These APIs are in-process Java classes, objects and methods.

It is important to note that each layer is *trusted*. Although the logic for *authorising actions* is in the business logic layer, the system relies on individual applications in the application layer to correctly and securely *authenticate* e-people. If a 'hostile' or insecure application were allowed to invoke the Public API directly, it could very easily perform actions as any e-person in the system.

The reason for this design choice is that authentication methods will vary widely between different applications, so it makes sense to leave the logic and responsibility for that in these applications.

The source code is organized to cohere very strictly to this three-layer architecture. Also, only methods in a component's public API are given the *public* access level. This means that the Java compiler helps ensure that the source code conforms to the architecture.

Packages within	Correspond to components in
<i>org.dspace.app</i>	Application layer
<i>org.dspace</i>	Business logic layer (except <i>storage</i> and <i>app</i>)
<i>org.dspace.storage</i>	Storage layer

The storage and business logic layer APIs are extensively documented with Javadoc-style comments. Generate the HTML version of these by entering the [dspace-source]/dspace directory and running:

```
mvn javadoc:javadoc
```

The resulting documentation will be at *[dspace-source]dspace-api/target/site/apidocs/index.html*. The package-level documentation of each package usually contains an overview of the package and some example usage. This information is not repeated in this architecture document; this and the Javadoc APIs are intended to be used in parallel.

Each layer is described in a separate section:

- [Storage Layer](#)
 - RDBMS
 - Bitstream Store
- [Business Logic Layer](#)
 - Core Classes
 - Content Management API
 - Workflow System

- Administration Toolkit
- E-person/Group Manager
- Authorisation
- Handle Manager/Handle Plugin
- Search
- Browse API
- History Recorder
- Checksum Checker
- [Application Layer](#)
 - Web User Interface
 - OAI-PMH Data Provider
 - Item Importer and Exporter
 - Transferring Items Between DSpace Instances
 - Registration
 - METS Tools
 - Media Filters
 - Sub-Community Management

7.4.2 Application Layer

The following explains how the application layer is built and used.

- 1 [Web User Interface](#)
 - 1.1 [Web UI Files](#)
 - 1.2 [The Build Process](#)
 - 1.3 [Servlets and JSPs \(JSPUI Only\)](#)
 - 1.4 [Custom JSP Tags \(JSPUI Only\)](#)
 - 1.5 [Internationalization \(JSPUI Only\)](#)
 - 1.5.1 [Message Key Convention](#)
 - 1.5.2 [Which Languages are currently supported?](#)
 - 1.6 [HTML Content in Items](#)
 - 1.7 [Thesis Blocking](#)
- 2 [OAI-PMH Data Provider](#)
- 3 [DSpace Command Launcher](#)
 - 3.1 [Older Versions](#)
 - 3.2 [Command Launcher Structure](#)

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 1.5, the deployment has changed. The build systems has moved to a maven-based system enabling the various projects (JSPUI, XMLUI, etc.) into separate projects. The system still uses the familiar 'Ant' to deploy the webapps in later stages.

Location	Description
<i>[dspace-source]/dspace-jspui/dspace-jspui-api/src/main/java/org/dspace/app/webui</i>	Web UI source files
<i>[dspace-source]/dspace-jspui/dspace-jspui-api/src/main/java/org/dspace/app/filters</i>	Servlet Filters (Servlet 2.3 spec)
<i>[dspace-source]/dspace-jspui/dspace-jspui-api/src/main/java/org/dspace/app/jsptag</i>	Custom JSP tag class files
<i>[dspace-source]/dspace-jspui/dspace-jspui-api/src/main/java/org/dspace/app/servlet</i>	Servlets for main Web UI (controllers)
<i>[dspace-source]/dspace-jspui/dspace-jspui-api/src/main/java/org/dspace/app/servlet/admin</i>	Servlets that comprise the administration part of the Web UI
<i>[dspace-source]/dspace-jspui/dspace-jspui-api/src/main/java/org/dspace/app/webui/util/</i>	Miscellaneous classes used by the servlets and filters
<i>[dspace-source]/dspace-jspui</i>	The JSP files
<i>[dspace-source]/dspace/modules/jspui/src/main/webapp</i>	This is where you place customized versions of JSPs, see JSPUI Configuration and Customization
<i>[dspace-source]/dspace/modules/xmlui/src/main/webapp</i>	This is where you place customizations for the Manakin interface, see XMLUI Configuration and Customization
<i>[dspace-source]/dspace/modules/jspui/src/main/resources</i>	This is where you can place you customize version of the <i>Messages.properties</i> file.
<i>[dspace-source]/dspace-jspui/dspace-jspui-webapp/src/main/webapp/WEB-INF/dspace-tags.tld</i>	Custom DSpace JSP tag descriptor

The Build Process

The DSpace Maven build process constructs a full DSpace installation template directory structure containing a series of web applications. The results are placed in `[dspace-source]/dspace/target/dspace-installer/`. The process works as follows:

- All the DSpace source code is compiled, and/or automatically downloaded from the Maven Central code/libraries repository.
- A full DSpace "installation template" folder is built in `[dspace-source]/dspace/target/dspace-installer/`
 - This DSpace "installation template" folder has a structure identical to the [Installed Directory Layout](#)

In order to then install & deploy DSpace from this "installation template" folder, you must run the following from `[dspace-source]/dspace/target/dspace-installer/` :

```
ant -D [dspace]/config/dspace.cfg update
```

Please see the [Installing DSpace](#) instructions for more details about the Installation process.

Servlets and JSPs (JSPUI Only)

The JSPUI 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 *HttpServletRequest* object that represents the HTTP request being processed. This is done by invoking the *setAttribute* method of the *javax.servlet.http.HttpServletRequest* object that is passed into the servlet from Tomcat. The servlet then forwards control of the request to the appropriate JSP using the *JSPManager.showJSP* method.

The *JSPManager.showJSP* method uses the standard Java servlet forwarding mechanism is then used to forward the HTTP request to the JSP. The JSP is processed by Tomcat and the results sent back to the user's browser.

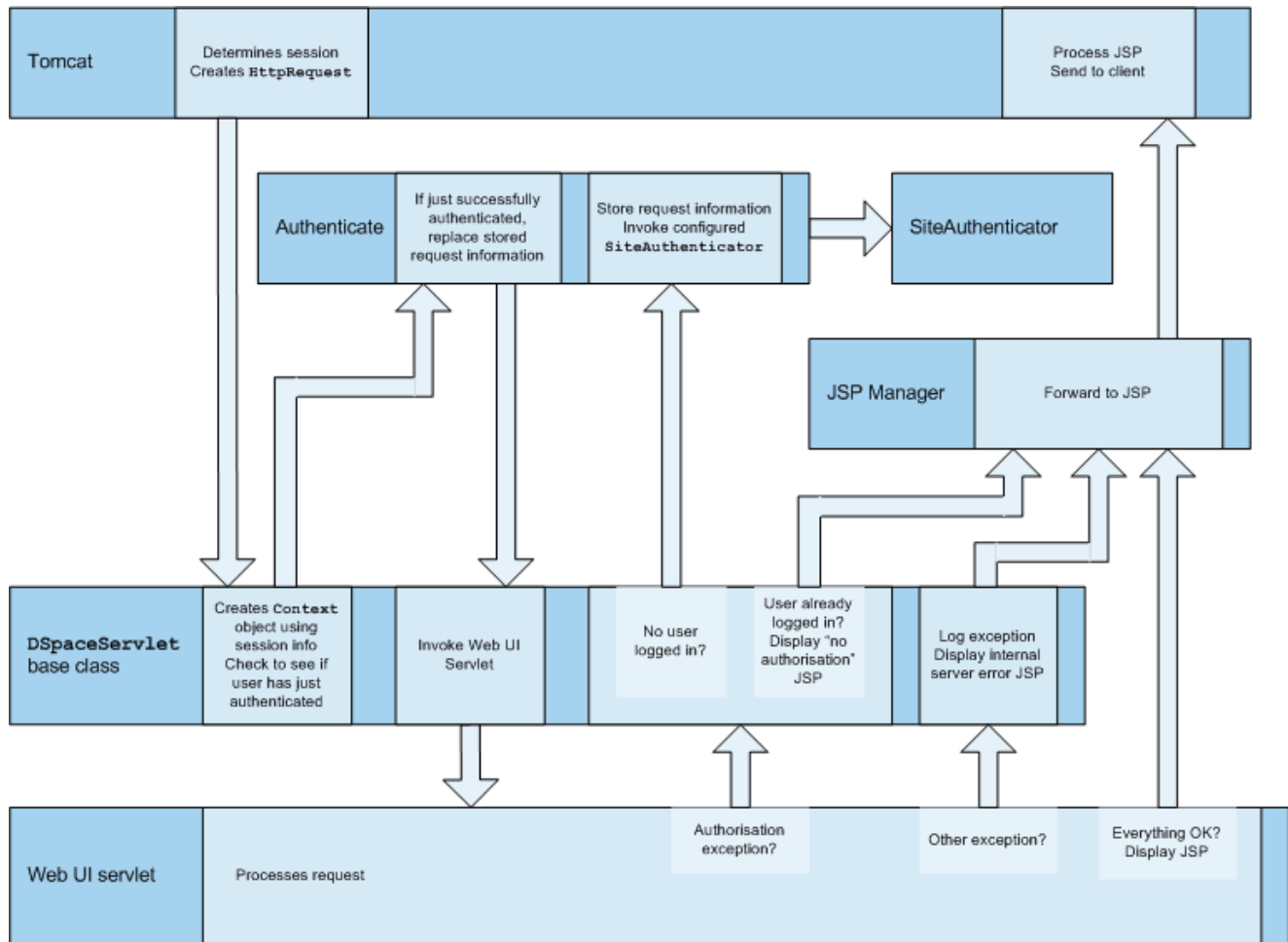
There is an exception to this servlet/JSP style: *index.jsp*, the 'home page', receives the HTTP request directly from Tomcat without a servlet being invoked first. This is because in the servlet 2.3 specification, there is no way to map a servlet to handle only requests made to '/'; such a mapping results in every request being directed to that servlet. By default, Tomcat forwards requests to '/' to *index.jsp*. To try and make things as clean as possible, *index.jsp* contains some simple code that would normally go in a servlet, and then forwards to *home.jsp* using the *JSPManager.showJSP* method. This means localized versions of the 'home page' can be created by placing a customized *home.jsp* in *[dspace-source]/jsp/local*, in the same manner as other JSPs.

[dspace-source]/jsp/dspace-admin/index.jsp, the administration UI index page, is invoked directly by Tomcat and not through a servlet for similar reasons.

At the top of each JSP file, right after the license and copyright header, is documented the appropriate attributes that a servlet must fill out prior to forwarding to that JSP. No validation is performed; if the servlet does not fill out the necessary attributes, it is likely that an internal server error will occur.

Many JSPs containing forms will include hidden parameters that tell the servlets which form has been filled out. The submission UI servlet (*SubmissionController*) is a prime example of a servlet that deals with the input from many different JSPs. The *step* and *page* hidden parameters (written out by the *SubmissionController.getSubmissionParameters()* method) are used to inform the servlet which page of which step has just been filled out (i.e. which page of the submission the user has just completed).

Below is a detailed, scary diagram depicting the flow of control during the whole process of processing and responding to an HTTP request. More information about the authentication mechanism is mostly described in the configuration section.



Flow of Control During HTTP Request Processing

Custom JSP Tags (JSPUI Only)

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.jsp.tag`. 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 (JSPUI Only)



XMLUI Internationalization

For information about XMLUI Internationalization please see: [XMLUI Multilingual Support](#).

The [Java Standard Tag Library v1.0](#) is used to specify messages in the JSPs like this:

OLD:

```
<H1>Search Results</H1>
```

NEW:

```
<H1><fmt:message key="jsp.search.results.title"/></H1>
```

This message can now be changed using the *config/language-packs/Messages.properties* file. (This must be done at build-time: *Messages.properties* is placed in the *dspace.war* Web application file.)

```
jsp.search.results.title = Search Results
```

Phrases may have parameters to be passed in, to make the job of translating easier, reduce the number of 'keys' and to allow translators to make the translated text flow more appropriately for the target language.

OLD:

```
<P>Results <%= r.getFirst() %> to <%= r.getLast() %> of <%=r.getTotal() %></P>
```

NEW:

```
<fmt:message key="jsp.search.results.text">
  <fmt:param><%= r.getFirst() %></fmt:param>
  <fmt:param><%= r.getLast() %></fmt:param>
  <fmt:param><%= r.getTotal() %></fmt:param>
</fmt:message>
```

(Note: JSTL 1.0 does not seem to allow JSP `<%= %>` expressions to be passed in as values of attribute in `<fmt:param value=""/>`)

The above would appear in the *Messages_xx.properties* file as:

```
jsp.search.results.text = Results {0}-{1} of {2}
```

Introducing number parameters that should be formatted according to the locale used makes no difference in the message key compared to string parameters:

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

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

```
<fmt:message key="jsp.submit.show-uploaded-file.size-in-bytes">
  <fmt:param><fmt:formatNumber><%= bitstream.getSize() %></fmt:formatNumber></fmt:param>
</fmt:message>
```

(Note: JSTL offers a way to include numbers in the message keys as *jsp.foo.key = {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](#). e.g. you can add German and Canadian French translations:

```
Messages_de.properties
Messages_fr_CA.properties
```

The end user's browser settings determine which language is used. The English language file *Messages.properties* (or the default server locale) will be used as a default if there's no language bundle for the end user's preferred language. (Note that the English file is not called *Messages_en.properties* – this is so it is always available as a default, regardless of server configuration.)

The *dSPACE:layout* tag has been updated to allow dictionary keys to be passed in for the titles. It now has two new parameters: *titlekey* and *parenttitlekey*. So where before you'd do:

```
<dSPACE:layout title="Here"
  parentlink="/myspace"
  parenttitle="My DSpace">
```

You now do:

```
<dSPACE:layout titlekey="jsp.page.title"
  parentlink="/myspace"
  parenttitlekey="jsp.myspace">
```

And so the layout tag itself gets the relevant stuff out of the dictionary. *title* and *parenttitle* still work as before for backwards compatibility, and the odd spot where that's preferable.

Message Key Convention

When translating further pages, please follow the convention for naming message keys to avoid clashes.

For text in JSPs use the complete path + filename of the JSP, then a one-word name for the message. e.g. for the title of *jsp/myspace/main.jsp* use:

```
jsp.myspace.main.title
```

Some common words (e.g. "Help") can be brought out into keys starting *jsp*. for ease of translation, e.g.:

```
jsp.admin = Administer
```

Other common words/phrases are brought out into 'general' parameters if they relate to a set (directory) of JSPs , e.g.

```
jsp.tools.general.delete = Delete
```

Phrases that relate **strongly** to a topic (eg. MyDSpace) but used in many JSPs outside the particular directory are more convenient to be cross-referenced. For example one could use the key below in *jsp/submit/saved.jsp* to provide a link back to the user's *MyDSpace*:

*(Cross-referencing of keys **in general** is not a good idea as it may make maintenance more difficult. But in some cases it has more advantages as the meaning is obvious.)*

```
jsp.mydspace.general.goto-mydspace = Go to My DSpace
```

For text in servlet code, in custom JSP tags or wherever applicable use the fully qualified classname + a one-word name for the message. e.g.

```
org.dspace.app.webui.jsptag.ItemListTag.title = Title
```

Which Languages are currently supported?

To view translations currently being developed, please refer to the [i18n page](#) 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 constraints. This is usually possible with some kind of batch import. If, however, the document has been uploaded one file at a time using the Web UI, the path information has been stripped. The system can cope with relative links that refer to a deeper path, e.g.

```
<IMG SRC="images/figure1.gif">
```

If the item has been uploaded via the Web submit UI, in the Bitstream table in the database we have the 'name' field, which will contain the filename with no path (*figure1.gif*). We can still work out what *images/figure1.gif* is by making the HTML document servlet strip any path that comes in from the URL, e.g.

```
https://dspace.mit.edu/html/1721.1/12345/images/figure1.gif
          ^^^^^^^
          Strip this
```

BUT all the filenames (regardless of directory names) must be unique. For example, this wouldn't work:

```
contents.html
chapter1.html
chapter2.html
chapter1_images/figure.gif
chapter2_images/figure.gif
```

since the HTML document servlet wouldn't know which bitstream to serve up for:

```
https://dspace.mit.edu/html/1721.1/12345/chapter1_images/figure.gif
https://dspace.mit.edu/html/1721.1/12345/chapter2_images/figure.gif
```

since it would just have *figure.gif*

To prevent "infinite URL spaces" appearing (e.g. if a file *foo.html* linked to *bar/foo.html*, which would link to *bar/bar/foo.html*...) this behavior can be configured by setting the configuration property *webui.html.max-depth-guess*.

For example, if we receive a request for *foo/bar/index.html*, and we have a bitstream called just *index.html*, we will serve up that bitstream for the request if *webui.html.max-depth-guess* is 2 or greater. If *webui.html.max-depth-guess* is 1 or less, we would not serve that bitstream, as the depth of the file is greater. If *webui.html.max-depth-guess* is zero, the request filename and path must always exactly match the bitstream name. The default value (if that property is not present in *dspace.cfg*) is 3.

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

This information now has its own subpage: [OAI-PMH Data Provider 2.0 \(Internals\)](#)

DSpace Command Launcher

Introduced in Release 1.6, the DSpace Command Launcher brings together the various command and scripts into a standard-practice for running CLI runtime programs.

Older Versions

Prior to Release 1.6, there were various scripts written that masked a more manual approach to running CLI programs. The user had to issue *[dspace]/bin/dsrun* and then java class that ran that program. With release 1.5, scripts were written to mask the *[dspace]/bin/dsrun* command. We have left the java class in the System Administration section since it does have value for debugging purposes and for those who wish to learn about DSpace programming or wish to customize the code at any time.

Command Launcher Structure

There are two components to the command launcher: the `dspace` script and the `launcher.xml`. The DSpace command calls a java class which in turn refers to `launcher.xml` that is stored in the `[dspace]/config` directory

`launcher.xml` is made of several components:

- `<command>` begins the stanza for a command
- `<name>_name of command_</name>` the name of the command that you would use.
- `<description>_the description of the command_</description>`
- `<step> </step>` User arguments are parsed and tested.
- `<class>_the java class that is being used to run the CLI program_</class>`

Prior to release 1.5 if one wanted to regenerate the browse index, one would have to issue the following commands manually:

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

In release 1.5 a script was written and in release 1.6 the command `[dspace]/bin/dspace index-init` replaces the script. The stanza from `launcher.xml` show us how one can build more commands if needed:

```
<command>
  <name>index-update</name>
  <description>Update the search and browse indexes</description>
  <step passuserargs="false">
    <class>org.dspace.browse.IndexBrowse</class>
    <argument>-i</argument>
  </step>
  <step passuserargs="false">
    <class>org.dspace.browse.ItemCounter</class>
  </step>
  <step passuserargs="false">
    <class>org.dspace.search.DSIndexer</class>
  </step>
</command>
```

7.4.3 Business Logic Layer

- 1 [Core Classes](#)
 - 1.1 [The Configuration Manager](#)
 - 1.2 [Constants](#)

- [1.3 Context](#)
- [1.4 Email](#)
- [1.5 LogManager](#)
- [1.6 Utils](#)
- [2 Content Management API](#)
 - [2.1 Other Classes](#)
 - [2.2 Modifications](#)
 - [2.3 What's In Memory?](#)
 - [2.4 Dublin Core Metadata](#)
 - [2.5 Support for Other Metadata Schemas](#)
 - [2.6 Packager Plugins](#)
- [3 Plugin Manager](#)
 - [3.1 Concepts](#)
 - [3.2 Using the Plugin Manager](#)
 - [3.2.1 Types of Plugin](#)
 - [3.2.2 Self-Named Plugins](#)
 - [3.2.3 Obtaining a Plugin Instance](#)
 - [3.2.4 Lifecycle Management](#)
 - [3.2.5 Getting Meta-Information](#)
 - [3.3 Implementation](#)
 - [3.3.1 PluginManager Class](#)
 - [3.3.2 SelfNamedPlugin Class](#)
 - [3.3.3 Errors and Exceptions](#)
 - [3.4 Configuring Plugins](#)
 - [3.4.1 Configuring Singleton \(Single\) Plugins](#)
 - [3.4.2 Configuring Sequence of Plugins](#)
 - [3.4.3 Configuring Named Plugins](#)
 - [3.4.4 Configuring the Reusable Status of a Plugin](#)
 - [3.5 Validating the Configuration](#)
 - [3.6 Use Cases](#)
 - [3.6.1 Managing the MediaFilter plugins transparently](#)
 - [3.6.2 A Singleton Plugin](#)
 - [3.6.3 Plugin that Names Itself](#)
 - [3.6.4 Stackable Authentication](#)
- [4 Workflow System](#)
- [5 Administration Toolkit](#)
- [6 E-person/Group Manager](#)
- [7 Authorization](#)
 - [7.1 Special Groups](#)
 - [7.2 Miscellaneous Authorization Notes](#)
- [8 Handle Manager/Handle Plugin](#)
- [9 Search](#)
 - [9.1 Current Lucene Implementation](#)

- 9.2 [Indexed Fields](#)
- 9.3 [Harvesting API](#)
- 10 [Browse API](#)
 - 10.1 [Using the API](#)
 - 10.2 [Index Maintenance](#)
 - 10.3 [Caveats](#)
- 11 [Checksum checker](#)
- 12 [OpenSearch Support](#)
- 13 [Embargo Support](#)
 - 13.1 [What is an Embargo?](#)
 - 13.2 [Embargo Model and Life-Cycle](#)

Core Classes

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

The Configuration Manager

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 Tomcat, you may want to edit the copy in `[dspace-source]` and then run `ant update` or `ant overwrite_configs` rather than editing the 'live' version directly! This will ensure you have a backup copy of your modified configuration files, so that they are not accidentally overwritten in the future.

The *ConfigurationManager* class can also be invoked as a command line tool:

- `[dspace]/bin/dspace dsprop 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. 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*.

Here are a some of the most commonly used constants you might come across:

DSpace types

- Bitstream: 0
- Bundle: 1
- Item: 2
- Collection: 3
- Community: 4
- Site: 5
- Group: 6
- Eperson: 7

DSpace actions

- Read: 0
- Write: 1
- Delete: 2
- Add: 3
- Remove: 4

Refer to the [org.dspace.core.Constants](#) for all of the Constants.

Context

The *Context* class is central to the DSpace operation. Any code that wishes to use the any API in the business logic layer must first create itself a *Context* object. This is akin to opening a connection to a database (which is in fact one of the things that happens.)

A context object is involved in most method calls and object constructors, so that the method or object has access to information about the current operation. When the context object is constructed, the following information is automatically initialized:

- A connection to the database. This is a transaction-safe connection. i.e. the 'auto-commit' flag is set to false.
- A cache of content management API objects. Each time a content object is created (for example *Item* or *Bitstream*) it is stored in the *Context* object. If the object is then requested again, the cached copy is used. Apart from reducing database use, this addresses the problem of having two copies of the same object in memory in different states.

The following information is also held in a context object, though it is the responsibility of the application creating the context object to fill it out correctly:

- The current authenticated user, if any
- Any 'special groups' the user is a member of. For example, a user might automatically be part of a particular group based on the IP address they are accessing DSpace from, even though they don't have an e-person record. Such a group is called a 'special group'.

- Any extra information from the application layer that should be added to log messages that are written within this context. For example, the Web UI adds a session ID, so that when the logs are analyzed 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/log4j.properties`. 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	<i>2002-11-11 08:11:32,903</i>
Level (<i>FATAL</i> , <i>WARN</i> , <i>INFO</i> or <i>DEBUG</i>)	<i>INFO</i>
Java class	<i>org.dspace.app.webui.servlet.DSpaceServlet</i>

	@
User email or <i>anonymous</i>	<i>anonymous</i>
	:
Extra log info from context	<i>session_id=BD84E7C194C2CF4BD0EC3A6CAD0142BB</i>
	:
Action	<i>view_item</i>
	:
Extra info	<i>handle=1721.1/1686</i>

The above format allows the logs to be easily parsed and analyzed. The `[dspace]/bin/log-reporter` script is a simple tool for analyzing 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 methods 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.

Item_s are first created in the form of an implementation of *_InProgressSubmission*. An *InProgressSubmission* represents an item under construction; once it is complete, it is installed into the main archive and added to the relevant collection by the *InstallItem* class. The *org.dspace.content* package provides an implementation of *InProgressSubmission* called *Workspaceltem*, 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 happen in-memory. In these cases, invoking the *update* method lines up the in-memory changes to occur in storage when the *Context* is committed or completed. In general, methods that change any metadata field only make the change in-memory; methods that involve relationships with other objects in the system line up the changes to be committed with the context. See individual methods in the API Javadoc.

Some examples to illustrate this are shown below:

<pre>Context context = new Context(); Bitstream b = Bitstream.find (context, 1234); b.setName("newfile.txt"); b.update(); context.complete();</pre>	<p>Will change storage</p>
<pre>Context context = new Context(); Bitstream b = Bitstream.find (context, 1234); b.setName("newfile.txt"); b.update(); context.abort();</pre>	<p>Will not change storage (context aborted)</p>
<pre>Context context = new Context(); Bitstream b = Bitstream.find (context, 1234); b.setName("newfile.txt");</pre>	<p>The new name will not be stored since <i>update</i> was not invoked</p>

<pre>context.complete();</pre>	
<pre>Context context = new Context(); Bitstream bs = Bitstream.find(context, 1234); Bundle bnd = Bundle.find(context, 5678); bnd.add(bs); context.complete();</pre>	<p>The bitstream will be included in the bundle, since <i>update</i> doesn't need to be called</p>

What's In Memory?

Instantiating some content objects also causes other content objects to be loaded into memory.

Instantiating a *Bitstream* object causes the appropriate *BitstreamFormat* object to be instantiated. Of course the *Bitstream* object does not load the underlying bits from the bitstream store into memory!

Instantiating a *Bundle* object causes the appropriate *Bitstream* objects (and hence *_BitstreamFormat_s*) to be instantiated.

Instantiating an *Item* object causes the appropriate *Bundle* objects (etc.) and hence *_BitstreamFormat_s* to be instantiated. All the Dublin Core metadata associated with that item are also loaded into memory.

The reasoning behind this is that for the vast majority of cases, anyone instantiating an item object is going to need information about the bundles and bitstreams within it, and this methodology allows that to be done in the most efficient way and is simple for the caller. For example, in the Web UI, the servlet (controller) needs to pass information about an item to the viewer (JSP), which needs to have all the information in-memory to display the item without further accesses to the database which may cause errors mid-display.

You do not need to worry about multiple in-memory instantiations of the same object, or any inconsistencies that may result; the *Context* object keeps a cache of the instantiated objects. The *find* methods of classes in *org.dspace.content* will use a cached object if one exists.

It may be that in enough cases this automatic instantiation of contained objects reduces performance in situations where it is important; if this proves to be true the API may be changed in the future to include a *loadContents* method or somesuch, or perhaps a Boolean parameter indicating what to do will be added to the *find* methods.

When a *Context* object is completed, aborted or garbage-collected, any objects instantiated using that context are invalidated and should not be used (in much the same way an AWT button is invalid if the window containing it is destroyed).

Dublin Core Metadata

The *Metadatum* class is a simple container that represents a single Dublin Core-like element, optional qualifier, value and language. Note that since DSpace 1.4 the *MetadataValue* and associated classes are preferred (see Support for Other Metadata Schemas). The other classes starting with *DC* are utility classes for handling types of data in Dublin Core, such as people's names and dates. As supplied, the DSpace registry of elements and qualifiers corresponds to the [Library Application Profile](#) for Dublin Core. It should be noted that these utility classes assume that the values will be in a certain syntax, which will be true for all data generated within the DSpace system, but since Dublin Core does not always define strict syntax, this may not be true for Dublin Core originating outside DSpace.

Below is the specific syntax that DSpace expects various fields to adhere to:

Element	Qualifier	Syntax	Helper Class
<i>date</i>	Any or unqualified	ISO 8601 in the UTC time zone, with either year, month, day, or second precision. Examples: <code>_2000 2002-10 2002-08-14 1999-01-01T14:35:23Z _</code>	<i>DCDate</i>
<i>contributor</i>	Any or unqualified	In general last name, then a comma, then first names, then any additional information like "Jr.". If the contributor is an organization, then simply the name. Examples: <code>_Doe, John Smith, John Jr. van Dyke, Dick Massachusetts Institute of Technology _</code>	<i>DCPersonName</i>
<i>language</i>	<i>iso</i>	A two letter code taken ISO 639, followed optionally by a two letter country code taken from ISO 3166. Examples: <code>_en fr en_US _</code>	<i>DCLanguage</i>
<i>relation</i>	<i>ispartofseries</i>	The series name, following by a semicolon followed by the number in that series. Alternatively, just free text. <code>_MIT-TR; 1234 My Report Series; ABC-1234 NS1234 _</code>	<i>DCSeriesNumber</i>

Support for Other Metadata Schemas

To support additional metadata schemas a new set of metadata classes have been added. These are backwards compatible with the DC classes and should be used rather than the DC specific classes wherever possible. Note that hierarchical metadata schemas are not currently supported, only flat schemas (such as DC) are able to be defined.

The *MetadataField* class describes a metadata field by schema, element and optional qualifier. The value of a *MetadataField* is described by a *MetadataValue* which is roughly equivalent to the older *Metadatum* 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 life cycle 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 implementation of a service, for the entire system, at configuration time. Your application just fetches the plugin for that interface and gets the configured-in choice. See the `getSinglePlugin()` method.
2. **Sequence Plugins** You need a sequence or series of plugins, to implement a mechanism like Stackable Authentication or a pipeline, where each plugin is called in order to contribute its implementation of a process to the whole. The Plugin Manager supports this by letting you configure a sequence of plugins for a given interface. See the `getPluginSequence()` method.
3. **Named Plugins** Use a named plugin when the application has to choose one plugin implementation out of many available ones. Each implementation is bound to one or more names (symbolic identifiers) in the configuration. The name is just a string to be associated with the combination of implementation class and interface. It may contain any characters except for comma (,) and equals (=). It may contain embedded spaces. Comma is a special character used to separate names in the configuration entry. Names must be unique within an interface: No plugin classes implementing the same interface may have the same name. Think of plugin names as a controlled vocabulary – for a given plugin interface, there is a set of names for which plugins can be found. The designer of a Named Plugin interface is responsible for deciding what the name means and how to derive it; for example, names of metadata crosswalk plugins may describe the target metadata format. See the `getNamedPlugin()` method and the `getPluginNames()` methods.

Self-Named Plugins

Named plugins can get their names either from the configuration or, for a variant called self-named plugins, from within the plugin itself.

Self-named plugins are necessary because one plugin implementation can be configured itself to take on many "personalities", each of which deserves its own plugin name. It is already managing its own 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 its own configuration.

An example helps clarify the point: There is a named plugin that does crosswalks, call it *CrosswalkPlugin*. It has several implementations that crosswalk some kind of metadata. Now we add a new plugin which uses XSL stylesheet transformation (XSLT) to crosswalk many types of metadata – so the single plugin can act like many different plugins, depending on which stylesheet it employs.

This XSLT-crosswalk plugin has its own configuration that maps a Plugin Name to a stylesheet – it has to, since of course the Plugin Manager doesn't know anything about stylesheets. It becomes a self-named plugin, so that it reads its configuration data, gets the list of names to which it can respond, and passes those on to the Plugin Manager.

When the Plugin Manager creates an instance of the XSLT-crosswalk, it records the Plugin Name that was responsible for that instance. The plugin can look at that Name later in order to configure itself correctly for the Name that created it. This mechanism is all part of the *SelfNamedPlugin* class which is part of any self-named plugin.

Obtaining a Plugin Instance

The most common thing you will do with the Plugin Manager is obtain an instance of a plugin. To request a plugin, you must always specify the plugin interface you want. You will also supply a name when asking for a named plugin.

A sequence plugin is returned as an array of `_Object_s` since it is actually an ordered list of plugins.

See the `getSinglePlugin()`, `getPluginSequence()`, `getNamedPlugin()` methods.

Lifecycle Management

When *PluginManager* fulfills a request for a plugin, it checks whether the implementation class is reusable; if so, it creates one instance of that class and returns it for every subsequent request for that interface and name. If it is not reusable, a new instance is always created.

For reasons that will become clear later, the manager actually caches a separate instance of an implementation class for each name under which it can be requested.

You can ask the *PluginManager* to forget about (decache) a plugin instance, by releasing it. See the `PluginManager.releasePlugin()` method. The manager will drop its reference to the plugin so the garbage collector can reclaim it. The next time that plugin/name combination is requested, it will create a new instance.

Getting Meta-Information

The *PluginManager* can list all the names of the Named Plugins which implement an interface. You may need this, for example, to implement a menu in a user interface that presents a choice among all possible plugins. See the `getPluginNames()` method.

Note that it only returns the plugin name, so if you need a more sophisticated or meaningful "label" (i.e. a key into the I18N message catalog) then you should add a method to the plugin itself to return that.

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 interface)`
`throws PluginConfigurationException;`

Returns an instance of the singleton (single) plugin implementing the given interface. There must be exactly one single plugin configured for this interface, otherwise the *PluginConfigurationException* is thrown. Note that this is the only "get plugin" method which throws an exception. It is typically used at initialization time to set up a permanent part of the system so any failure is fatal. See the *plugin.single* configuration key for configuration details.

- `static Object[] getPluginSequence(Class interface);`

Returns instances of all plugins that implement the interface *interface*, 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 interface, String name);`

Returns an instance of a plugin that implements the interface *interface* 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 interface);
```

Returns all of the names under which a named plugin implementing the interface *interface* 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.SimpleDispatcher
```

Configuring Sequence of Plugins

This kind of configuration entry defines a Sequence Plugin, which is bound to a sequence of implementation classes. The key identifies the interface, and the value is a comma-separated list of classnames:

plugin.sequence.interface = classname, ...

The plugins are returned by *getPluginSequence()* in the same order as their classes are listed in the configuration value.

For example, this entry configures Stackable Authentication with three implementation classes:

```
plugin.sequence.org.dspace.eperson.AuthenticationMethod = \
    org.dspace.eperson.X509Authentication, \
    org.dspace.eperson.PasswordAuthentication, \
    edu.mit.dspace.MITSpecialGroup
```

Configuring Named Plugins

There are two ways of configuring named plugins:

1. **Plugins Named in the Configuration** A named plugin which gets its name(s) from the configuration is listed in this kind of entry: `_plugin.named.interface = classname = name [, name..] [classname = name..]` The syntax of the configuration value is: classname, followed by an equal-sign and then at least one plugin name. Bind more names to the same implementation class by adding them here, separated by commas. Names may include any character other than comma (,) and equal-sign (=). For example, this entry creates one plugin with the names GIF, JPEG, and image/png, and another with the name TeX:

```
plugin.named.org.dspace.app.mediafilter.MediaFilter = \
    org.dspace.app.mediafilter.JPEGFilter = GIF, JPEG, image/png \
    org.dspace.app.mediafilter.TeXFilter = TeX
```

This example shows a plugin name with an embedded whitespace character. Since comma (,) is the separator character between plugin names, spaces are legal (between words of a name; leading and trailing spaces are ignored). This plugin is bound to the names "Adobe PDF", "PDF", and "Portable Document Format".

```
plugin.named.org.dspace.app.mediafilter.MediaFilter = \
    org.dspace.app.mediafilter.TeXFilter = TeX \
    org.dspace.app.mediafilter.PDFFilter = Adobe PDF, PDF, Portable Document Format
```

NOTE: Since there can only be one key with `plugin.named.` followed by the interface name in the configuration, all of the plugin implementations must be configured in that entry.

2. **Self-Named Plugins** Since a self-named plugin supplies its own names through a static method call, the configuration only has to include its interface and classname: `plugin.selfnamed.interface = classname [, classname..]` The following example first demonstrates how the plugin class, `XsltDisseminationCrosswalk` is configured to implement its own names "MODS" and "DublinCore". These come from the keys starting with `crosswalk.dissemination.stylesheet.`. The value is a stylesheet file. The class is then configured as a self-named plugin:

```
crosswalk.dissemination.stylesheet.DublinCore = xwalk/TESTDIM-2-DC_copy.xml
crosswalk.dissemination.stylesheet.MODS = xwalk/mods.xml
plugin.selfnamed.crosswalk.org.dspace.content.metadata.DisseminationCrosswalk = \
    org.dspace.content.metadata.MODSDisseminationCrosswalk, \
    org.dspace.content.metadata.XsltDisseminationCrosswalk
```

NOTE: Since there can only be one key with *plugin.selfnamed*, followed by the interface name in the configuration, all of the plugin implementations must be configured in that entry. The *MODSDisseminationCrosswalk* class is only shown to illustrate this point.

Configuring the Reusable Status of a Plugin

Plugins are assumed to be reusable by default, so you only need to configure the ones which you would prefer not to be reusable. The format is as follows:

```
plugin.reusable.classname = ( true | false )
```

For example, this marks the PDF plugin from the example above as non-reusable:

```
plugin.reusable.org.dspace.app.mediafilter.PDFFilter = false
```

Validating the Configuration

The Plugin Manager is very sensitive to mistakes in the DSpace configuration. Subtle errors can have unexpected consequences that are hard to detect: for example, if there are two "plugin.single" entries for the same interface, one of them will be silently ignored.

To validate the Plugin Manager configuration, call the *PluginManager.checkConfiguration()* method. It looks for the following mistakes:

- Any duplicate keys starting with "plugin".
- Keys starting *plugin.single*, *plugin.sequence*, *plugin.named*, and *plugin.selfnamed* that don't include a valid interface.
- Classnames in the configuration values that don't exist, or don't implement the plugin interface in the key.
- Classes declared in *plugin.selfnamed* lines that don't extend the *SelfNamedPlugin* class.
- Any name collisions among named plugins for a given interface.
- Named plugin configuration entries without any names.
- Classnames mentioned in *plugin.reusable* keys must exist and have been configured as a plugin implementation class.

The *PluginManager* class also has a *main()* method which simply runs *checkConfiguration()*, so you can invoke it from the command line to test the validity of plugin configuration changes.

Eventually, someone should develop a general configuration-file sanity checker for DSpace, which would just call *PluginManager.checkConfiguration()*.

Use Cases

Here are some usage examples to illustrate how the Plugin Manager works.

Managing the MediaFilter plugins transparently

The existing DSpace 1.3 MediaFilterManager implementation has been largely replaced by the Plugin Manager. The MediaFilter classes become plugins named in the configuration. Refer to the configuration guide for further details.

A Singleton Plugin

This shows how to configure and access a single anonymous plugin, such as the BitstreamDispatcher plugin:

Configuration:

```
plugin.single.org.dspace.checker.BitstreamDispatcher=org.dspace.checker.SimpleDispatcher
```

The following code fragment shows how dispatcher, the service object, is initialized and used:

```
BitstreamDispatcher dispatcher =
    (BitstreamDispatcher)PluginManager.getSinglePlugin(BitstreamDispatcher
.class);
int id = dispatcher.next();
while (id != BitstreamDispatcher.SENTINEL)
{
    /*
     * do some processing here
     */
    id = dispatcher.next();
}
```

Plugin that Names Itself

This crosswalk plugin acts like many different plugins since it is configured with different XSL translation stylesheets. Since it already gets each of its stylesheets out of the DSpace configuration, it makes sense to have the plugin give PluginManager the names to which it answers instead of forcing someone to configure those names in two places (and try to keep them synchronized).

NOTE: Remember how *getPlugin()* caches a separate instance of an implementation class for every name bound to it? This is why: the instance can look at the name under which it was invoked and configure itself specifically for that name. Since the instance for each name might be different, the Plugin Manager has to cache a separate instance for each name.

Here is the configuration file listing both the plugin's own configuration and the *PluginManager* config line:

```
crosswalk.dissemination.stylesheet.DublinCore = xwalk/TESTDIM-2-DC_copy.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 Section on Stackable Authentication for further details.

Workflow System

The primary classes are:

<i>org.dspace.content.WorkspaceItem</i>	contains an Item before it enters a workflow
<i>org.dspace.workflow.WorkflowItem</i>	contains an Item while in a workflow
<i>org.dspace.workflow.WorkflowManager</i>	responds to events, manages the WorkflowItem states
<i>org.dspace.content.Collection</i>	contains List of defined workflow steps

<i>org.dspace.eperson.Group</i>	people who can perform workflow tasks are defined in EPerson Groups
<i>org.dspace.core.Email</i>	used to email messages to Group members and submitters

The workflow system models the states of an Item in a state machine with 5 states (SUBMIT, STEP_1, STEP_2, STEP_3, ARCHIVE.) These are the three optional steps where the item can be viewed and corrected by different groups of people. Actually, it's more like 8 states, with STEP_1_POOL, STEP_2_POOL, and STEP_3_POOL. These pooled states are when items are waiting to enter the primary states.

The WorkflowManager is invoked by events. While an Item is being submitted, it is held by a Workspaceltem. Calling the start() method in the WorkflowManager converts a Workspaceltem 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 Workspaceltem 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/dspace 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.Metadatum* class. Only administrators may modify the Dublin Core type registry.

The *org.dspace.administer.RegistryLoader* class contains methods for initializing 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 *EPerson* 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:

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

Three new attributes have been introduced in the *ResourcePolicy* class as part of the DSpace 3.0 [Embargo Contribution](#):

- *rpname*: resource policy name
- *rptype*: resource policy type
- *rpdescription*: resource policy description

While *rpname* and *rpdescription* are fields manageable by the users the *_rptype* is a field managed by the system. It represents a type that a resource policy can assume between the following:

- *TYPE_SUBMISSION*: all the policies added automatically during the submission process
- *TYPE_WORKFLOW*: all the policies added automatically during the workflow stage
- *TYPE_CUSTOM*: all the custom policies added by users
- *TYPE_INHERITED*: all the policies inherited by the *DSO* father.

An custom policy, created for the purpose of creating an embargo could look like:

```
policy_id: 4847
resource_type_id: 2
resource_id: 89
action_id: 0
eperson_id:
epersongroup_id: 0
start_date: 2013-01-01
end_date:
rpname: Embargo Policy
```

```
rpdescription: Embargoed through 2012
rptype: TYPE_CUSTOM
```

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 their collection's read policy. There once was a separate item read default policy in each collection, and perhaps there will be again since it appears that administrators are notoriously bad at defining collection's read policies. There is also code in place to enable policies that are timed , have a start and end date. However, the admin tools to enable these sorts of policies have not been written.

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 located at `[dspace]/config/log4j-handle-plugin.properties`. 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 can be invoked by the `dspace/bin/index-init` (complete rebuild) or `dspace/bin/index-update` (update) script. The intent was for the *main()* method to be invoked on a regular basis to avoid index corruption, but we have had no problem with that so far.

Which fields are indexed by *DSIndexer*? These fields are defined in `dspace.cfg` in the section "Fields to index for search" as name-value-pairs. The name must be unique in the form `search.index.i` (i is an arbitrary positive number). The value on the right side has a unique value again, which can be referenced in search-form (e.g. title, author). Then comes the metadata element which is indexed. '*' is a wildcard which includes all sub elements. 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.

Current Lucene Implementation

Currently we have our own Analyzer and Tokenizer classes (*DSAnalyzer* and *DSTokenizer*) to customize our indexing. They invoke the stemming and stop word features within Lucene. We create an *IndexReader* for each query, which we now realize isn't the most efficient use of resources - we seem to run out of filehandles on really heavy loads. (A wildcard query can open many filehandles!) Since Lucene is thread-safe, a better future implementation would be to have a single Lucene IndexReader shared by all queries, and then is invalidated and re-opened when the index changes. Future API growth could include relevance scores (Lucene generates them, but we ignore them,) and abstractions for more advanced search concepts such as booleans.

Indexed Fields

The *DSIndexer* class shipped with DSpace indexes the Dublin Core metadata in the following way:

Search Field	Taken from Dublin Core Fields
Authors	<i>contributor.creator.description.statementofresponsibility</i>
Titles	<i>title.*</i>
Keywords	<i>subject.*</i>
Abstracts	<i>description.abstractdescription.tableofcontents</i>
Series	<i>relation.ispartofseries</i>
MIME types	<i>format.mimetype</i>
Sponsors	<i>description.sponsorship</i>
Identifiers	<i>identifier.*</i>

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 indexes 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" would appear 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 catalogers.
- **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 same issue date. Say 30 items in a collection have the issue date 2002. To be able to page through the index 20 items at a time, you need to be able to specify exactly which item's 2002 is the focus of the browse, otherwise each time you invoked the browse code, the results would start at the first item with the issue date 2002.

Author browses return *String* objects with the actual author names. You can only specify the focus as a full or partial literal *String*.

Another important point to note is that presently, the browse indexes 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 indexes from scratch. In general the content management API invokes the necessary browse API calls to keep the browse indexes in sync with what is in the archive, so most applications will not need to invoke those methods.

If the browse index becomes inconsistent for some reason, the *InitializeBrowse* class is a command line tool (generally invoked using the `[dspace]/bin/dspace index-init` command) that causes the indexes to be regenerated from scratch.

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.

Checksum checker

Checksum checker is used to verify every item within DSpace. While DSpace calculates and records the checksum of every file submitted to it, the checker can determine whether the file has been changed. The idea being that the earlier you can identify a file has changed, the more likely you would be able to record it (assuming it was not a wanted change).

`org.dspace.checker.CheckerCommand` class, is the class for the checksum checker tool, which calculates checksums for each bitstream whose ID is in the *most_recent_checksum* table, and compares it against the last calculated checksum for that bitstream.

OpenSearch Support

DSpace is able to support OpenSearch. For those not acquainted with the standard, a very brief introduction, with emphasis on what possibilities it holds for current use and future development.

OpenSearch is a small set of conventions and documents for describing and using 'search engines', meaning any service that returns a set of results for a query. It is nearly ubiquitous, but also nearly invisible, in modern web sites with search capability. If you look at the page source of Wikipedia, Facebook, CNN, etc you will find

buried a link element declaring OpenSearch support. It is very much a lowest-common-denominator abstraction (think Google box), but does provide a means to extend its expressive power. This first implementation for DSpace supports *none* of these extensions, many of which are of potential value, so it should be regarded as a foundation, not a finished solution. So the short answer is that DSpace appears as a 'search-engine' to OpenSearch-aware software.

Another way to look at OpenSearch is as a RESTful web service for search, very much like SRW/U, but considerably simpler. This comparative loss of power is offset by the fact that it is widely supported by web tools and players: browsers understand it, as do large metasearch tools.

How Can It Be Used

- **Browser Integration** Many recent browsers (IE7+, FF2+) can detect, or 'autodiscover', links to the document describing the search engine. Thus you can easily add your or other DSpace instances to the drop-down list of search engines in your browser. This list typically appears in the upper right corner of the browser, with a search box. In Firefox, for example, when you visit a site supporting OpenSearch, the color of the drop-down list widget changes color, and if you open it to show the list of search engines, you are offered an opportunity to add the site to the list. IE works nearly the same way but instead labels the web sites 'search providers'. When you select a DSpace instance as the search engine and enter a search, you are simply sent to the regular search results page of the instance.
- **Flexible, interesting RSS Feeds** Because one of the formats that OpenSearch specifies for its results is RSS (or Atom), you can turn any search query into an RSS feed. So if there are keywords highly discriminative of content in a collection or repository, these can be turned into a URL that a feed reader can subscribe to. Taken to the extreme, one could take any search a user makes, and dynamically compose an RSS feed URL for it in the page of returned results. To see an example, if you have a DSpace with OpenSearch enabled, try:

```
http://dspace.mysite.edu/open-search/?query=<your query>
```

The default format returned is Atom 1.0, so you should see an Atom document containing your search results.

- You can extend the syntax with a few other parameters, as follows:

Parameter	Values
format	atom, rss, html
scope	handle of a collection or community to restrict the search to
rpp	number indicating the number of results per page (i.e. per request)
start	number of page to start with (if paginating results)
sort_by	number indicating sorting criteria (same as DSpace advanced search values)

Multiple parameters may be specified on the query string, using the "&" character as the delimiter, e.g.:

```
http://dspace.mysite.edu/open-search/?query=<your query>&format=rss&scope=123456789/1
```

- Cheap metasearch aggregators like A9 (Amazon) recognize OpenSearch-compliant providers, and so can be added to metasearch sets using their UIs. Then your site can be used to aggregate search results with others.

Configuration is through the `dspace.cfg` file. See [OpenSearch Support](#) for more details.

Embargo Support

What is an Embargo?

An embargo is a temporary access restriction placed on content, commencing at time of accession. Its scope or duration may vary, but the fact that it eventually expires is what distinguishes it from other content restrictions. For example, it is not unusual for content destined for DSpace to come with permanent restrictions on use or access based on license-driven or other IP-based requirements that limit access to institutionally affiliated users. Restrictions such as these are imposed and managed using standard administrative tools in DSpace, typically by attaching specific policies to Items or Collections, Bitstreams, etc. The embargo functionality introduced in 1.6, however, includes tools to automate the imposition and removal of restrictions in managed timeframes.

Embargo Model and Life-Cycle

Functionally, the embargo system allows you to attach 'terms' to an item before it is placed into the repository, which express how the embargo should be applied. What do 'we mean by terms' here? They are really any expression that the system is capable of turning into (1) the time the embargo expires, and (2) a concrete set of access restrictions. Some examples:

"2020-09-12" - an absolute date (i.e. the date embargo will be lifted)
"6 months" - a time relative to when the item is accessioned
"forever" - an indefinite, or open-ended embargo
"local only until 2015" - both a time and an exception (public has no access until 2015, local users OK immediately)
"Nature Publishing Group standard" - look-up to a policy somewhere (typically 6 months)

These terms are 'interpreted' by the embargo system to yield a specific date on which the embargo can be removed or 'lifted', and a specific set of access policies. Obviously, some terms are easier to interpret than others (the absolute date really requires none at all), and the 'default' embargo logic understands only the most basic terms (the first and third examples above). But as we will see below, the embargo system provides you with the ability to add in your own 'interpreters' to cope with any terms expressions you wish to have. This date that is the result of the interpretation is stored with the item and the embargo system detects when that date has passed, and removes the embargo ("lifts it"), so the item bitstreams become available. Here is a more detailed life-cycle for an embargoed item:

1. **Terms Assignment.** The first step in placing an embargo on an item is to attach (assign) 'terms' to it. If these terms are missing, no embargo will be imposed. As we will see below, terms are carried in a configurable DSpace metadata field, so assigning terms just means assigning a value to a metadata field. This can be done in a web submission user interface form, in a SWORD deposit package, a batch import,

etc. - anywhere metadata is passed to DSpace. The terms are not immediately acted upon, and may be revised, corrected, removed, etc, up until the next stage of the life-cycle. Thus a submitter could enter one value, and a collection editor replace it, and only the last value will be used. Since metadata fields are multivalued, theoretically there can be multiple terms values, but in the default implementation only one is recognized.

2. **Terms interpretation/imposition.** In DSpace terminology, when an item has exited the last of any workflow steps (or if none have been defined for it), it is said to be 'installed' into the repository. At this precise time, the 'interpretation' of the terms occurs, and a computed 'lift date' is assigned, which like the terms is recorded in a configurable metadata field. It is important to understand that this interpretation happens only once, (just like the installation), and cannot be revisited later. Thus, although an administrator can assign a new value to the metadata field holding the terms after the item has been installed, this will have no effect on the embargo, whose 'force' now resides entirely in the 'lift date' value. For this reason, you cannot embargo content already in your repository (at least using standard tools). The other action taken at installation time is the actual imposition of the embargo. The default behavior here is simply to remove the read policies on all the bundles and bitstreams except for the "LICENSE" or "METADATA" bundles. See the section on *Extending Embargo Functionality* for how to alter this behavior. Also note that since these policy changes occur before installation, there is no time during which embargoed content is 'exposed' (accessible by non-administrators). The terms interpretation and imposition together are called 'setting' the embargo, and the component that performs them both is called the embargo 'setter'.
3. **Embargo Period.** After an embargoed item has been installed, the policy restrictions remain in effect until removed. This is not an automatic process, however: a 'lifter' must be run periodically to look for items whose 'lift date' is past. Note that this means the effective removal of an embargo is **not** the lift date, but the earliest date after the lift date that the lifter is run. Typically, a nightly cron-scheduled invocation of the lifter is more than adequate, given the granularity of embargo terms. Also note that during the embargo period, all metadata of the item remains visible. This default behavior can be changed. One final point to note is that the 'lift date', although it was computed and assigned during the previous stage, is in the end a regular metadata field. That means, if there are extraordinary circumstances that require an administrator (or collection editor, anyone with edit permissions on metadata) to change the lift date, they can do so. Thus, they can 'revise' the lift date without reference to the original terms. This date will be checked the next time the 'lifter' is run. One could immediately lift the embargo by setting the lift date to the current day, or change it to 'forever' to indefinitely postpone lifting.
4. **Embargo Lift.** When the lifter discovers an item whose lift date is in the past, it removes (lifts) the embargo. The default behavior of the lifter is to add the resource policies *that would have been added* had the embargo not been imposed. That is, it replicates the standard DSpace behavior, in which an item inherits its policies from its owning collection. As with all other parts of the embargo system, you may replace or extend the default behavior of the lifter (see section V. below). You may wish, e.g. to send an email to an administrator or other interested parties, when an embargoed item becomes available.
5. **Post Embargo.** After the embargo has been lifted, the item ceases to respond to any of the embargo life-cycle events. The values of the metadata fields reflect essentially historical or provenance values. With the exception of the additional metadata fields, they are indistinguishable from items that were never subject to embargo.



More Embargo Details

More details on Embargo configuration, including specific examples can be found in the [Embargo](#) section of the documentation.

7.4.4 DSpace Services Framework

- 1 [Architectural Overview](#)
 - 1.1 [DSpace Kernel](#)
 - 1.1.1 [Kernel registration](#)
 - 1.2 [Service Manager](#)
- 2 [Basic Usage](#)
 - 2.1 [Standalone Applications](#)
 - 2.2 [Application Frameworks \(Spring, Guice, etc.\)](#)
 - 2.3 [Web Applications](#)
- 3 [Providers and Plugins](#)
 - 3.1 [Activators](#)
 - 3.2 [Provider Stacks](#)
- 4 [Core Services](#)
 - 4.1 [Caching Service](#)
 - 4.2 [Configuration Service](#)
 - 4.3 [EventService](#)
 - 4.4 [RequestService](#)
 - 4.5 [SessionService](#)
- 5 [Examples](#)
 - 5.1 [Configuring Event Listeners](#)
- 6 [Tutorials](#)

The DSpace Services Framework is a backporting of the DSpace 2.0 Development Group's work in creating a reasonable and abstractable "Core Services" layer for DSpace components to operate within. The Services Framework represents a "best practice" for new DSpace architecture and implementation of extensions to the DSpace application. DSpace Services are best described as a "Simple Registry" where plugins can be "looked up" or located. The DS2 ([DSpace 2.0](#)) core services are the main services that make up a DS2 system. These includes services for things like user and permissions management and storage and caching. These services can be used by any developer writing DS2 plugins (e.g. statistics), providers (e.g. authentication), or user interfaces (e.g. JSPUI).

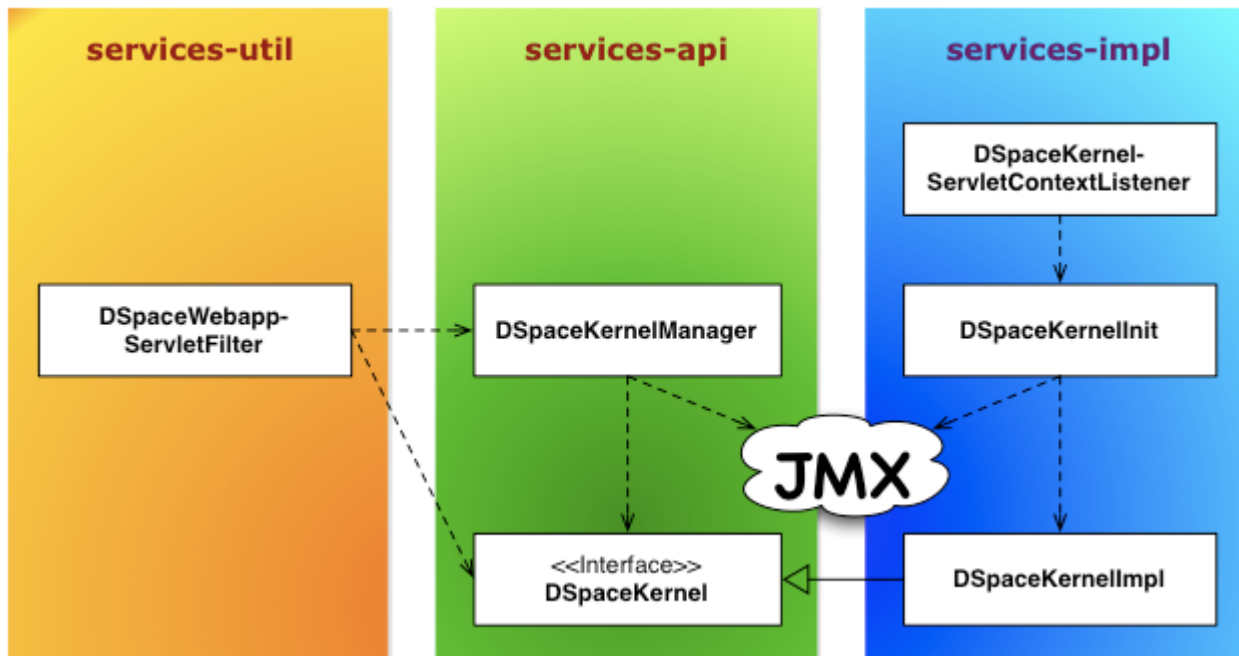
Architectural Overview

DSpace Kernel

The DSpace Kernel manages the start up and access services in the DSpace Services framework. It is meant to allow for a simple way to control the core parts of DSpace and allow for flexible ways to startup the kernel. For example, the kernel can be run inside a single webapp along with a frontend piece (like JSPUI) or it can be started as part of the servlet container so that multiple webapps can use a single kernel (this increases speed and efficiency). The kernel is also designed to happily allow multiple kernels to run in a single servlet container using identifier keys.

Kernel registration

The kernel will automatically register itself as an MBean when it starts up so that it can be managed via [JMX](#). It allows startup and shutdown and provides direct access to the ServiceManager and the ConfigurationService. All the other core services can be retrieved from the ServiceManager by their APIs.

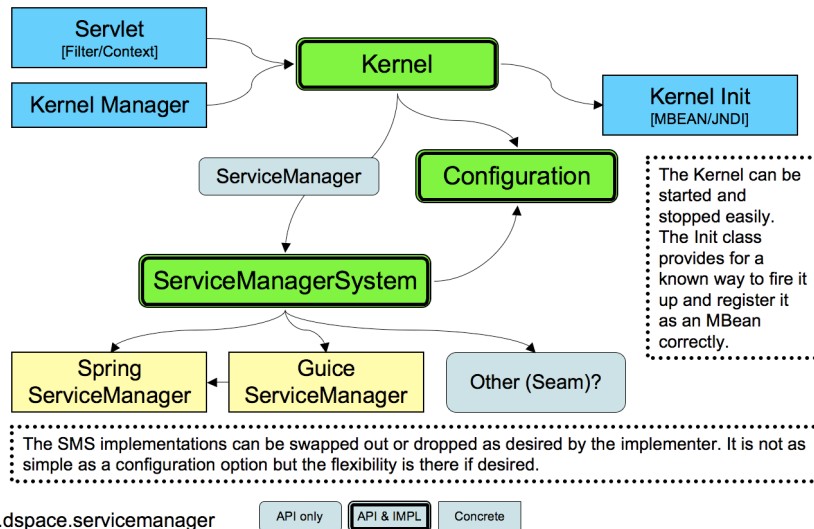


Service Manager

The ServiceManager abstracts the concepts of service lookups and lifecycle control. It also manages the configuration of services by allowing properties to be pushed into the services as they start up (mostly from the ConfigurationService). The ServiceManagerSystem abstraction allows the DSpace ServiceManager to use different systems to manage its services. The current implementations include Spring and Guice. This allows DSpace 2 to have very little service management code but still be flexible and not tied to specific technology. Developers who are comfortable with those technologies can consume the services from a parent Spring ApplicationContext or a parent Guice Module. The abstraction also means that we can replace Spring/Guice or add other dependency injection systems later without requiring developers to change their code. The interface

provides simple methods for looking up services by interface type for developers who do not want to have to use or learn a dependency injection system or are using one which is not currently supported.

DSpace 2 Service Manager (DS2 Kernel / Service Manager)



The DS2 kernel is compact so it can be completely started up in a unit test (technically integration test) environment. (This is how we test the kernel and core services currently). This allows developers to execute code against a fully functional kernel while developing and then deploy their code with high confidence.

Basic Usage

To use the Framework you must begin by instantiating and starting a DSpaceKernel. The kernel will give you references to the ServiceManager and the ConfigurationService. The ServiceManager can be used to get references to other services and to register services which are not part of the core set.

Access to the kernel is provided via the Kernel Manager through the DSpace object, which will locate the kernel object and allow it to be used.

Standalone Applications

For standalone applications, access to the kernel is provided via the Kernel Manager and the DSpace object which will locate the kernel object and allow it to be used.

```

/* Instantiate the Utility Class */
DSpace dspace = new DSpace();
/* Access get the Service Manager by convenience method */
ServiceManager manager = dspace.getServiceManager();
/* Or access by convenience method for core services */
EventService service = dspace.getEventService();
    
```

The DSpace launcher (

```
bin/dspace
```

) initializes a kernel before dispatching to the selected command.

Application Frameworks (Spring, Guice, etc.)

Similar to [Standalone Applications](#), but you can use your framework to instantiate an `org.dspace.utils.DSpace` object.

```
<bean id="dspace" class="org.dspace.utils.DSpace" />
```

Web Applications

In web applications, the kernel can be started and accessed through the use of Servlet Filter/ContextListeners which are provided as part of the DSpace 2 utilities. Developers don't need to understand what is going on behind the scenes and can simply write their applications and package them as webapps and take advantage of the services which are offered by DSpace 2.

Providers and Plugins

For developers (how we are trying to make your lives easier): The DS2 ServiceManager supports a plugin/provider system which is runtime hot-swappable. The implementor can register any service/provider bean or class with the DS2 kernel ServiceManager. The ServiceManager will manage the lifecycle of beans (if desired) and will instantiate and manage the lifecycle of any classes it is given. This can be done at any time and does not have to be done during Kernel startup. This allows providers to be swapped out at runtime without disrupting the service if desired. The goal of this system is to allow DS2 to be extended without requiring any changes to the core codebase or a rebuild of the code code.

Activators

Developers can provide an activator to allow the system to startup their service or provider. It is a simple interface with 2 methods which are called by the ServiceManager to startup the provider(s) and later to shut them down. These simply allow a developer to run some arbitrary code in order to create and register services if desired. It is the method provided to add plugins directly to the system via configuration as the activators are just listed in the configuration file and the system starts them up in the order it finds them.

Provider Stacks

Utilities are provided to assist with stacking and ordering providers. Ordering is handled via a priority number such that 1 is the highest priority and something like 10 would be lower. 0 indicates that priority is not important for this service and can be used to ensure the provider is placed at or near the end without having to set some arbitrarily high number.

Core Services

The core services are all behind APIs so that they can be reimplemented without affecting developers who are using the services. Most of the services have plugin/provider points so that customizations can be added into the system without touching the core services code. For example, let's say a deployer has a specialized authentication system and wants to manage the authentication calls which come into the system. The implementor can simply implement an `AuthenticationProvider` and then register it with the DS2 kernel's `ServiceManager`. This can be done at any time and does not have to be done during Kernel startup. This allows providers to be swapped out at runtime without disrupting the DS2 service if desired. It can also speed up development by allowing quick hot redeploys of code during development.

Caching Service

Provides for a centralized way to handle caching in the system and thus a single point for configuration and control over all caches in the system. Provider and plugin developers are strongly encouraged to use this rather than implementing their own caching. The caching service has the concept of scopes so even storing data in maps or lists is discouraged unless there are good reasons to do so.

Configuration Service

The `ConfigurationService` controls the external and internal configuration of DSpace 2. It reads `Properties` files when the kernel starts up and merges them with any dynamic configuration data which is available from the services. This service allows settings to be updated as the system is running, and also defines listeners which allow services to know when their configuration settings have changed and take action if desired. It is the central point to access and manage all the configuration settings in DSpace 2.

Manages the configuration of the DSpace 2 system. Can be used to manage configuration for providers and plugins also.

EventService

Handles events and provides access to listeners for consumption of events.

RequestService

In DS2 a request is an atomic transaction in the system. It is likely to be an HTTP request in many cases but it does not have to be. This service provides the core services with a way to manage atomic transactions so that when a request comes in which requires multiple things to happen they can either all succeed or all fail without each service attempting to manage this independently. In a nutshell this simply allows identification of the current request and the ability to discover if it succeeded or failed when it ends. Nothing in the system will enforce usage of the service, but we encourage developers who are interacting with the system to make use of this service so they know if the request they are participating in with has succeeded or failed and can take appropriate actions.

SessionService

In DS2 a session is like an `HttpSession` (and generally is actually one) so this service is here to allow developers to find information about the current session and to access information in it. The session identifies the current

user (if authenticated) so it also serves as a way to track user sessions. Since we use HttpSession directly it is easy to mirror sessions across multiple servers in order to allow for no-interruption failover for users when servers go offline.

Examples

Configuring Event Listeners

Event Listeners can be created by overriding the the EventListener interface:

In Spring:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>
  <bean id="dspace" class="org.dspace.utils.DSpace"/>
  <bean id="dspace.eventService"
        factory-bean="dspace"
        factory-method="getEventService"/>
  <bean class="org.my.EventListener">
    <property name="eventService" >
      <ref bean="dspace.eventService"/>
    </property>
  </bean>
</beans>
```

(org.my.EventListener will need to register itself with the EventService, for which it is passed a reference to that service via the eventService property.)

or in Java:

```
DSpace dspace = new DSpace();
EventService eventService = dspace.getEventService();
EventListener listener = new org.my.EventListener();
eventService.registerEventListener(listener);
```

(This registers the listener externally – the listener code assumes it is registered.)

Tutorials

Several tutorials on Spring / DSpace Services are available:

- [DSpace Spring Services Tutorial](#)
- [The TAO of DSpace Services](#)

7.4.5 Storage Layer

In this section, we explain the storage layer: the database structure, maintenance, and the bitstream store and configurations.

- 1 [RDBMS / Database Structure](#)
 - 1.1 [Maintenance and Backup](#)
 - 1.2 [Configuring the RDBMS Component](#)
- 2 [Bitstream Store](#)
 - 2.1 [Cleanup](#)
 - 2.2 [Backup](#)
 - 2.3 [Configuring the Bitstream Store](#)
 - 2.3.1 [Configuring Traditional Storage](#)
 - 2.3.2 [Configuring SRB Storage](#)

RDBMS / Database Structure

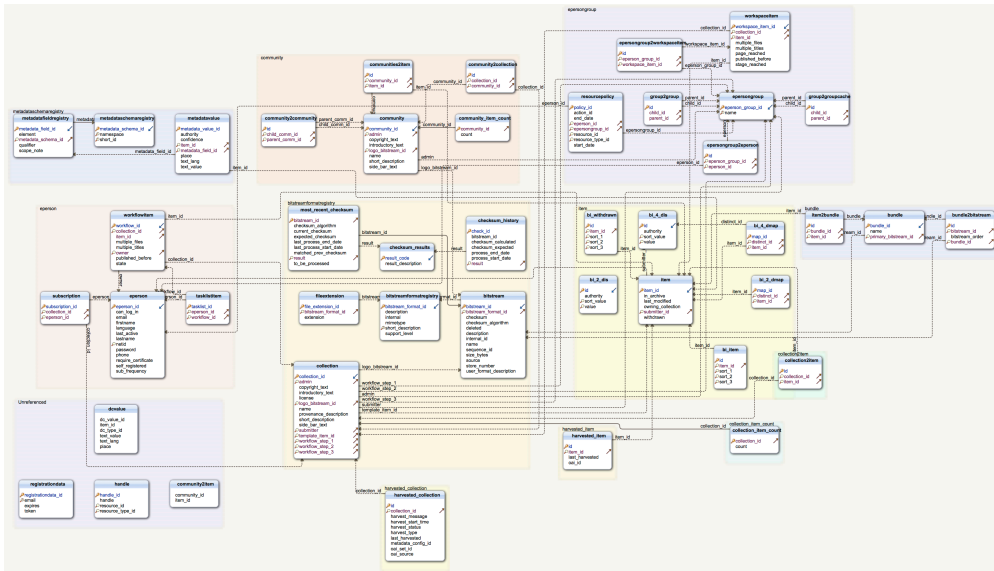
DSpace uses a relational database to store all information about the organization of content, metadata about the content, information about e-people and authorization, and the state of currently-running workflows. The DSpace system also uses the relational database in order to maintain indices that users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



1.8 Schema

This Database schema is not fully up to date with DSpace 4.0. 2 additional table to store [item level versioning](#) information were added to DSpace 3 are currently not represented in this diagram.

More significantly, DSpace 5 changes the way many attributes are stored and moves them from DSpaceObjects to the [metadatavalue](#) table ([DS-1582](#)).



Most of the functionality that DSpace uses can be offered by any standard SQL database that supports transactions. However at this time, DSpace APIS use some features specific to [PostgreSQL](#) and [Oracle](#), 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 primary class is `DatabaseManager`, which executes SQL queries and returns `TableRow` or `TableRowIterator` objects.

The database schema used by DSpace is initialized and upgraded automatically using [Flyway DB](#). The `DatabaseUtils` class manages all Flyway API calls, and executes the SQL migrations under the `org.dspace.storage.rdbms.sqlmigration` package and the Java migrations under the `org.dspace.storage.rdbms.migration` package. While Flyway is automatically initialized and executed during the initialization of `DatabaseManager`, various [Database Utilities](#) are also available on the command line..

Maintenance and Backup

When using PostgreSQL, it's a good idea to perform regular 'vacuuming' of the database to optimize performance. By default, PostgreSQL performs [automatic vacuuming](#) on your behalf. However, if you have this feature disabled, then we recommend scheduling the `vacuumdb` command to run on a regular basis.

```
# clean up the database nightly
40 2 * * * /usr/local/pgsql/bin/vacuumdb --analyze dspace > /dev/null 2>&1
```

Backups: The DSpace database can be backed up and restored using usual [PostgreSQL Backup and Restore](#) methods, for example with `pg_dump` and `psql`. However when restoring a database, you will need to perform these additional steps:

- 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]/etc/postgres/update-sequences.sql`, for example with:

```
psql -U dspace -f [dspace]/etc/update-sequences.sql
```

Configuring the RDBMS Component

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

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

Bitstream Store

DSpace offers two means for storing content.

1. Storage in the file system on the server
2. Storage using [SRB \(Storage Resource Broker\)](#)

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

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:

```
[dspace]/assetstore/12/34/56/12345678901234567890123456789012345678
```

The reasons for storing files this way are:

- Using a randomly-generated 38-digit number means that the 'number space' is less cluttered than simply using the primary keys, which are allocated sequentially and are thus close together. This means that the bitstreams in the store are distributed around the directory structure, improving access efficiency.
- The internal ID is used as the filename partly to avoid requiring an extra lookup of the filename of the bitstream, and partly because bitstreams may be received from a variety of operating systems. The original name of a bitstream may be an illegal UNIX filename.

When storing a bitstream, the *BitstreamStorageManager* DOES set the following fields in the corresponding database table row:

- *bitstream_id*
- *size*
- *checksum*
- *checksum_algorithm*
- *internal_id*
- *deleted*
- *store_number*

- The remaining fields are the responsibility of the *Bitstream* content management API class.

The bitstream storage manager is fully transaction-safe. In order to implement transaction-safety, the following algorithm is used to store bitstreams:

1. A database connection is created, separately from the currently active connection in the current DSpace context.
2. An unique internal identifier (separate from the database primary key) is generated.
3. The bitstream DB table row is created using this new connection, with the *deleted* column set to *true*.
4. The new connection is *_commit_ted*, so the 'deleted' bitstream row is written to the database
5. The bitstream itself is stored in a file in the configured 'asset store directory', with a directory path and filename derived from the internal ID
6. The *deleted* flag in the bitstream row is set to *false*. This will occur (or not) as part of the current DSpace *Context*.

This means that should anything go wrong before, during or after the bitstream storage, only one of the following can be true:

- No bitstream table row was created, and no file was stored
 - A bitstream table row with *deleted=true* was created, no file was stored
 - A bitstream table row with *deleted=true* was created, and a file was 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.

Cleanup

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* command, which can in turn be easily executed from a shell on the server machine using `[dspace]/bin/dspace 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.

```
# Clean up any deleted files from local storage on first of the month at 2:40am
40 2 1 * * [dspace]/bin/dspace cleanup > /dev/null 2>&1
```

Backup

The bitstreams (files) in traditional storage may be backed up very easily by simply 'tarring' or 'zipping' the `[dspace]/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.

With DSpace 1.7 and above, there is also the option to backup both files and metadata via the [AIP Backup and Restore](#) feature.

Configuring the Bitstream Store

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

Configuring Traditional Storage

Bitstream stores in the file system on the server are configured like this:

```
assetstore.dir = [dspace]/assetstore
```

(Remember that `[dspace]` is a placeholder for the actual name of your DSpace install directory).

The above example specifies a single asset store.

```
assetstore.dir = [dspace]/assetstore_0
assetstore.dir.1 = /mnt/other_filesystem/assetstore_1
```

The above example specifies two asset stores. `assetstore.dir` specifies the asset store number 0 (zero); after that use `assetstore.dir.1`, `assetstore.dir.2` and so on. The particular asset store a bitstream is stored in is held in the database, so don't move bitstreams between asset stores, and don't renumber them.

By default, newly created bitstreams are put in asset store 0 (i.e. the one specified by the `assetstore.dir` property.) This allows backwards compatibility with pre-DSpace 1.1 configurations. To change this, for example when asset store 0 is getting full, add a line to `dspace.cfg` like:

```
assetstore.incoming = 1
```

Then restart DSpace (Tomcat). New bitstreams will be written to the asset store specified by *assetstore.dir.1*, which is */mnt/other_filesystem/assetstore_1* in the above example.

Configuring SRB Storage

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

For example, let's say asset store number 1 will refer to SRB. There will be a set of SRB account parameters like this:

```
srb.host.1 = mysrbmcahost.myu.edu
srb.port.1 = 5544
srb.mcatzone.1 = mysrbzone
srb.mdasdomainname.1 = mysrbdomain
srb.defaultstorageresource.1 = mydefaultsrbresource
srb.username.1 = mysrbuser
srb.password.1 = mysrbpassword
srb.homedirectory.1 = /mysrbzone/home/mysrbuser.mysrbdomain
srb.parentdir.1 = mysrbdspaceassetstore
```

Several of the terms, such as *mcatzone*, have meaning only in the SRB context and will be familiar to SRB users. The last, *srb.parentdir.n*, can be used to add (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.

7.5 History



- [Changes in 5.x](#)
- [Changes in 4.x](#)
- [Changes in 3.x](#)

- [Changes in 1.8.x](#)
- [Changes in 1.7.x](#)
- [Changes in 1.6.x](#)
- [Changes in 1.5.x](#)
- [Changes in 1.4.x](#)
- [Changes in 1.3.x](#)
- [Changes in 1.2.x](#)
- [Changes in 1.1.x](#)



7.5.1 Changes in 5.x

- [Changes in DSpace 5.1](#)
- [Changes in DSpace 5.0](#)

Changes in DSpace 5.1

Key	Summary	T	Created	Updated	Due	Assignee	Reporter	P	Status	Resolution
DS-2290	DSpace5 Estonian translation		Nov 13, 2014	Feb 19, 2015		Ivan Masár	Heiki Epner	 Minor	Closed	Fixed

1 issue

Key	Summary	T	Created	Updated	Due	Assignee	Reporter	P	Status	Resolution
DS-2429	Mirage 2 - No mention of Git prerequisite in documentation		Jan 28, 2015	Jan 28, 2015		Unassigned	Art Lowel	 Minor	Closed	Fixed

1 issue

Key	Summary	T	Created	Updated	Due	Assignee	Rep
DS-640	Internal System Error when browsing with wrong argument		Aug 07, 2010	Feb 03, 2015		Kim Shepherd	Har
DS-1265	Sitemap generator does not defend against empty repository		Sep 13, 2012	Feb 22, 2015		Mark H. Wood	Mar
DS-1702	Cross-site scripting (XSS injection) is possible in JSPUI Recent Submissions listings		Oct 15, 2013	Feb 24, 2015		Luigi Andrea Pascarelli	Jean Zha

Key	Summary	T	Created	Updated	Due	Assignee	Rep
DS-1896	XMLUI returns 500 response for most invalid "/" static URLs	<input checked="" type="checkbox"/>	Jan 30, 2014	Feb 24, 2015		Tim Donohue	Tim
DS-2034	Problem in [Control Panel]->[Dspace Configuration] - org.dspace.app.xmlui.wing.WingInvalidArgument : The 'characters' parameter is required for list items.	<input checked="" type="checkbox"/>	Jun 20, 2014	Feb 17, 2015		Hardy Pottinger	Roy
DS-2044	Cross-site scripting (XSS injection) is possible in JSPUI Discovery search form	<input checked="" type="checkbox"/>	Jun 30, 2014	Feb 25, 2015		Luigi Andrea Pascarelli	Gab Mirc
DS-2130	XMLUI allows access to theme XSL files	<input checked="" type="checkbox"/>	Sep 03, 2014	Feb 24, 2015		Tim Donohue	Har Pott
DS-2201	Unable to complete installation of DSpace with non-empty variable "db.schema" configuration file "build.properties"	<input checked="" type="checkbox"/>	Oct 17, 2014	Jan 28, 2015		Unassigned	CTU Dev
DS-2278	Error pages improperly formatted on certain url patterns	<input checked="" type="checkbox"/>	Nov 11, 2014	Feb 23, 2015		Tim Donohue	Terr
DS-2355	bug in SpiderDetector.java	<input checked="" type="checkbox"/>	Dec 10, 2014	Feb 20, 2015		Unassigned	Willi Tan
DS-2412	The oai "Show more" link in stylesheet has a fixed verb	<input checked="" type="checkbox"/>	Jan 16, 2015	Jan 20, 2015		Ivan Masár	Ond Koš
DS-2415	cannot add new metadata field to an existing item with Oracle back-end database	<input checked="" type="checkbox"/>	Jan 20, 2015	Feb 20, 2015		Unassigned	Har Pott
DS-2419	JSP UI ignores authorization.admin.usage	<input checked="" type="checkbox"/>	Jan 23, 2015	Feb 22, 2015		Unassigned	Eike
DS-2425	Typos in xoai.xml prevent filters from being used	<input checked="" type="checkbox"/>	Jan 26, 2015	Jan 27, 2015		Ivan Masár	Chri Sch
DS-2427	DSpace API does not always filter results by the DB schema of current connection	<input checked="" type="checkbox"/>	Jan 27, 2015	Feb 11, 2015		Tim Donohue	Tim
DS-2435	JSPUI should send HTTP 400 Bad Request if specified Browse index is not configured	<input checked="" type="checkbox"/>	Feb 03, 2015	Feb 03, 2015		Pascal-Nicolas Becker	Pas Bec
DS-2438	OAI indexing fails if a metadata field contains a value with more than 32766 Bytes	<input checked="" type="checkbox"/>	Feb 04, 2015	Feb 04, 2015		Ivan Masár	Chri Sch

Key	Summary	T	Created	Updated	Due	Assignee	Rep
DS-2445	XMLUI Directory Traversal Vulnerability	<input checked="" type="checkbox"/>	Feb 05, 2015	Feb 26, 2015		Tim Donohue	Tim
DS-2448	JSPUI Path Traversal Vulnerability	<input checked="" type="checkbox"/>	Feb 09, 2015	Feb 25, 2015		Pascal-Nicolas Becker	Pas Bec

19 issues

Changes in DSpace 5.0

New Features in 5.0 (16 issues)

Key	Summary	Assignee	Reporter
DS-1222	Alternative Metrics - Framework, Integration of AltMetric and PlumX	Mark H. Wood	Graham Triggs
DS-1641	Perform Batch Imports from Administrative UI	Peter Dietz	Tim Donohue
DS-1880	Language menu	Ivan Masár	Jordan Pišanc
DS-1968	Select the collection already selected in the previous operation	Keiji Suzuki	Keiji Suzuki
DS-1994	Use HTML5 to upload files in Submission Process of JSPUI	Pascal-Nicolas Becker	Pascal-Nicolas Becker
DS-2049	DSpace/ORCID integration	Kevin Van de Velde (@mire)	Hardy Pottinger
DS-2052	Mirage 2 Responsive theme for the XMLUI	Kevin Van de Velde (@mire)	Bram Luyten (@mire)
DS-2053	XMLUI compatibility for Sherpa/Romeo lookup during item submission	Kevin Van de Velde (@mire)	Bram Luyten (@mire)
DS-2061	Linked (Open) Data support for DSpace	Pascal-Nicolas Becker	Pascal-Nicolas Becker
DS-2105	Provide ImageMagick / Ghostscript Filter Media Plugin for Thumbnail Generation	Peter Dietz	Terry Brady
	Provide a place for third-party plugins	Mark H. Wood	Mark H. Wood

Key	Summary	Assignee	Reporter
DS-2107			
DS-2108	Add an XMLUI aspect to report Google Analytics stats	Robin Taylor	Robin Taylor
DS-2162	Per item configurable visual indicators for browse and search results	Kostas Stamatis	Kostas Stamatis
DS-2168	New REST api with CRUD operations	Peter Dietz	CTU Developers
DS-2175	Generate citation PDF with cover page	Peter Dietz	Peter Dietz
DS-2177	Batch Import via the UI (new features and improvements)	Kostas Stamatis	Kostas Stamatis

16 issues

Improvements in 5.0 (64 issues)

Key	Summary	Assignee	Reporter
DS-635	Rendering MathML code in abstracts using MathJax	Peter Dietz	George Simeonov
DS-1577	Replace dependency on commons-httpclient	Mark H. Wood	Mark H. Wood
DS-1578	improve clean_database ant target to work better with Oracle	Hardy Pottinger	Hardy Pottinger
DS-1582	All DSpaceObjects should have metadata support	Mark H. Wood	Mark H. Wood
DS-1596	Page not found look and feel	Tim Donohue	Bram Luyten (@mire)
DS-1649	OAI 2.1 : Improvements (and fixes)	João Melo	João Melo
DS-1738	Support NUMERIC columns in database query result sets	Mark H. Wood	Mark H. Wood

Key	Summary	Assignee	Reporter
DS-1746	Remove strange, redundant pool validation from DBMS layer	Mark H. Wood	Mark H. Wood
DS-1775	Increase robustness of SolrServiceImpl date format guessing	Mark H. Wood	Bram Luyten (@mire)
DS-1797	Add a new oai format 'junii2'	Ivan Masár	Keiji Suzuki
DS-1817	Improvement of Collection Dropdown on Move Item Page	Ivan Masár	Thomas Misilo
DS-1820	Introduction of URL validators in SolrServiceImpl breaks .local urls	Unassigned	Roeland Dillen (@mire)
DS-1831	Official statement on DSpace 5 browser support	Bram Luyten (@mire)	Bram Luyten (@mire)
DS-1838	Solr monthly statistics is not localised	Tim Donohue	Eliana de Mattos Pinto Coelho
DS-1842	Use geoup and dnsjava official artifacts at Maven Central	Mark H. Wood	Mark H. Wood
DS-1861	Use the official JMockit artifacts instead of our own	Mark H. Wood	Mark H. Wood
DS-1883	Rehabilitate DCValue	Mark H. Wood	Mark H. Wood
DS-1889	Build DSpace into a standard directory to better support install automation	Mark H. Wood	Tim Donohue
DS-1906	Shibboleth attributes may need to be reconverted	Hardy Pottinger	Pascal-Nicolas Becker
DS-1917	JSPUI validity fixes	Andrea Bollini	Ivan Masár

Showing 20 out of [64 issues](#)

Bugs Fixed in 5.0 (159 issues)

Key	Summary	Assignee	Reporter
DS-682	The Select Collection step performs badly with a large number of collections	Peter Dietz	Robin Taylor
DS-1411	uncaught NPE in stats-log-converter -m	Mark H. Wood	Ivan Masár
DS-1418	Month names without translation	Unassigned	Anonymous (No Reply)
DS-1447	Login as User redirecting to wrong page	Ivan Masár	Christos Rodosthenous
DS-1531	bug in current DSpace (3.1) with log importing	Mark H. Wood	James Halliday
DS-1597	Browse subjects by collection count bug	Kevin Van de Velde (@mire)	Kevin Van de Velde (@mire)
DS-1599	l18nUtil.getMessage() does not return the intended language's message	Andrea Bollini	Keiji Suzuki
DS-1781	LDAP never uses the specified email_field	Tim Donohue	Ivan Masár
DS-1795	When run command dspace "dspace stat-initial"	Mark H. Wood	Anonymous (No Reply)
DS-1798	Distributed dspace.cfg causes noise about unconfigured XSLTDisseminationCrosswalk	Mark H. Wood	Mark H. Wood
DS-1805	Maven filtering broken for SOLR artifact	Hardy Pottinger	Bram Luyten (@mire)
DS-1816	Incorrect label for search in community in navigation section.	Ivan Masár	Bavo Van Geit
DS-1821	Internationalize the bitstream access icon alt text	Ivan Masár	Andrea Bollini
DS-1823	Move dspace.url to build.properties as an independent variable (default value in dspace.cfg can cause issues)	Bram Luyten (@mire)	Bram Luyten (@mire)
DS-1835	Invalid bootstrap CSS	Mark H. Wood	Mark H. Wood
	OAI-PMH indexes metadata of non-public Items	João Melo	Tim Donohue

Key	Summary	Assignee	Reporter
DS-1856			
DS-1858	String "Browse Items by:" of navbar is not in messages.properties	Tim Donohue	Tiago Murakami
DS-1862	Too many ROMEs	Mark H. Wood	Mark H. Wood
DS-1873	Checksum Checker Emailer sends emails for 0 issues, doesn't specify any site info	Tim Donohue	Tim Donohue
DS-1886	Delete unnecessary spring discovery configuration xml.	Kevin Van de Velde (@mire)	Bavo Van Geit









Showing 20 out of 159 issues

7.5.2 Changes in 4.x

- [Changes in DSpace 4.3](#)
- [Changes in DSpace 4.2](#)
- [Changes in DSpace 4.1](#)
- [Changes in DSpace 4.0](#)





Changes in DSpace 4.3







Key	Summary	T	Created	Updated	Due	Assignee	Reporter	P	Status	Res
DS-1702	Cross-site scripting (XSS injection) is possible in JSPUI Recent Submissions listings	<input checked="" type="checkbox"/>	Oct 15, 2013	Feb 24, 2015		Luigi Andrea Pascarelli	Jean-Paul Zhao	↑ Major	Closed	Fixe
DS-1896	XMLUI returns 500 response for	<input checked="" type="checkbox"/>	Jan 30, 2014	Feb 24, 2015		Tim Donohue	Tim Donohue	↓ Minor	Closed	Fixe

Key	Summary	T	Created	Updated	Due	Assignee	Reporter	P	Status	Res
	most invalid "/static" URLs									
DS-2044	Cross-site scripting (XSS injection) is possible in JSPUI Discovery search form		Jun 30, 2014	Feb 25, 2015		Luigi Andrea Pascarelli	Gabriela Mircea	 Major	Closed	Fixe
DS-2130	XMLUI allows access to theme XSL files		Sep 03, 2014	Feb 24, 2015		Tim Donohue	Hardy Pottinger	 Minor	Closed	Fixe
DS-2445	XMLUI Directory Traversal Vulnerability		Feb 05, 2015	Feb 26, 2015		Tim Donohue	Tim Donohue	 Blocker	Closed	Fixe
DS-2448	JSPUI Path Traversal Vulnerability		Feb 09, 2015	Feb 25, 2015		Pascal-Nicolas Becker	Pascal-Nicolas Becker	 Blocker	Closed	Fixe













6 issues

Changes in DSpace 4.2




Key	Summary	T	Created	Updated	Due	Assignee	Reporter	P	Status	Resolution
DS-913	Ukrainian translation for Manakin web interface		May 30, 2011	Jul 16, 2014		Ivan Masár	Parhomenko Yaroslav	 Minor	Closed	Fixed
DS-1906	Shibboleth attributes		Feb 07, 2014	Jul 07, 2014		Hardy Pottinger	Pascal-Nicolas Becker	 Minor	Closed	Fixed

Key	Summary	T	Created	Updated	Due	Assignee	Reporter	P	Status	Resolution
	may need to be reconverted									
DS-1932	Complete Update translation of JSPUI pt_BR		Mar 01, 2014	Mar 06, 2014		Ivan Masár	Washington Ribeiro	 Minor	Closed	Fixed
DS-1943	Language Selection _ Turkish Option		Mar 11, 2014	May 29, 2014		Mark H. Wood	Sonmez CELIK	 Minor	Closed	Fixed
DS-2047	zh_TW language for dspace 4.1 jspui		Jul 02, 2014	Jul 16, 2014		Ivan Masár	Chunmin Tai	 Minor	Closed	Fixed

5 issues




Key	Summary	T	Created	Updated	Due	Assignee	Reporter	P	Status	F
DS-1411	uncaught NPE in stats-log-converter -m		Dec 03, 2012	Mar 26, 2014		Mark H. Wood	Ivan Masár	 Minor	Closed	F
DS-1584	XMLUI "Browse by" sorting Bug		Jun 20, 2013	Apr 23, 2014		Bram Luyten (@mire)	Denis Fdz	 Major	Closed	F
DS-1919	Solr Search Empty FilterQuery bug		Feb 20, 2014	Jul 28, 2014		Hardy Pottinger	Denis Fdz	 Major	Closed	F
DS-1928	OAI-PMH Identify response well-formed but invalid		Feb 27, 2014	Jul 17, 2014		Ivan Masár	Ondej Košarko	 Trivial	Closed	F
DS-1940	DS-1867 caused error running "mvn package"		Mar 08, 2014	Jul 28, 2014		Tim Donohue	Mohsen	 Major	Closed	F
DS-1944	http://mobile.demo.dspace.org/xmlui/		Mar 11, 2014	Mar 12, 2014		Hardy Pottinger	Thomas Misilo	 Critical	Closed	F

Key	Summary	T	Created	Updated	Due	Assignee	Reporter	P	Status	F
DS-1946	Solr floods catalina.out with unwanted messages	<input checked="" type="checkbox"/>	Mar 12, 2014	Apr 09, 2014		Mark H. Wood	Mark H. Wood	↓ Minor	Closed	F
DS-1947	Missing m-tweaks.js for mobile theme	<input checked="" type="checkbox"/>	Mar 13, 2014	Jul 16, 2014		Tim Donohue	Thomas Misilo	↓ Minor	Closed	F
DS-1957	incorrect xml workflow script for oracle	<input checked="" type="checkbox"/>	Apr 01, 2014	Jul 09, 2014		Mark H. Wood	Roeland Dillen (@mire)	↑ Major	Closed	F
DS-1958	Discovery OutOfMemoryError when indexing Large Bitstreams	<input checked="" type="checkbox"/>	Apr 02, 2014	Jul 17, 2014		Mark H. Wood	Mark Diggory	↓ Trivial	Closed	F
DS-1961	Use HTTPS with oss.sonatype.org repository	<input checked="" type="checkbox"/>	Apr 04, 2014	Jul 28, 2014		Mark H. Wood	Mark H. Wood	↓ Minor	Closed	F
DS-1970	to many open files exception when update lucene index	<input checked="" type="checkbox"/>	Apr 15, 2014	Apr 15, 2014		Hardy Pottinger	Roeland Dillen (@mire)	↑ Major	Closed	F
DS-1971	'bte-io' (v 0.9.2.3) dependency from EKT has an invalid SNAPSHOT dependency in its POM	<input checked="" type="checkbox"/>	Apr 15, 2014	Jun 03, 2014		Hardy Pottinger	Tim Donohue	↓ Minor	Closed	F
DS-1986	REST API holds on to context for too long, should use DB pool	<input checked="" type="checkbox"/>	Apr 27, 2014	Jul 16, 2014		Peter Dietz	Peter Dietz	↑ Critical	Closed	F
DS-1998	"dspace classpath" CLI command does nothing, throws error	<input checked="" type="checkbox"/>	May 11, 2014	Jun 05, 2014		Mark H. Wood	Ivan Masár	↓ Minor	Closed	F
DS-2004	Catalan translation of Discovery strings	<input checked="" type="checkbox"/>	May 15, 2014	May 16, 2014		Ivan Masár	Àlex Magaz Graça	↓ Minor	Closed	F
DS-2013	JSPUI with Oracle DB - Browse items with	<input checked="" type="checkbox"/>	May 22, 2014	Jun 05, 2014		Mark H. Wood	Denis Fdz	↑ Major	Closed	F


Key	Summary	T	Created	Updated	Due	Assignee	Reporter	P	Status	F
	THUMBNAILS unimplemented									
DS-2035	dim crosswalk has missing values		Jun 24, 2014	Jul 16, 2014		Tim Donohue	Antoine Snyers (@mire)	↓ Minor	Closed	F
DS-2036	DSpace upgrade with oracle database, no discovery results		Jun 24, 2014	Sep 22, 2014		Kevin Van de Velde (@mire)	Kevin Van de Velde (@mire)	↑ Major	Closed	F
DS-2038	Oracle dspace-schema_3-4.sql upgrade script contains a minor error		Jun 24, 2014	Aug 07, 2014		Unassigned	Hardy Pottinger	↓ Minor	Closed	F

Showing 20 out of 22 issues

Changes in DSpace 4.1

Key	Summary	T	Created	Updated	Due	Assignee	Reporter	P	Status	Resolution
DS-1860	Community-list doesn't show all collections		Jan 13, 2014	Jul 28, 2014		Ivan Masár	Denis Fdz	↑ Major	Closed	Fixed
DS-1866	"Consuming Web Services" curation task is missing documentation		Jan 15, 2014	Jan 21, 2014		Richard Rodgers	Tim Donohue	↓ Minor	Closed	Fixed
DS-1911	Update translation of JSPUI pt_BR		Feb 14, 2014	Feb 14, 2014		Ivan Masár	Tiago Murakami	↓ Minor	Closed	Fixed

3 issues

Key	Summary	T	Created	Updated	Due	Assignee	Reporter	P	Status	Reso
DS-1445	XOAI validation issues - ListRecords		Jan 09, 2013	Jul 24, 2014		João Melo	Christos Rodosthenous	↑ Major	Closed	Fixed

Key	Summary	T	Created	Updated	Due	Assignee	Reporter	P	Status	Reso
	response gave a noRecordsMatch									
DS-1531	bug in current DSpace (3.1) with log importing	<input checked="" type="checkbox"/>	Apr 06, 2013	Feb 05, 2014		Mark H. Wood	James Halliday	↓ Minor	Closed	Fixed
DS-1536	having a DOT in handle prefix causes identifier.uri to be cut off when being created	<input checked="" type="checkbox"/>	Apr 17, 2013	Sep 22, 2014		Ivan Masár	Jose Blanco	↑ Major	Closed	Fixed
DS-1744	Upgrade to latest log4j	<input checked="" type="checkbox"/>	Oct 30, 2013	Dec 20, 2013		Mark H. Wood	Mark H. Wood	↓ Minor	Closed	Fixed
DS-1756	Wrongly aligned text on item view in mobile theme	<input checked="" type="checkbox"/>	Nov 05, 2013	Jan 29, 2014		Ivan Masár	Marina Muilwijk	↓ Trivial	Closed	Fixed
DS-1757	Missing images in mobile theme	<input checked="" type="checkbox"/>	Nov 05, 2013	Jan 22, 2014		Ivan Masár	Marina Muilwijk	↓ Trivial	Closed	Fixed
DS-1779	Pagination link error in JSPUI discovery search	<input checked="" type="checkbox"/>	Nov 11, 2013	Feb 20, 2014		Kim Shepherd	Raul Ruiz	↑ Major	Closed	Fixed
DS-1795	When run command dspace "dspace stat-initial"	<input checked="" type="checkbox"/>	Nov 17, 2013	Feb 26, 2014		Mark H. Wood	Anonymous (No Reply)	↑ Major	Closed	Fixed
DS-1816	Incorrect label for search in community in navigation section.	<input checked="" type="checkbox"/>	Dec 02, 2013	Jan 27, 2014		Ivan Masár	Bavo Van Geit	↓ Minor	Closed	Fixed
DS-1821	Internationalize the bitstream access icon alt text	<input checked="" type="checkbox"/>	Dec 04, 2013	Feb 12, 2014		Ivan Masár	Andrea Bollini	↓ Minor	Closed	Fixed

Key	Summary	T	Created	Updated	Due	Assignee	Reporter	P	Status	Reso
DS-1832	Proxy configuration set in system properties if empty	<input checked="" type="checkbox"/>	Dec 11, 2013	Dec 21, 2013		Kevin Van de Velde (@mire)	Kevin Van de Velde (@mire)	↑ Major	Closed	Fixed
DS-1833	Collection content source harvesting test does not work with ORE	<input checked="" type="checkbox"/>	Dec 11, 2013	Jan 30, 2014		Kevin Van de Velde (@mire)	Kevin Van de Velde (@mire)	↑ Major	Closed	Fixed
DS-1834	Collection content source harvesting test does not check sets properly	<input checked="" type="checkbox"/>	Dec 11, 2013	Feb 17, 2014		Kevin Van de Velde (@mire)	Kevin Van de Velde (@mire)	↑ Major	Closed	Fixed
DS-1835	Invalid bootstrap CSS	<input checked="" type="checkbox"/>	Dec 11, 2013	Jan 29, 2014		Mark H. Wood	Mark H. Wood	↓ Trivial	Closed	Fixed
DS-1846	Cannot deposit new item via SWORD	<input checked="" type="checkbox"/>	Dec 18, 2013	Jul 28, 2014		Andrea Schweer	Àlex Magaz Graça	↑ Major	Closed	Fixed
DS-1848	OAI harvest issues when starting from control panel/ command line	<input checked="" type="checkbox"/>	Dec 20, 2013	Feb 18, 2014		Kevin Van de Velde (@mire)	Kevin Van de Velde (@mire)	↓ Minor	Closed	Fixed
DS-1857	Unhandled exception in BTE batch import when uploading CSV files with misconfiguration options	<input checked="" type="checkbox"/>	Jan 08, 2014	Jan 31, 2014		Kostas Stamatis	Kostas Stamatis	↑ Critical	Closed	Fixed
DS-1863	JSPUI eperson and group selection should	<input checked="" type="checkbox"/>	Jan 15, 2014	Feb 20, 2014		Ivan Masár	Denis Fdz	↓ Minor	Closed	Fixed

Key	Summary	T	Created	Updated	Due	Assignee	Reporter	P	Status	Reso
	use the new theme									
DS-1867	Maven build issues from [src] /dspace/ , error finding target/build.properties	<input checked="" type="checkbox"/>	Jan 15, 2014	Jul 28, 2014		Tim Donohue	Tim Donohue	↓ Minor	Closed	Fixed
DS-1873	Checksum Checker Emailer sends emails for 0 issues, doesn't specify any site info	<input checked="" type="checkbox"/>	Jan 21, 2014	Jan 31, 2014		Tim Donohue	Tim Donohue	↓ Minor	Closed	Fixed

Showing 20 out of 32 issues

Changes in DSpace 4.0

New Features in 4.0 (27 issues)

Key	Summary	Assignee	Reporter
DS-824	Request Copy function for XMLUI and JSPUI	Ivan Masár	Brian Freels-Stendel
DS-831	Recent items addon : Listing of most recently added items to DSpace	Keiji Suzuki	João Melo
DS-1083	Create new users from the command line	Mark H. Wood	Stuart Lewis
DS-1252	(JSP)UI Import from bibliographics database/formats	Andrea Bollini	Andrea Bollini
DS-1269	EmailService to encapsulate the sending of mail	Mark H. Wood	Mark H. Wood
DS-1336	Creative Commons Locale	Ivan Masár	Juan Corrales Correyero
DS-1456	"dspace version" command-line script	Mark H. Wood	Ivan Masár

Key	Summary	Assignee	Reporter
DS-1482	Add a way for harvesters to find recently added items (request from Google)	Unassigned	Tim Donohue
DS-1483	Store link to "primary bitstream" in citation_pdf_url for Google Scholar (request from Google)	Andrea Schweer	Tim Donohue
DS-1535	DOI support for dspace-api	Mark H. Wood	Pascal-Nicolas Becker
DS-1567	Stream multiple commands into one invocation of bin/dspace	Mark H. Wood	Mark H. Wood
DS-1613	Porting curation task administrative UI to JSPUI	Andrea Bollini	Keiji Suzuki
DS-1622	Porting of the Login As feature to JSPUI	Andrea Bollini	Andrea Bollini
DS-1623	Upgrade DSpace-SOLR to SOLR 4	Andrea Bollini	Andrea Bollini
DS-1624	Reset password in edit eperson for administrator (as in XMLUI)	Andrea Bollini	Andrea Bollini
DS-1633	Sherpa/Romeo integration in the submission upload step	Andrea Bollini	Andrea Bollini
DS-1639	AJAX progress bar for file upload in JSPUI	Andrea Bollini	Andrea Bollini
DS-1647	Curation Task for Consuming Web Services	Richard Rodgers	Richard Rodgers
DS-1657	Adopt/Create an official DSpace REST API	Peter Dietz	Tim Donohue
DS-1675	New JSPUI look & feel	Andrea Bollini	Andrea Bollini

Showing 20 out of [27 issues](#)

Improvements in 4.0 (50 issues)

Key	Summary	Assignee	Reporter
DS-286	Remove dspace/bin/dspace_migrate script	Mark H. Wood	Stuart Lewis
DS-790	SOLR - Spider detection to match on hostname or useragent	Mark H. Wood	Peter Dietz
DS-792	When 'mail.server.disabled = true' put text of email in log file?	Mark H. Wood	usha sharma
DS-842	Language switch for xmlui and some basic i18n stuff	Ivan Masár	Claudia Jürgen
DS-1085	EPerson last_active field is defined but never filled	Mark H. Wood	Mark H. Wood
DS-1106	Solr search accent insensitive	Andrea Bollini	Fabio Bolognesi
DS-1168	Show a single search box in the front page	Unassigned	Àlex Magaz Graça
DS-1259	use better image downscaling method in filter-media	Bram Luyten (@mire)	Ivan Masár
DS-1272	Enable Discovery By Default in XMLUI	Kevin Van de Velde (@mire)	Mark Diggory
DS-1355	batch-create users from command line	Mark H. Wood	Ivan Masár
DS-1360	A porting advanced embargo function to JSPUI	Andrea Bollini	Keiji Suzuki
DS-1409	Discovery should obsolete webui.strengths.cache	Ivan Masár	Ivan Masár
DS-1459	Testing of dissemination crosswalks	Ivan Masár	Pascal-Nicolas Becker
DS-1460	Add SOLR logging config file	Mark H. Wood	Hilton Gibson
DS-1472	Fix Capitalization of Submissions & workflow tasks in xmlui messages.xml	Ivan Masár	Thomas Misilo
	Improvement of Collection Dropdown	Ivan Masár	Thomas Misilo

Key	Summary	Assignee	Reporter
DS-1475			
DS-1481	"dc.date.issued" is often incorrectly set (reported from Google)	Unassigned	Tim Donohue
DS-1484	l18n in default.license and input-forms.xml	Bram Luyten (@mire)	Onivaldo Rosa Junior
DS-1492	Stop ehcache new-version check	Ivan Masár	Mark H. Wood
DS-1542	make current interface language accessible in DRI	Ivan Masár	Ivan Masár

Showing 20 out of 50 issues

Bugs Fixed in 4.0 (111 issues)

Key	Summary	Assignee	Reporter
DS-402	Item mapper search case sensitive (jspui only)	Keiji Suzuki	Claudia Jürgen
DS-449	Command line utility org.dspace.app.harvest.Harvest -S throws AuthorizeException	Mark H. Wood	Toni Prieto
DS-803	'dspace harvest -g' (ping) doesn't	Mark H. Wood	Mark H. Wood
DS-888	file description at UploadStep	Mark H. Wood	Kostas Maistrelis
DS-951	Monthly stats report ignores items archived on first and last day of the month	Andrea Schweer	Andrea Schweer
DS-992	Browse by author or subject with special characters	Unassigned	Cedric Devaux
DS-1119	xmlui "wildcard policy admin tool" does nothing	Ivan Masár	james bardin
DS-1132	ItemImport BitStream Registration does not properly set the Description	Kostas Stamatis	Thomas Autry

Key	Summary	Assignee	Reporter
DS-1149	BinaryContentIngestor in SWORDv2 creates a new ORIGINALS bundle every time a bitstream is ingested to an Item	Richard Jones	Marco Fabiani
DS-1188	collection view doesn't show content by default	Kevin Van de Velde (@mire)	Ivan Masár
DS-1205	DSpace org.dspace.core.Context caching problem	Unassigned	DSpace @ Lyncode
DS-1212	Only collections are exported when exporting a community	Keiji Suzuki	Àlex Magaz Graça
DS-1235	IP authentication configuration does not apply netmask and CIDR ranges correctly	Mark H. Wood	Alexey Maslov
DS-1278	Provide a link to More Submissions at the bottom of Recent Submissions	Kevin Van de Velde (@mire)	Samuel Ottenhoff
DS-1322	Item without Title inaccessible via the UI unless for the admin via ID or Handle	Kostas Stamatis	Claudia Jürgen
DS-1335	Clarify documentation: versioned items will re-enter Collection workflow approval?	Kevin Van de Velde (@mire)	Tim Donohue
DS-1357	Mobile XMLUI theme fails to load when reloading item view page	Ivan Masár	Moises A.
DS-1399	Adding supervisor order bug	Keiji Suzuki	Jonathan Blood
DS-1410	ShibbolethAuthentication has multiple NPE and Findbugs issues	Hardy Pottinger	Ian Boston
DS-1422	Duplicate Headers when bitstream has a comma in the title (Chrome)	Ivan Masár	Jonathan Blood

Showing 20 out of 111 issues


7.5.3 Changes in 3.x

- [Changes in DSpace 3.3](#)
- [Changes in DSpace 3.2](#)
- [Changes in DSpace 3.1](#)

- [Changes in DSpace 3.0](#)

Changes in DSpace 3.3

Key	Summary	T	Created	Updated	Due	Assignee	Reporter	P	Status	Resolu
DS-1536	having a DOT in handle prefix causes identifier.uri to be cut off when being created	<input checked="" type="checkbox"/>	Apr 17, 2013	Sep 22, 2014		Ivan Masár	Jose Blanco	 Major	Closed	Fixed
DS-1619	Unable to remove items after enabling SOLRBrowseDAOs	<input checked="" type="checkbox"/>	Aug 10, 2013	Jan 16, 2015		Andrea Bollini	Andrea Bollini	 Minor	Closed	Fixed
DS-1834	Collection content source harvesting test does not check sets properly	<input checked="" type="checkbox"/>	Dec 11, 2013	Feb 17, 2014		Kevin Van de Velde (@mire)	Kevin Van de Velde (@mire)	 Major	Closed	Fixed
DS-1893	Get page refresh after adding a value in the submission forms clears all metadata in XMLUI	<input checked="" type="checkbox"/>	Jan 30, 2014	Jan 30, 2014		Kevin Van de Velde (@mire)	Kevin Van de Velde (@mire)	 Critical	Closed	Fixed
DS-1898	OAI not always closing contexts	<input checked="" type="checkbox"/>	Jan 31, 2014	Jul 28, 2014		Kevin Van de Velde (@mire)	Kevin Van de Velde (@mire)	 Major	Closed	Fixed
DS-1958	Discovery OutOfMemoryError when indexing Large Bitstreams	<input checked="" type="checkbox"/>	Apr 02, 2014	Jul 17, 2014		Mark H. Wood	Mark Diggory	 Trivial	Closed	Fixed
DS-1961	Use HTTPS with oss.sonatype.org repository	<input checked="" type="checkbox"/>	Apr 04, 2014	Jul 28, 2014		Mark H. Wood	Mark H. Wood	 Minor	Closed	Fixed
DS-1998	"dspace classpath" CLI command does	<input checked="" type="checkbox"/>	May 11, 2014	Jun 05, 2014		Mark H. Wood	Ivan Masár	 Minor	Closed	Fixed

Key	Summary	T	Created	Updated	Due	Assignee	Reporter	P	Status	Resolu
	nothing, throws error									
DS-2013	JSPUI with Oracle DB - Browse items with THUMBNAILS unimplemented	<input checked="" type="checkbox"/>	May 22, 2014	Jun 05, 2014		Mark H. Wood	Denis Fdz	 Major	Closed	Fixed

9 issues

Changes in DSpace 3.2

Improvements in 3.2 (1 issues)

Key	Summary	Assignee	Reporter
DS-1592	"HEYYYYY!!!!" appears in logs of dspace-oai	Tim Donohue	Tim Donohue

1 issue

Bugs Fixed in 3.2 (12 issues)

Key	Summary	Assignee	Reporter
DS-1123	"clean_backups" removed from help section of build.xml	Ivan Masár	Brian Freels-Stendel
DS-1479	In OAI-PMH "Identify" response, the <description> is no longer configurable	João Melo	Tim Donohue
DS-1507	OAI 2.0 Bug (set & from/until parameters)	João Melo	João Melo
DS-1527	Memory leak in CachingService	Robin Taylor	Robin Taylor
DS-1537	Invalid bitstream URL in OAI	João Melo	Keiji Suzuki
DS-1540	DSpace's .gitignore wrongly ignores all *.properties files	Tim Donohue	Tim Donohue
	dspace-Ini-client is detached from the project tree and won't build		Mark H. Wood

Key	Summary	Assignee	Reporter
DS-1550		Mark H. Wood	
DS-1554	swordv2-server.cfg not updated during build process	Ivan Masár	Andrew Waterman
DS-1576	fix names of LDAP configuration properties	Ivan Masár	LifeH2O
DS-1581	Restricted resource message shown twice	Ivan Masár	Andrea Schweer
DS-1593	discovery.cfg doesn't use the "solr.server" setting from build.properties, hardcodes its own URL	Tim Donohue	Tim Donohue
DS-1609	DSpace 3.2 OAI-PMH Functionality needs JDK 1.7 (Java 7)	João Melo	Samuel Ottenhoff

12 issues

This release also includes the following security fix:

- [DS-1603](#) - Resolves a security issue in JSPUI

Changes in DSpace 3.1

Improvements in 3.1 (3 issues)

Key	Summary	Assignee	Reporter
DS-1361	Porting the document type-based submission (DS-1127) to JSPUI	Ivan Masár	Keiji Suzuki
DS-1407	Refactor SOLR Statistics to use OpenCSV or Apache Commons CSV	Kevin Van de Velde (@mire)	Tim Donohue
DS-1457	In OAI src for jquery uses an http only	João Melo	Thomas Misilo

3 issues

Bugs Fixed in 3.1 (14 issues)

Key	Summary	Assignee	Reporter
DS-1414	OpenAIRE and Driver OAI does not work	João Melo	Juan Corrales Correyero
DS-1415	Setting collection OAI provider	João Melo	João Melo
DS-1416	NPE when removing roles from Collection workflow steps	Kevin Van de Velde (@mire)	Ian Boston
DS-1417	Thumbnails in discovery search results do not point to the item	Ivan Masár	Elvi S. Nemiz
DS-1424	IdentifierProvider.register(Context, DSpaceObject, String) should be able to throw a IdentifierException	Ivan Masár	Pascal-Nicolas Becker
DS-1425	DSpace OAI - Oracle DB issues	João Melo	Artur Konczak
DS-1426	SolrLogger performance issue	Andrea Schweer	Andrea Schweer
DS-1427	Second-level browse	Unassigned	Andrea Schweer
DS-1435	DSpace 3.0 Oracle compatibility	Hardy Pottinger	Hardy Pottinger
DS-1449	missing related items feature (xmlui)	Kevin Van de Velde (@mire)	Thomas Misilo
DS-1454	Missing type-bind in input-forms.dtd	Ivan Masár	Ivan Masár
DS-1455	In OAI DOCTYPE tag is displayed on Firefox	Ivan Masár	Thomas Misilo
DS-1461	OAI Harvester settings missing from oai.cfg	João Melo	Tim Donohue
DS-1464	StatisticsServlet attempts to show JSP twice when there are no reports	Ivan Masár	Bram Luyten (@mire)

[14 issues](#)

Changes in DSpace 3.0

New Features in 3.0 (15 issues)

Key	Summary	Assignee	Reporter
DS-829	OAI Extended Addon : Adding filter and modifying capacities to the OAI interface	João Melo	João Melo
DS-981	Created a DSpace API module to contain api changes	Kevin Van de Velde (@mire)	Kevin Van de Velde (@mire)
DS-1012	DSpace Shibboleth authentication module needs to support Lazy Authentication, NetID based authentication, and additional EPerson metadata	Scott Phillips	Scott Phillips
DS-1017	mobile dspace theme	Ivan Masár	Jonathon Scott
DS-1059	Statistics utilities should be filters	Mark H. Wood	Mark H. Wood
DS-1080	Search results preview	Mark Diggory	Howard Shand
DS-1081	Ensure that DSpace can run on java 7	Robin Taylor	Kevin Van de Velde (@mire)
DS-1130	Create controlled vocabulary support for the XMLUI.	Kevin Van de Velde (@mire)	Kevin Van de Velde (@mire)
DS-1192	New config setting to skip IP checks when authenticating a user	Sands Fish	Samuel Ottenhoff
DS-1194	Item Level Versioning	Mark Diggory	Mark Diggory
DS-1202	DSpace XOAI Data Provider	Ivan Masár	DSpace @ Lyncode
DS-1217	DSpace Discovery for JSPUI	Andrea Bollini	Andrea Bollini
DS-1218	BrowseDAO based on discovery	Andrea Bollini	Andrea Bollini

Key	Summary	Assignee	Reporter
DS-1226	Batch import from basic bibliographic formats (Endnote, BibTex, RIS, TSV, CSV)	Robin Taylor	Kostas Stamatis
DS-1241	Statistics implementation in Elastic Search	Peter Dietz	Peter Dietz

15 issues

Improvements in 3.0 (55 issues)

Key	Summary	Assignee	Reporter
DS-172	Hit highlighting in search results	Unassigned	Charles Kiplagat
DS-277	Make the OAI sets configurable	Ben Bosman	Ben Bosman
DS-601	Enable styling collection "strength"	Tim Donohue	Ivan Masár
DS-722	On login screen, keyboard input focus should be set to the first field (E-mail Address) so you don't have to use the mouse (XMLUI)	Peter Dietz	Andrea Bollini
DS-820	SFX button + SFX in Mirage	Mark Diggory	Ivan Masár
DS-844	Simplify, internationalize org.dspace.statistics.util.LocationUtils	Mark H. Wood	Mark H. Wood
DS-861	Salt PasswordAuthentication	Mark H. Wood	Alex Lemann
DS-895	Advanced Embargo Support	Mark Diggory	Mark Diggory
DS-908	Embargo Overhaul: Utilize ResourcePolicy Start and Stop timestamps for enforcing embargo in DSpace	Unassigned	Mark Diggory
DS-1078	Assign users in LDAP group to DSpace group on login	Ivan Masár	Samuel Ottenhoff
DS-1084	Handle authority and confidence fields in Bulk Editing	Ivan Masár	Keiji Suzuki

Key	Summary	Assignee	Reporter
DS-1116	Adding "referrer" to solr statistics schema	Unassigned	Fabio Bolognesi
DS-1124	Increase the default upload limit to the maximum allowed by cocoon (2GB)	Scott Phillips	Scott Phillips
DS-1127	Submission improvements: document type-based submission	Robin Taylor	Nestor Oviedo
DS-1144	Maven Project Consolidation	Mark Diggory	Mark Diggory
DS-1150	Migrate to GitHub	Tim Donohue	Mark Diggory
DS-1156	Refactor Browse related code out of InitializeDatabase into InitializeBrowseDatabase	Robin Taylor	Robin Taylor
DS-1158	Cleanup various code comment typos and whitespace issues	Tim Donohue	Ivan Masár
DS-1160	Refactor class InitializeDatabase to use Configuration Service rather than ConfigurationManager	Robin Taylor	Robin Taylor
DS-1180	LDAP: if no adminUser is set, build the DN using the object_context	Ivan Masár	Samuel Ottenhoff

Showing 20 out of 55 issues

Bugs Fixed in 3.0 (90 issues)

Key	Summary	Assignee	Reporter
DS-334	Input Form Fields, fields with restricted visibility can't be made mandatory	Unassigned	Claudia Jürgen
DS-489	OAI-ORE, References to bitstreams in harvested records incorrect	Unassigned	Keith Gilbertson
DS-766	OAI-PMH non-persistent oai identifiers	João Melo	Claudia Jürgen
DS-851	When a "qualdrop_value" is set to "required", submit form always fails	Kevin Van de Velde (@mire)	Onivaldo Rosa Junior

Key	Summary	Assignee	Reporter
DS-859	DSpace Test supporting files get quickly out of date	Mark H. Wood	Mark Diggory
DS-886	DSpaceControlledVocabulary always returns an empty list	Mark H. Wood	Ariel J. Lira
DS-899	Last modified timestamp doesn't trigger on bitstream delete	Kevin Van de Velde (@mire)	Bram Luyten (@mire)
DS-910	Encoding of discovery facet urls	Kevin Van de Velde (@mire)	Jennifer Whalan
DS-918	Concurrent task claiming and editing of metadata possible for same item in submission workflow	Kevin Van de Velde (@mire)	Bill Hays
DS-944	Authority Control Bug	Andrea Bollini	João Melo
DS-972	Mirage theme authority control popup (choice-support.js) breaks on more results	Tim Donohue	David Chandek-Stark
DS-1023	No linebreaks allowed in submission form textboxes	Mark Diggory	Bram Luyten (@mire)
DS-1039	Item view in Mirage theme broken when ORIGINAL or CONTENT bundle present and empty	Kevin Van de Velde (@mire)	Jennifer Whitney
DS-1043	'ant help' refers to 'install_code' target which does not exist	Mark H. Wood	Mark H. Wood
DS-1044	Select Collection step limits length of collection name, leading to difficulty in picking the correct collection.	Peter Dietz	Peter Dietz
DS-1048	superfluous warning in dspace.log	Kevin Van de Velde (@mire)	Ivan Masár
DS-1052	Items without date.accessioned are permanently sorted to the top of all date based searches.	Scott Phillips	Scott Phillips
DS-1055	References to bitstreams not from the 'ORIGINAL' bundle are shown in harvested items	Scott Phillips	Àlex Magaz Graça
DS-1056	When multiple authentication methods are enabled the LoginChooser will place an blank div prior to logging in	Scott Phillips	Scott Phillips
			Mark Diggory

Key	Summary	Assignee	Reporter
DS-1061	Filenames and BitstreamFormat detection break on filenames with equal signs in them	Kevin Van de Velde (@mire)	

Showing 20 out of 90 issues

7.5.4 Changes in 1.8.x

- [Changes in DSpace 1.8.3](#)
- [Changes in DSpace 1.8.2](#)
- [Changes in DSpace 1.8.1](#)
- [Changes in DSpace 1.8.0](#)

Changes in DSpace 1.8.3

Improvements in 1.8.3 (1 issues)

Key	Summary	Assignee	Reporter
DS-1587	Update 1.7.x and 1.8.x branches for Git/GitHub	Tim Donohue	Tim Donohue

[1 issue](#)

Bug Fixes in 1.8.3

- [DS-1603](#) - Resolves a security issue in JSPUI

Changes in DSpace 1.8.2

JIRA Issues Macro: \$exceptionMessage

Bugs Fixed in 1.8.2 (8 issues)

Key	Summary	Assignee	Reporter
DS-910	Encoding of discovery facet urls	Kevin Van de Velde (@mire)	Jennifer Whalan
DS-918	Concurrent task claiming and editing of metadata possible for same item in submission workflow	Kevin Van de Velde (@mire)	Bill Hays
	superfluous warning in dspace.log		Ivan Masár

Key	Summary	Assignee	Reporter
DS-1048		Kevin Van de Velde (@mire)	
DS-1107	System-wide Curation Task UI is missing a "Task" label	Tim Donohue	Tim Donohue
DS-1108	AIP Backup & Restore doesn't restore a Bitstream's "Sequence ID"	Tim Donohue	Tim Donohue
DS-1120	AIP Backup & Restore : SITE AIP has a different checksum everytime when orphaned Collection/Community groups exist	Tim Donohue	Tim Donohue
DS-1122	When adding a Bitstream to a Bundle, the 'bitstream_order' is always set to the 'sequence_id'	Tim Donohue	Tim Donohue
DS-1129	Edit Harvesting Collection Content Source tab broken	Kevin Van de Velde (@mire)	Kevin Van de Velde (@mire)

8 issues

Changes in DSpace 1.8.1

Improvements in 1.8.1 (1 issues)

Key	Summary	Assignee	Reporter
DS-1099	Bulgarian Translation for DSpace 1.8.1	Claudia Jürgen	Vladislav Zhivkov

1 issue

Bugs Fixed in 1.8.1 (14 issues)

Key	Summary	Assignee	Reporter
DS-899	Last modified timestamp doesn't trigger on bitstream delete	Kevin Van de Velde (@mire)	Bram Luyten (@mire)
DS-1052	Items without date.accessioned are permanently sorted to the top of all date based searches.	Scott Phillips	Scott Phillips
DS-1055	References to bitstreams not from the 'ORIGINAL' bundle are shown in harvested items	Scott Phillips	Àlex Magaz Graça

Key	Summary	Assignee	Reporter
DS-1056	When multiple authentication methods are enabled the LoginChooser will place an blank div prior to logging in	Scott Phillips	Scott Phillips
DS-1062	Subscription email reports new items twice, sometimes.	Scott Phillips	Scott Phillips
DS-1064	Authentication error with external login in JSPUI	Kevin Van de Velde (@mire)	Kevin Van de Velde (@mire)
DS-1068	Removing a metadata field from an item does not update the browse sorting indexes.	Scott Phillips	Scott Phillips
DS-1070	DSpaceObjectManager unnecessarily keeps references to DSpace objects	Andrea Schweer	Andrea Schweer
DS-1074	DB connection leak in DAVServlet	Robin Taylor	Bo Gundersen
DS-1075	Separator replacement in URIs fail	Robin Taylor	Bo Gundersen
DS-1076	Visualisation of static pages is broken in 1.8	Peter Dietz	Àlex Magaz Graça
DS-1077	XMLUI & CLI always show a NullPointerException after running a Site-wide Curation Task	Tim Donohue	Tim Donohue
DS-1090	CC license process fails with java.lang.NegativeArraySizeException.	Peter Dietz	Dan Ishimitsu
DS-1094	Potential NPE error when mapping items	Scott Phillips	Scott Phillips

14 issues

Changes in DSpace 1.8.0

New Features in 1.8.0 (13 issues)

Key	Summary	Assignee	Reporter
DS-528	RSS feeds to support richer features, such as iTunes Podcast or Media RSS	Peter Dietz	Peter Dietz
	Marker ticket for developing a Sword client for DSpace.	Robin Taylor	Robin Taylor

Key	Summary	Assignee	Reporter
DS-602			
DS-638	check files on input for viruses, and verify file format	Robin Taylor	Jose Blanco
DS-673	My Archived Submissions in XMLUI	Tim Donohue	Brian Freels-Stendel
DS-737	Make launcher's classpath calculation available to external scripts	Mark H. Wood	Mark H. Wood
DS-749	allow for bitstream display order to be changed	Kevin Van de Velde (@mire)	Jose Blanco
DS-811	Delete / withdraw items via bulk csv editing	Stuart Lewis	Stuart Lewis
DS-848	Add MARCXML crosswalk for OAI-PMH	Robin Taylor	Timo Aalto
DS-903	Link Checker curaton task	Kim Shepherd	Stuart Lewis
DS-968	XML configurable workflow	Ben Bosman	Bram De Schouwer
DS-984	Provide links to RSS Feeds	Peter Dietz	Peter Dietz
DS-1001	DSpace 1.8: Add Curation Task Groups to GUI	Richard Rodgers	Wendy Bossons
DS-1005	SWORD v2 implementation for DSpace	Stuart Lewis	Stuart Lewis

13 issues

Improvements in 1.8.0 (45 issues)

Key	Summary	Assignee	Reporter
DS-514	Need to remove all release repository and pluginRepository entries from Maven poms.	Mark Diggory	Mark Diggory

Key	Summary	Assignee	Reporter
DS-615	Ability to perform maintenance on SOLR with solr.optimize	Ben Bosman	Peter Dietz
DS-690	Tidy up URL mapping for DisplayStatisticsServlet (JSPUI servlet that handles solr statistics)	Kim Shepherd	Kim Shepherd
DS-708	Deprecate & Remove old 'org.dspace.app.mets.METSExport' class, as it is obsolete	Tim Donohue	Tim Donohue
DS-715	Unification of license treatment in xmlui and jspui	Claudia Jürgen	Claudia Jürgen
DS-791	Add ability to disable the building of particular DSpace modules/interfaces from source code	Tim Donohue	Tim Donohue
DS-798	Czech localization of 1.7.0	Claudia Jürgen	Ivan Masár
DS-801	Bulgarian Translation for DSpace 1.7.0	Claudia Jürgen	Vladislav Zhivkov
DS-837	Translate file to spanish	Claudia Jürgen	Álvaro López
DS-839	Adding Field to Choice Authority to allow Authorities to be able to know field being required	Mark Diggory	Fabio Bolognesi
DS-840	Add Ability to create Top Level Community in at the home page.	Mark Diggory	Mark Diggory
DS-849	create a non-Porter Stemming analyzer for DSpace	Tim Donohue	Hardy Pottinger
DS-852	Split the Creative Commons and Licence steps into two seperate steps.	Robin Taylor	Robin Taylor
DS-854	Licenses on non-DSpace files have been replaced by DSpace boilerplate license	Tim Donohue	Peter Dietz
DS-857	CHANGES file now obsolete in SVN - point at online History	Tim Donohue	Tim Donohue
DS-862	Add ability to rename bitstreams (filenames) in XMLUI	Kim Shepherd	Kim Shepherd
	Upgrade DSpace XMLUI to use Spring 3.0.5.RELEASE		

Key	Summary	Assignee	Reporter
DS-884		Mark Diggory	Mark Diggory
DS-890	./bin/generate-sitemaps with ./bin/dspace generate-sitemaps	Mark H. Wood	Jason Stirnaman
DS-896	Improve Logging & XMLUI Error Handling of Curation Tools	Tim Donohue	Tim Donohue
DS-901	Add documentation about overlays	Robin Taylor	Alex Lemann

Showing 20 out of [45 issues](#)

Bugs Fixed in 1.8.0 (82 issues)

Key	Summary	Assignee	Reporter
DS-135	Withdrawn items displayed as "restricted" rather than withdrawn	Robin Taylor	Tim Donohue
DS-215	Single-argument Item.getMetadata does not work with mixed-case metadata	Stuart Lewis	Nicholas Riley
DS-401	Wrong date issued during submission	Robin Taylor	Claudia Jürgen
DS-435	UI cosmetics, "My Exports" displayed in navigation bar, when no user is logged in	Robin Taylor	Claudia Jürgen
DS-533	Collection Short Description not visible	Unassigned	Ronee Francis
DS-599	SOLR statistics file download displays all files and not only those in the Bundle Original	Kevin Van de Velde (@mire)	Claudia Jürgen
DS-612	Unfinished submissions see cc-rdf file instead of their uploaded PDF in the uploads step.	Richard Rodgers	Peter Dietz
DS-620	Exceed maximum while uploading files got the user stuck should lead to a friendly error page	Peter Dietz	Claudia Jürgen
DS-631	Wrong Parameter Name in web.xml comment	Mark H. Wood	Andy Smith

Key	Summary	Assignee	Reporter
DS-641	Page does not exist	Peter Dietz	Hardik Mishra
DS-642	IPAuthentication doesn't work with IPv6 addresses	Mark H. Wood	Stuart Lewis
DS-717	Duplicated template in Kubrick theme	Mark H. Wood	Mark H. Wood
DS-761	MetadataSchema: cache out of sync after calling delete()	Claudia Jürgen	Janne Pietarila
DS-764	OAI-PMH ListRecords false no result answer and missing resumptionToken	Unassigned	Claudia Jürgen
DS-768	All XMLUI Error Pages respond with 200 OK, instead of 404 Not Found	Kim Shepherd	Tim Donohue
DS-785	SWORD deposits fail when ingest events are fired if Discovery event consumer is configured	Kim Shepherd	Kim Shepherd
DS-789	HTTPS renders with errors due to a hardcoded HTTP link	Peter Dietz	Bram Luyten (@mire)
DS-793	missing # in Update Configuration Files	Claudia Jürgen	Jason Stirnaman
DS-806	Item.match() incorrect logic for schema testing	Stuart Lewis	Stuart Lewis
DS-808	jqueryUI javascript gets imported without corresponding CSS	Ben Bosman	Bram Luyten (@mire)

Showing 20 out of 82 issues

7.5.5 Changes in 1.7.x

- [Changes in DSpace 1.7.3](#)
- [Changes in DSpace 1.7.2](#)
- [Changes in DSpace 1.7.1](#)
- [Changes in DSpace 1.7.0](#)

Changes in DSpace 1.7.3

Improvements in 1.7.3 (2 issues)

Key	Summary	Assignee	Reporter
DS-1587	Update 1.7.x and 1.8.x branches for Git/GitHub	Tim Donohue	Tim Donohue
DS-1588	Update 1.7.x branch to build properly with Maven 3	Tim Donohue	Tim Donohue

[2 issues](#)

Bug Fixes

- [DS-1603](#) - Resolves a security issue in JSPUI

Changes in DSpace 1.7.2

Bugs Fixed in 1.7.2 (3 issues)

Key	Summary	Assignee	Reporter
DS-841	'IllegalArgumentException: No such column rnum' error in DSpace 1.7.x XMLUI admin eperson (with Oracle backend)	Peter Dietz	Hardy Pottinger
DS-871	XMLUI caches community / collection page which doesn't show a recently submitted item immediately	Peter Dietz	Peter Dietz
DS-875	DSpace Configuration service error when using "dspace" script.	Mark Diggory	Kevin Van de Velde (@mire)

[3 issues](#)

Changes in DSpace 1.7.1

Improvements in 1.7.1 (11 issues)

Key	Summary	Assignee	Reporter
DS-715	Unification of license treatment in xmlui and jspui	Claudia Jürgen	Claudia Jürgen
DS-720	Solr statistics documentation in DSpace manual and DSDOC is out-of-date, wrong, and inconsistent with dspace.cfg	Kim Shepherd	Kim Shepherd
DS-770	New Japanese messages for 1.7.0	Claudia Jürgen	Keiji Suzuki

Key	Summary	Assignee	Reporter
DS-798	Czech localization of 1.7.0	Claudia Jürgen	Ivan Masár
DS-801	Bulgarian Translation for DSpace 1.7.0	Claudia Jürgen	Vladislav Zhivkov
DS-828	Enable "restore mode" ingestion via sword	Scott Phillips	Scott Phillips
DS-837	Translate file to spanish	Claudia Jürgen	Álvaro López
DS-839	Adding Field to Choice Authority to allow Authorities to be able to know field being required	Mark Diggory	Fabio Bolognesi
DS-840	Add Ability to create Top Level Community in at the home page.	Mark Diggory	Mark Diggory
DS-856	Release DSpace Services version 2.0.3	Mark Diggory	Mark Diggory
DS-857	CHANGES file now obsolete in SVN - point at online History	Tim Donohue	Tim Donohue

11 issues

Bugs Fixed in 1.7.1 (20 issues)

Key	Summary	Assignee	Reporter
DS-215	Single-argument Item.getMetadata does not work with mixed-case metadata	Stuart Lewis	Nicholas Riley
DS-435	UI cosmetics, "My Exports" displayed in navigation bar, when no user is logged in	Robin Taylor	Claudia Jürgen
DS-620	Exceed maximum while uploading files got the user stuck should lead to a friendly error page	Peter Dietz	Claudia Jürgen
DS-641	Page does not exist	Peter Dietz	Hardik Mishra
DS-758	Mirage theme - lists of unfinished submission/workflow task wron link in collection column	Claudia Jürgen	Claudia Jürgen

Key	Summary	Assignee	Reporter
DS-761	MetadataSchema: cache out of sync after calling delete()	Claudia Jürgen	Janne Pietarila
DS-776	Collection admin cannot add bitstreams unless there is at least one bundle	Peter Dietz	Eija Airio
DS-785	SWORD deposits fail when ingest events are fired if Discovery event consumer is configured	Kim Shepherd	Kim Shepherd
DS-788	DSpace 1.7.0 only builds properly for Maven 2.2.0 or above	Unassigned	Tim Donohue
DS-789	HTTPS renders with errors due to a hardcoded HTTP link	Peter Dietz	Bram Luyten (@mire)
DS-793	missing # in Update Configuration Files	Claudia Jürgen	Jason Stirnaman
DS-806	Item.match() incorrect logic for schema testing	Stuart Lewis	Stuart Lewis
DS-808	jqueryUI javascript gets imported without corresponding CSS	Ben Bosman	Bram Luyten (@mire)
DS-809	Empty dc.abstract dim field (in mets XML) creates an empty span tag, causing page display errors in all Internet Explorer version	Ben Bosman	Bram Luyten (@mire)
DS-821	AbstractMETSIIngester creates an item before adding descriptive metadata	Tim Donohue	Stuart Lewis
DS-823	DatabaseManager is no longer Oracle compliant	Ben Bosman	Ben Bosman
DS-843	Autocomplete in authority control contains small errors in Mirage	Ben Bosman	Ben Bosman
DS-853	MetadataExposure settings for dc.description.provenance are ignored/overridden by XMLUI templates	Kim Shepherd	Kim Shepherd
DS-858	Multicore SOLR needs prevent remote access to solr cores	Mark Diggory	Kim Shepherd
	SWORD still uses dspace.url rather than dspace.baseUrl		

Key	Summary	Assignee	Reporter
DS-860		Kim Shepherd	Stuart Lewis

20 issues

Changes in DSpace 1.7.0

New Features in 1.7.0 (12 issues)

Key	Summary	Assignee	Reporter
DS-396	Provide metatags used by Google Scholar for enhanced indexing	Sands Fish	Sarah Shreeves
DS-466	Add ability to export/import entire Community/Collection/Item structure (for easier backups, migrations, etc.)	Tim Donohue	Tim Donohue
DS-525	Move item - inherit default policies of destination collection	Stuart Lewis	Stuart Lewis
DS-603	Having a most used item list similar to the recent submissions	Ben Bosman	Claudia Jürgen
DS-643	New testing framework (GSoC 2010)	Stuart Lewis	Stuart Lewis
DS-710	New Base Theme For DSpace 1.7.0	Ben Bosman	Ben Bosman
DS-711	Discovery release for XMLUI	Ben Bosman	Ben Bosman
DS-714	PowerPoint Text Extraction for DSpace Media Filter	Keith Gilbertson	Keith Gilbertson
DS-726	Modular Configuration (Curation)	Richard Rodgers	Richard Rodgers
DS-728	Curation System (Core Elements)	Richard Rodgers	Richard Rodgers
DS-730	Administrative UI for Curation (XMLUI)	Richard Rodgers	Richard Rodgers
	Tools for (load) testing		

Key	Summary	Assignee	Reporter
DS-733		Graham Triggs	Graham Triggs

12 issues

Improvements in 1.7.0 (50 issues)

Key	Summary	Assignee	Reporter
DS-192	Bitstreams should be returned ordered	Peter Dietz	Flávio Botelho
DS-387	Add ability for various Packager plugins to report their custom "options" via command line	Tim Donohue	Tim Donohue
DS-467	Consider making the JSPUI styles.css.jsp a static file	Stuart Lewis	Stuart Lewis
DS-550	Upgrade to latest Google Analytics tracking code	Stuart Lewis	Stuart Lewis
DS-557	LC Authority Names - Lookup Feature - names w/o dates	Kim Shepherd	Mark Diggory
DS-561	On login screen, keyboard input focus should be set to the first field (E-mail Address) so you don't have to use the mouse (JSPUI)	Andrea Bollini	Oleksandr Sytnyk
DS-571	Upgrade DSpace Services to next release	Mark Diggory	Mark Diggory
DS-577	Use modified Cocoon Servlet Service Impl in place of existing to support proper Cocoon Block addition.	Mark Diggory	Mark Diggory
DS-588	Patch for SFX (OpenURL resolver)	Jeffrey Trimble	Yin Yin Latt
DS-590	New INSTALL event when an Item is approved	Mark H. Wood	Mark H. Wood
DS-613	Error Handling in the XMLUI interface after session expired	Tim Donohue	Antero Neto
DS-618	Recommended versions of prerequisites becoming outdated	Mark H. Wood	Mark H. Wood

Key	Summary	Assignee	Reporter
DS-621	Export cleanup	Robin Taylor	Claudia Jürgen
DS-625	Bulgarian for DSpace 1.6.0	Claudia Jürgen	Vladislav Zhivkov
DS-628	Make the timeout for the extended resolver dnslookup configurable	Jeffrey Trimble	Claudia Jürgen
DS-646	Remove /bin scripts (replaced by 'dspace' command)	Jeffrey Trimble	Stuart Lewis
DS-647	Need Help Testing LNI refactoring changes in AIP Backup/Restore Work	Unassigned	Tim Donohue
DS-648	Modern Browsers are not identified in XMLUI main sitemap.xmap	Tim Donohue	Tim Donohue
DS-650	Bulgarian Localizatoin for DSpace 1.6.2	Claudia Jürgen	Vladislav Zhivkov
DS-653	Brazilian Portuguese (pt_BR) translation for XML-UI 1.6.2	Claudia Jürgen	Erick Rocha Fonseca

Showing 20 out of 50 issues

Bugs Fixed in 1.7.0 (89 issues)

Key	Summary	Assignee	Reporter
DS-63	xmlui hardcoded string in AuthenticationUtil.java - ID: 2088360	Mark H. Wood	Andrea Bollini
DS-123	xmlui browse in empty collection displays "Now showing items 1-0" of 0 - incorrect numbering	Scott Phillips	Keith Gilbertson
DS-242	Special groups shown for logged in user rather than for user being examined	Stuart Lewis	Stuart Lewis
DS-268	XMLUI Item Mapper cannot handle multiple words in search box	Stuart Lewis	Tim Donohue
DS-426	Item's submission license accessible without being configured to be public	Claudia Jürgen	Claudia Jürgen

Key	Summary	Assignee	Reporter
DS-431	Restricted Bitstream prompts for login, then forwards user to MyDSpace	Kim Shepherd	Tim Donohue
DS-469	DCDate.displayDate(false,*) displays only year	Mark H. Wood	Mark H. Wood
DS-471	Accessing site-level 'mets.xml' in XMLUI doesn't work properly for handle prefixes with periods (e.g. 2010.1)	Kim Shepherd	Tim Donohue
DS-493	Url in browser is incorrect after login	Ben Bosman	Ben Bosman
DS-494	DatabaseManager.process() unnecessarily limits range of DECIMAL or NUMERIC	Mark H. Wood	Mark H. Wood
DS-495	Broken link in the documentation section 8.2.3.	Jeffrey Trimble	Robin Taylor
DS-497	Date month and day get default values when user returns to describe form	Robin Taylor	Gabriela Mircea
DS-501	Kubrick Theme - NaN in Item Browse	Robin Taylor	Keith Gilbertson
DS-509	Retrieving country names in SOLR can return ArrayIndexOutOfBoundsException when country code is unchecked	Peter Dietz	Peter Dietz
DS-518	Duplicate listing of dependencies in dspace-sword/pom.xml	Stuart Lewis	Caryn N.
DS-527	Withdrawn items not shown as deleted in OAI	Kim Shepherd	John
DS-537	Malformed Japanese option values in the authority lookup window	Kim Shepherd	Keiji Suzuki
DS-538	restricted items are being returned in OAI GetRecord method while using harvest.includerestricted.oai	Ben Bosman	Ben Bosman
DS-539	Misspelled attribute in MODS/METS output	Keith Gilbertson	Andrew Hankinson
DS-543	Harvest not internationalized	Claudia Jürgen	Claudia Jürgen

Showing 20 out of 89 issues

7.5.6 Changes in 1.6.x

- [Changes in DSpace 1.6.2](#)
- [Changes in DSpace 1.6.1](#)
- [Changes in DSpace 1.6.0](#)

Changes in DSpace 1.6.2

Improvements in 1.6.2 (1 issues)

Key	Summary	Assignee	Reporter
DS-595	Czech localization of 1.6.1	Claudia Jürgen	Ivan Masár

1 issue

Bugs Fixed in 1.6.2 (7 issues)

Key	Summary	Assignee	Reporter
DS-516	DSRUN does not start Service Manager	Stuart Lewis	Mark Diggory
DS-584	start-handle-server script broken - Error in launcher.xml: Invalid class name	Unassigned	Keith Gilbertson
DS-604	Errors in 1.5.x -> 1.6.x and 1.6.0 - 1.6.1 upgrade steps	Jeffrey Trimble	Kim Shepherd
DS-607	Invalid identifiers not escaped	Stuart Lewis	Stuart Lewis
DS-608	Batch metadata import missing item headers	Stuart Lewis	Stuart Lewis
DS-609	update-handle-prefix wrong in docs	Jeffrey Trimble	Stuart Lewis
DS-610	SyndicationFeed expects dc.date.issued to be available as a java.util.Date	Robin Taylor	Robin Taylor

7 issues

Changes in DSpace 1.6.1

Improvements in 1.6.1 (7 issues)

Key	Summary	Assignee	Reporter
DS-430	Embargo	Jeffrey Trimble	Jim Ottaviani
DS-500	Ukrainian for DSpace 1.6.0	Claudia Jürgen	Serhij Dubyk
DS-508	Attachment spelled as attachement in DailyReportEmailer	Stuart Lewis	Keith Gilbertson
DS-534	Documentation for "schema" attribute in metadata xml files	Jeffrey Trimble	Keith Gilbertson
DS-557	LC Authority Names - Lookup Feature - names w/o dates	Kim Shepherd	Mark Diggory
DS-571	Upgrade DSpace Services to next release	Mark Diggory	Mark Diggory
DS-577	Use modified Cocoon Servlet Service Impl in place of existing to support proper Cocoon Block addition.	Mark Diggory	Mark Diggory

7 issues

Bugs Fixed in 1.6.1 (37 issues)

Key	Summary	Assignee	Reporter
DS-239	java.net.MalformedURLException: unknown protocol: resource	Mark Diggory	Mark Diggory
DS-242	Special groups shown for logged in user rather than for user being examined	Stuart Lewis	Stuart Lewis
DS-295	CC License being assigned incorrect Mime Type during submission.	Jeffrey Trimble	Steven Williams
DS-471	Accessing site-level 'mets.xml' in XMLUI doesn't work properly for handle prefixes with periods (e.g. 2010.1)	Kim Shepherd	Tim Donohue
DS-483	statistics.item.authorization.admin ignored by xmlui	Ben Bosman	Claudia Jürgen
DS-493	Url in browser is incorrect after login	Ben Bosman	Ben Bosman

Key	Summary	Assignee	Reporter
DS-497	Date month and day get default values when user returns to describe form	Robin Taylor	Gabriela Mircea
DS-501	Kubrick Theme - NaN in Item Browse	Robin Taylor	Keith Gilbertson
DS-506	embargo-lifter command missing from launcher.xml	Stuart Lewis	Stuart Lewis
DS-507	Log Converter difference between docs (log-converter) and launcher (stats-log-converter)	Jeffrey Trimble	Peter Dietz
DS-509	Retrieving country names in SOLR can return ArrayIndexOutOfBoundsException when country code is unchecked	Peter Dietz	Peter Dietz
DS-513	Connection leak in SWORD authentication process	Andrea Bollini	Andrea Bollini
DS-516	DSRUN does not start Service Manager	Stuart Lewis	Mark Diggory
DS-518	Duplicate listing of dependencies in dspace-sword/pom.xml	Stuart Lewis	Caryn N.
DS-523	Reordering of 1.5 -> 1.6 upgrade steps in DSpace manual	Jeffrey Trimble	Stuart Lewis
DS-526	ItemUpdate - script and manual updates	Jeffrey Trimble	Stuart Lewis
DS-527	Withdrawn items not shown as deleted in OAI	Kim Shepherd	John
DS-537	Malformed Japanese option values in the authority lookup window	Kim Shepherd	Keiji Suzuki
DS-538	restricted items are being returned in OAI GetRecord method while using harvest.includerestricted.oai	Ben Bosman	Ben Bosman
DS-539	Misspelled attribute in MODS/METS output	Keith Gilbertson	Andrew Hankinson

Showing 20 out of [37 issues](#)

Changes in DSpace 1.6.0

New Features in 1.6.0 (20 issues)

Key	Summary	Assignee	Reporter
DS-161	Bulk Metadata Editing	Jeffrey Trimble	Charles Kiplagat
DS-194	Give METS ingester configuration option to make use of collection templates	Stuart Lewis	Stuart Lewis
DS-195	Allow the primary bitstream to be set in the item importer / exporter	Stuart Lewis	Stuart Lewis
DS-204	New -zip option for item exporter and importer	Unassigned	Stuart Lewis
DS-205	Creative Commons - option to set legal jurisdiction	Unassigned	Stuart Lewis
DS-228	Community Admin XMLUI: Delegated Admins Patch	Andrea Bollini	Tim Donohue
DS-236	Authority Control, and plug-in choice control for Metadata Fields	Jeffrey Trimble	Larry Stone
DS-247	Contribution of @MIRE Solr Based Statistics Engine to DSpace.	Mark Diggory	Mark Diggory
DS-288	Hide metadata from full item view	Larry Stone	Claudia Jürgen
DS-289	OAI-PMH + OAI-ORE harvesting support	Jeffrey Trimble	Scott Phillips
DS-317	Embargo feature	Jeffrey Trimble	Stuart Lewis
DS-321	DSpace command launcher	Jeffrey Trimble	Stuart Lewis
DS-323	ItemUpdate - new feature to batch update metadata and bitstreams	Jeffrey Trimble	Richard Rodgers (OLD acct)
DS-324	Add support for OpenSearch syndicated search conventions	Jeffrey Trimble	Richard Rodgers (OLD acct)
DS-330	Create new session on login / invalidate sessions on logout	Stuart Lewis	Stuart Lewis

Key	Summary	Assignee	Reporter
DS-359	Add alternate file appender for log4j	Graham Triggs	Graham Triggs
DS-363	JSPUI tags/views for @mire Solr statistics module	Kim Shepherd	Kim Shepherd
DS-377	Add META tags identifying DSpace source version to Web UIs	Larry Stone	Larry Stone
DS-388	Item importer - new option to enable workflow notification emails	Jeffrey Trimble	Stuart Lewis
DS-447	Email test script	Jeffrey Trimble	Stuart Lewis

20 issues

Improvements in 1.6.0 (46 issues)

Key	Summary	Assignee	Reporter
DS-52	Factor out common webapp installation - ID: 2042160	Mark H. Wood	Charles Kiplagat
DS-196	METS exposed via OAI-PMH includes description.provenance information	Stuart Lewis	Stuart Lewis
DS-201	handle.jar 6.2 needs adding to DSpace Maven repository	Mark Diggory	Stuart Lewis
DS-213	IPAuthentication extended to allow negative matching	Stuart Lewis	Stuart Lewis
DS-219	Internal Server error - include login details of user	Stuart Lewis	Vanessa Newton-Wade
DS-221	XMLUI 'current activity' recognises Google Chrome as Safari	Stuart Lewis	Stuart Lewis
DS-234	Configurable passing of Javamail parameter settings	Stuart Lewis	Stuart Lewis
DS-238	Move item function in xmlui	Unassigned	Stuart Lewis

Key	Summary	Assignee	Reporter
DS-241	DSpace Assembly Improvement	Mark Diggory	Mark Diggory
DS-251	Bulk Metadata Editing: XMLUI aspect and forms	Kim Shepherd	Kim Shepherd
DS-252	Interpolate variables in the Subject: line of email templates as well	Stuart Lewis	Larry Stone
DS-261	Community Admin JSPUI: porting of the DS-228 patch	Andrea Bollini	Andrea Bollini
DS-270	Make delegate admin permissions configurable	Jeffrey Trimble	Andrea Bollini
DS-271	Make the OAI DC crosswalk configurable	Unassigned	Andrea Bollini
DS-291	README update for top level of dspace 1.6.0 package directory	Stuart Lewis	Van Ly
DS-297	Refactor SQL source and Ant script to avoid copying Oracle versions over PostgreSQL	Larry Stone	Larry Stone
DS-299	Allow long values to be specified for the max upload request (for uploading files greater than 2Gb)	Graham Triggs	Stuart Lewis
DS-306	Option to disable mailserver	Jeffrey Trimble	Ben Bosman
DS-307	Offer access in AbstractSearch to QueryResults for subclasses	Ben Bosman	Ben Bosman
DS-308	documentation on an added optional configuration parameter	Jeffrey Trimble	Ben Bosman

Showing 20 out of [46 issues](#)

Bugs Fixed in 1.6.0 (102 issues)

Key	Summary	Assignee	Reporter
DS-44	Monthly statistics skip first and last of month - ID: 2541435	Stuart Lewis	Charles Kiplagat

Key	Summary	Assignee	Reporter
DS-114	Links not working due to trailing white space in dspace.url	Claudia Jürgen	Claudia Jürgen
DS-118	File preview link during submission leads to page not found	Claudia Jürgen	Claudia Jürgen
DS-121	XMLUI Feedback form breaks with multiple hostnames	Kim Shepherd	Keith Gilbertson
DS-123	xmlui browse in empty collection displays "Now showing items 1-0" of 0 - incorrect numbering	Scott Phillips	Keith Gilbertson
DS-128	Anchor in submission doesn't work	Larry Stone	Andrea Bollini
DS-156	File description not available in XMLUI	Stuart Lewis	Samuel Ottenhoff
DS-191	metadataschemaregistry_seq is not initialized correctly under Oracle	Stuart Lewis	Larry Stone
DS-193	OAI RDF crosswalk fails when DC value is null	Stuart Lewis	Larry Stone
DS-197	Deleting a primary bitstream does not clear the primary_bitstream_id on the bundle table	Claudia Jürgen	Graham Triggs
DS-198	File descriptions can not be removed/cleared in XMLUI	Unassigned	Kim Shepherd
DS-199	SWORD module doesn't accept X-No-Op header (dry run)	Unassigned	Claudio Venturini
DS-200	SWORD module requires the X-Packaging header	Stuart Lewis	Claudio Venturini
DS-206	Input form visibility restriction doesn't work properly	Andrea Bollini	Andrea Bollini
DS-209	Context.java turnOffAuthorisationSystem() can throw a NPE	Stuart Lewis	Stuart Lewis
DS-212	NPE thrown during Harvest of non-items when visibility restriction is enabled	Stuart Lewis	Stuart Lewis
	Migrating items that use additional metadata schemas causes an NPE	Unassigned	

Key	Summary	Assignee	Reporter
DS-216			Stuart Lewis
DS-217	Hardcoded String in the license bitstream	Andrea Bollini	Andrea Bollini
DS-218	Cannot add/remove email subscriptions from Profile page in XMLUI	Tim Donohue	Tim Donohue
DS-222	Email alerts due to internal errors are not sent, if context is missing	Claudia Jürgen	Claudia Jürgen

Showing 20 out of 102 issues

7.5.7 Changes in 1.5.x

- [Changes in DSpace 1.5.2](#)
- [Changes in DSpace 1.5.1](#)
- [Changes in DSpace 1.5.0](#)

Changes in DSpace 1.5.2

New Features in 1.5.2 (3 issues)

Key	Summary	Assignee	Reporter
DS-48	shibboleth+dspace1.5.1 patch - ID: 2412723	Mark Diggory	Charles Kiplagat
DS-108	Usage event (statistics) Plugin hook for 1.5 (SF 2025998)	Mark H. Wood	Bradley McLean
DS-214	Catalan translation	Claudia Jürgen	Centre de Supercomputació de Catalunya

3 issues

Improvements in 1.5.2 (38 issues)

Key	Summary	Assignee	Reporter
DS-4	Refactor LDAPServlet to use Stackable Authentication - ID: 2057231	Stuart Lewis	Charles Kiplagat
DS-11	'My Account' disappears following exports - ID: 2495728	Stuart Lewis	Charles Kiplagat
DS-13	Fix for bug [1774958] Nested folders do not export correctly - ID: 2513300	Stuart Lewis	Charles Kiplagat
DS-16	Hierarchical LDAP support - ID: 2057378	Stuart Lewis	Charles Kiplagat
DS-19	Feature Request #1896717 Registration notification missin - ID: 2041754	Stuart Lewis	Charles Kiplagat
DS-21	Fix for hardcoded metadata language qualifiers - ID: 2433387	Claudia Jürgen	Charles Kiplagat
DS-30	Hardcoded String in jspui browse - ID: 2526153	Claudia Jürgen	Charles Kiplagat
DS-31	Bug 2512868 Double quote problem in some fields of JSPUI - ID: 2525942	Claudia Jürgen	Charles Kiplagat
DS-34	Add File Format Descriptions to XMLUI 1.5.x - ID: 2433852	Unassigned	Charles Kiplagat
DS-35	Enable Google Sitemaps for XMLUI - ID: 2462293	Unassigned	Charles Kiplagat
DS-36	DSpace 1.5 XMLUI - Enable METS <amdSec> using crosswalks - ID: 2477820	Unassigned	Charles Kiplagat
DS-39	Fix for toDate method in DCDate - ID: 2385187	Stuart Lewis	Charles Kiplagat
DS-45	Messages_th.properties for DSpace 1.5.1 JSPUI - ID: 2540683	Unassigned	Charles Kiplagat
DS-46	Bug 1617889 Years < 1000 do not display in simple item view - ID: 2524083	Andrea Bollini	Charles Kiplagat
DS-47	Add support for rendering DOI links in JSPUI (1.4, 1.5) - ID: 2521493	Andrea Bollini	Charles Kiplagat
	Italian translation xmlui - ID: 1984513		

Key	Summary	Assignee	Reporter
DS-78		Andrea Bollini	Charles Kiplagat
DS-85	XMLUI Cocoon logs should not be stored under [xmlui-webapp]/WEB-INF/logs/	Unassigned	Tim Donohue
DS-87	XMLUI file download links break in Google search results if file 'sequence' number changes.	Tim Donohue	Tim Donohue
DS-93	Upgrade XMLUI to Cocoon 2.2	Mark Diggory	Mark Diggory
DS-94	Verify Configuration Options are still applicable with the Cocoon User community.	Mark Diggory	Mark Diggory

Showing 20 out of 38 issues

Bugs Fixed in 1.5.2 (75 issues)

Key	Summary	Assignee	Reporter
DS-1	Controlled vocab:he URL http://localhost:8080/jspui/subject-search gives internal error and a stack trace in dspace.log	Andrea Bollini	Andrew Peter Marlow
DS-2	"Not found" page returns 200 OK instead of 404 Not Found - ID: 2002866	Mark H. Wood	Charles Kiplagat
DS-5	DSpace1.5.1(XML) problem with Login to restricted bitstreams - ID: 2164955	Stuart Lewis	Charles Kiplagat
DS-6	XHTML Head Dissimination Crosswalk exposes provenance info - ID: 2343281	Stuart Lewis	Charles Kiplagat
DS-7	HTML tags not stripped in statistics display - ID: 1896225	Stuart Lewis	Charles Kiplagat
DS-8	DSpace Home link style in breadcrumb trail - ID: 1951859	Stuart Lewis	Charles Kiplagat
DS-9	Restricted Items metadata exposed via OAI - ID: 1730606	Stuart Lewis	Charles Kiplagat
	Implicit group for all registered users - ID: 1587270		

Key	Summary	Assignee	Reporter
DS-10		Stuart Lewis	Charles Kiplagat
DS-12	Exception handling for deleting a metadata field - ID: 1606439	Stuart Lewis	Charles Kiplagat
DS-14	xmlui Administrative log in as another eperson - ID: 2086481	Stuart Lewis	Charles Kiplagat
DS-15	Submission verify page handles dc.identifier.* incorrectly - ID: 2155479	Unassigned	Charles Kiplagat
DS-17	DSpace 1.5 Controlled Vocab (edit-metadata.jsp) - ID: 1931796	Stuart Lewis	Charles Kiplagat
DS-18	DSpace 1.5.1(XMLUI) Wrong dir usage of StatisticsLoader - ID: 2137425	Stuart Lewis	Charles Kiplagat
DS-20	2 Authentications with LoginPage cause connection exhaust - ID: 2352146	Claudia Jürgen	Charles Kiplagat
DS-22	News stored not language dependend - ID: 2125833	Unassigned	Charles Kiplagat
DS-23	DSQuery invalid check for empty query string - ID: 2343849	Unassigned	Charles Kiplagat
DS-24	Error in authorization to submit when you add collection. - ID: 1725817	Unassigned	Charles Kiplagat
DS-25	SWORD Service Document fails if Collection is untitled - ID: 1968082	Stuart Lewis	Charles Kiplagat
DS-26	Hardcoded Strings in DSQuery - ID: 2493794	Claudia Jürgen	Charles Kiplagat
DS-27	NullPointerException possible in review.jsp - ID: 1571645	Claudia Jürgen	Charles Kiplagat

Showing 20 out of [75 issues](#)

Changes in DSpace 1.5.1

General Improvements and Bug Fixes in 1.5.1

(Scott Philips)

- (Scott Phillips) Fixed bug where users could not finish registering nor reset their password because the authentication method signatures were changed.
- Jay Paz (SF#1898241) Additional fixes to patch to enable reuse of methods.
- Added the ability to manage sessions with site wide alerts to prevent users from authenticating.
- Fixes a bug where the ability to edit an item during workflow step 2 is not displayed.
- Jay Paz (SF#1898241) Add item Export from jspui and xmlui.
- Added easy support for google analytics statistics
- Added the ability for super admins to login as other users.
- Added an activity viewer to the Control Panel

(Mark Diggory)

- Fix for SF Bug #2082236 Subscription notification (sub-daily) no emails sent
- #2102580 William Hays: Duplicate Handle exception when replacing bitstreams
- #2102617 Sands Fish: X509Authentication fails to assign appropriate specialgroups
- (Sands Fish) Add "Select Primary Bitstream" functionality to submission workflow
- Guard against Community/Collection metadata having only whitespace characters and eliminate cases where null pointer exceptions would be thrown
- Improve DSIndexer logic in both branches to support removal of items from index when withdrawn from repository.
- (Sands Fish) Provides fix for AuthenticationUtil where users ID's are not properly compared.
- Fix NullPointerException cause by nullified Context object in LNI map item to new collection.
- Block Basic Authentication "details" from being exposed in dspace logs.
- (Bill Hays) Close InputStreamReaders explicitly to release any file handles back to OS.
- correct linking on pages when xmlui is the ROOT webapplication
- correct issue with sitemap redirection of mydspace uri.
- Add serlet-api to overlay wars to reduce compile time errors when adding classes
- Correct issues in feed generation
- XMLUI Adjust Advanced Search to use search properties from dspace.cfg.
- Correct bug in Body.toSAX where startElement is called instead of end element.
- Correct issue with libraries being excluded from wars

(Claudia Juergen)

- Fix for SF bug #2090761 Statistics wrong use of dspace.dir for log location
- Fix for SF bug #2081930 xmlui hardcoded strings in EditGroupForm.java
- Fix for SF bug #2080319 jspui hardcoded strings in browse
- Fix for SF bug #2078305 xmlui hardcoded strings used in UI in xmlui-api
- Fix for SF bug #2078324 xmlui hardcoded strings used in UI in General-Handler.xml
- SF patch #2076066 Review in jspui submission non-dc metadata
- SF Bug #1983859 added Foreign Lucene Analyzers to poms
- SF Bug #1989916 - missing LDAP authentication key

(Stuart Lewis)

- #1947036 Patch for SF Bug1896960 SWORD authentication and LDAP + 1989874 LDAPAuthentication pluggable method broken for current users
- Added copying of registration email template to 1.4 to 1.5 upgrade instructions
- Fix for SF bug #2055941 LDAP authentication fails for new users in SWORD and Manakin

(Zuki Ebetsu / Stuart Lewis)

- #1990660 SWORD Service Document are malformed / Corrected Atom publishing MIME types

(Stuart Lewis / Claudia Juergen)

- Updated installation and configuration documents for new statistics script, and removed references to Perl

(Tim Donohue)

- Fix for SF bug #2095402 - Non-interactive Submission Steps don't work in JSPUI 1.5
- Fix for SF bug #2013921 - Movement in Submission Workflow Causes Skipped Steps
- Fix for SF bug #2015988 - Configurable Submission bug in SubmissionController
- Fix for SF bug #2034372 - Resorting Search Results in JSPUI always gives no results
- Updates to Community/Collection Item Counts (i.e. strengths) for XMLUI.
- 1.5 upgrade instructions were missing Metadata Registry updates necessary to support SWORD.

(Graham Triggs)

- Fix various problems with resources potentially not being freed, and other minor fixes suggested by FindBugs
- Replace URLEncoder with StringEscapeUtils for better fix of escaping the hidden query field
- Fix #2034372 - Resorting in JSPUI gives no results
- Fix #1714851 - set eperson.subscription.onlynew in dspace.cfg to only include items that are new to the repository
- Fix issue where the browse and search indexes will not be updated correctly if you move an Item
- Fix problem with SWORD not accepting multiple concurrent submissions
- Fix #1963060 Authors listed in reverse order
- Fix #1970852 - XMLUI: Browse by Issue Date "Type in Year" doesn't work
- Statistics viewer for XMLUI, based on existing DStat. Note that this generates the view from the analysis files (.dat), does not require HTML report generation.
- Fixed incorrect downloading of bitstream on withdrawn item
- Add JSPUI compatible log messages to XMLUI transformers
- Clean up use of ThreadLocal
- Improved cleanup of database resources when web application is unloaded
- Fix bug #1931799 - duplicate "FROM metadatavalue"
- Fixed Oracle bugs with ILIKE operators and LIMIT/OFFSET clauses

Changes in DSpace 1.5.0

General Improvements in 1.5.0

- Highly configurable and theme-able new user interface (Manakin).
- Apache Maven-based modular build system.
- LNI (Lightweight Network Interface) service. Allows programmatic ingest of content via WebDAV or SOAP.
- SWORD (Simple Web-service Offering Repository Deposit): repository-standard ingest service using Atom Publishing Protocol.
- Highly configurable item web submission system. All submission steps are configurable not just metadata pages.
- Browse functionality allowing customisation of the available indexes via `dspace.cfg` and pluggable normalisation of the sort strings. Integration with both JSP-UI and XML-UI included.
- Extensible content event notification service.
- Generation of Google and HTML sitemaps

Bug fixes and smaller patches in 1.5.0

- New options for ItemImporter to support bitstream permissions and descriptions.
- 1824710 Fix - Change in Creative Commons RDF.
- 1794700 Fix - Stat-monthly and stat-report-monthly
- 1566820 Patch - Authentication code moved to new `org.dspace.authenticate` package, add IP AUth
- 1670093 Patch - More stable metadata and schema registry import Option to generate community and collection "strength" as a batch job
- 1659868 Patch - Improved database level debugging
- 1620700 Patch - Add Community and Sub-Community to OAI Sets
- 1679972 Fix - OAIDCCrosswalk NPE and invalid character fix, also invalid output prevented
- 1549290 Fix - Suggest Features uses hard coded strings
- 1727034 Fix - Method `MetadataField.unique()` is incorrect for null values
- 1614546 Fix - Get rid of unused `mets_bitstream_id` column
- 1450491 Patch - i18n configurable multilingualism support
- 1764069 Patch - Replace "String" with "Integer" in `PreparedStatement` where needed
- 1743188 Patch - for Request #1145499 - Move Items
- 179196 Patch - Oracle SQL in Bitstream Checker
- 1751638 Patch - Set `http disposition` header to force download of large bitstreams
- 1799575 Patch - New `EPersonConsumer` event consumer
- 1566572 Patch - Item metadata in XHTML head elements
- 1589429 Patch - "Self-Named" Media Filters (i.e. `MediaFilter` Plugins) (updated version of this patch)
- 1888652 Patch - Statistics Rewritten In Java
- 1444364 Request - Metadata registry exporter
- 1221957 Request - Admin browser for withdrawn items

- 1740454 Fix - Concurrency
- 1552760 Fix - Submit interface looks bad in Safari
- 1642563 Patch - bin/update-handle-prefix rewritten in Java
- 1724330 Fix - Removes "null" being displayed in community-home.jsp
- 1763535 Patch - Alert DSpace administrator of new user registration
- 1759438 Patch - Multilingualism Language Switch - DSpace Header

7.5.8 Changes in 1.4.x

- [Changes in DSpace 1.4.1](#)
- [Changes in DSpace 1.4.0](#)

Changes in DSpace 1.4.1

General Improvements in 1.4.1

- 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 in 1.4.1

- 1604037 - UIUtil.encodeBitstream() now correctly encodes URLs (no longer incorrectly substitutes '+' for spaces in non-query segment)
- 1592984 - Date comparisons strip time in org.dspace.harvest.Harvest
- 1589902 - Duplicate field checking error on input-forms.xml
- 1596952 - Collection Wizard create Template missing schema
- 1596978 - View unfinished submissions - collection empty
- 1588625 - Incorrect text on item mapper screen
- 1597805 - DIDL Crosswalk: wrong resource management
- 1605635 - NPE in Utils.java
- 1597504 - Search result page shows shortened query string
- 1532389 - Item Templates do not work for non-dc fields
- 1066771 - Metadata edit form dropping DC qualifier
- 1548738 - Multiple Metadata Schema, schema not shown on edit item page
- 1589895 - Not possible to add unqualified Metadata Field
- 1543853 - Statistics do not work in 1.4
- 1541381 - Browse-by-date and browse-by-title not working
- 1556947 - NullPointerException when no user selected to del/edit
- 1554064 - Fix exception handling for ClassCastException in BitstreamServlet
- 1548865 - Browse errors on withdrawn item
- 1554056 - Community/collection handle URL with / redirects to homepage
- 1571490 - UTF-8 encoded characters in licence
- 1571519 - UTF-8 in statistics
- 1544807 - Browse-by-Subject/Author paging mechanism broken
- 1543966 - "Special" groups inside groups bug
- 1480496 - Cannot turn off "ignore authorization" flag!
- 1515148 - Community policies not deleting correctly
- 1556829 - Docs mention old SiteAuthenticator class
- 1606435 - Workflow text out of context
- Fix for bitstream authorization timeout
- Fix to make sure cleanup() doesn't fail with NullPointerException
- Fix for removeBitstream() failing to update primary bitstream
- Fix for Advanced Search ignoring conjunctions for arbitrary number of queries
- Fix minor bug in Harvest.java for Oracle users

- Fix missing title for news editor page
- Small Messages.properties modification (change of DSpace copyright text)
- fix PDFBox tmp file issue
- Fix HttpServletRequest encoding issues
- Fix bug in TableRow toString() method where NPE is thrown if tablename not set
- Update DIDL license and change coding style to DSpace standard

Changes in DSpace 1.4.0

General Improvements in 1.4.0

- Content verification through periodic checksum checking
- Support for branded preview image
- Add/replace Creative Commons in 'edit item' tool
- Customisable item listing columns and browse indices
- Script for updating handle prefixes (e.g. for moving from development to production)
- Configurable boolean search operator
- Controlled vocabulary patch to provide search on classification terms, and addition of terms during submission.
- Add 'visibility' element to input-forms.xml
- Browse by subject feature
- Log4J enhancement to use XML configuration
- QueryArgs class can support any number of fields in advanced search.
- Community names no longer have to be unique
- Enhanced Windows support
- Support for multiple (flat) metadata schemas
- Suggest an item page
- RSS Feeds
- Performance enhancements
- Stackable authentication methods
- Plug-in manager
- Pluggable SIP/DIP support and metadata crosswalks
- Nested groups of e-people
- Expose METS and MPEG-21 DIDL DIPs via OAI-PMH
- Configurable Lucene search analyzer (e.g. for Chinese metadata)
- Support for SMTP servers requiring authentication

Bug fixes in 1.4.0

- 1358197 - Edit Item, empty DC fields not removable
- 1363633 - Submission step 1 fails when there are no collections
- 1255264 - Resource policy eperson value was set to wrong column
- 1380494 - Error deleting an item with multiple metadata schema support

- 1443649 - Cannot configure unqualified elements for advanced search index
- 1333687 - Browse-(title|date) fails on withdrawn item
- 1066713 - Two (sub)communities cannot have one name
- 1284055 - Two Communities of same name throws error
- 1035366 - Bitstream size column should be bigint
- 1352257 - Selecting a Group for GroupToGroup while Creating Collection
- 1352226 - Navigation and Sorting in Group List (Select Groups) fails
- 1348276 - Null in collection name causes OAI ListSets to fail
- 1160898 - dspace_migrate removes Date.Issued from prev published items
- 1261191 - Malformed METS metadata exported

7.5.9 Changes in 1.3.x

- [Changes in DSpace 1.3.2](#)
- [Changes in DSpace 1.3.1](#)
- [Changes in DSpace 1.3.0](#)

Changes in DSpace 1.3.2

General Improvements in 1.3.2

- DSpace UI XHTML/WAI compliant
- Configure metadata fields shown on simple item display
- Supervisor/workspace help documentation

Bug fixes in 1.3.2

- 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 in 1.3.1

- 1252153 - Error on fresh install

Changes in DSpace 1.3.0

General Improvements in 1.3.0

- Initial i18n Support for JSPs - Note: the implementation of this feature required changes to almost all JSP pages
- LDAP authentication support
- Log file analysis and report generation
- Configurable item licence viewing
- Supervision order/collaborative workspace administrative tools
- Basic workspace for submissions in progress, with support for supervision
- SRB storage system option
- Updated handle server system
- Database optimisations
- Latest versions of Xerces, Xalan and OAICAT jars
- Various documentation additions and cleanups

Bug fixes in 1.3.0

- 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

7.5.10 Changes in 1.2.x

- [Changes in DSpace 1.2.2](#)
- [Changes in DSpace 1.2.1](#)
- [Changes in DSpace 1.2.0](#)

Changes in DSpace 1.2.2

General Improvements in 1.2.2

- Customisable submission forms added
- Configurable number of index terms in Lucene for full-text indexing
- Improved scalability in media filter
- Submit button on collection pages only appears if user has authorisation
- PostgreSQL 8.0 compatibility
- Search scope retention to improve browsing
- Community and collection strengths displayed
- Upgraded OAICat software

Bug fixes in 1.2.2

- 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 in 1.2.2

- *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.jspchanged*
- *submit/progressbar.jspchanged*
- *submit/review.jspchanged*
- *submit/select-collection.jspchanged*
- *submit/show-license.jspchanged*
- *submit/show-uploaded-file.jspchanged*
- *submit/upload-error.jspchanged*
- *submit/upload-file-list.jspchanged*

Changes in DSpace 1.2.1

General Improvements in 1.2.1

- 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 in 1.2.1

- 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 in 1.2.1

- *collection-homechanged*
- *community-homechanged*
- *display-itemchanged*
- *dspace-admin/confirm-delete-collectionmoved to tools/ and changed*
- *dspace-admin/confirm-delete-communitymoved to tools/ and changed*
- *dspace-admin/edit-collectionmoved to tools/ and changed*
- *dspace-admin/edit-communitymoved to tools/ and changed*
- *dspace-admin/indexchanged*

- *dspace-admin/upload-logochanged*
- *dspace-admin/wizard-basicinfochanged*
- *dspace-admin/wizard-default-itemchanged*
- *dspace-admin/wizard-permissionschanged*
- *dspace-admin/wizard-questionschanged*
- *help/formats.htmlremoved*
- *help/formatschanged*
- *indexchanged*
- *layout/navbar-adminchanged*

Changes in DSpace 1.2.0

General Improvements in 1.2.0

- 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 `cvcs -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`

7.5.11 Changes in 1.1.x

- [Changes in DSpace 1.1.1](#)
- [Changes in DSpace 1.1](#)

Changes in DSpace 1.1.1

Bug fixes in 1.1.1

- 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 DSspaces 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 in 1.1.1

- `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.)

7.6 DSpace Item State Definitions

Workspace item

An item that is under submission and active edit by an authorized user. The workspace item is visible only to the submitter and the system administrators. (Currently there is no simple way to find/browse such items other than with the direct item ID or to use the supervisor functionality). Using the supervisor functionality, a system admin can allow other authorized user to see/edit the item in the workspace state.

Expected use cases:

- Self deposit

- Collaboration over an in-progress submission for a small group of researchers. (This use case is implemented only with major limitations, using the supervision feature – concurrency, lack of delegation: supervision must be defined by the system administrators, etc.)

Workflow Item

An item that is under review for quality control and policy compliance. The workflow item is visible to the original submitter (currently only basic metadata are visible out-of-box in the mydspace summary list), users assigned to the specific workflow step where the item resides, and system administrators. (Currently there is no simple way to find/browse such items other than with the direct item ID or to use the abort workflow functionality).

Expected use cases:

- Quality control
- Improvements to the bibliographic record (metadata available in workflow can be different than those asked of the submitter)
- Check of policy / copyright

Withdrawn item

It is a logical deletion. The Item can be restored and it can be used to keep track of what has been available for a while on the public site.

Expected use cases:

- Staging area for item to be removed when copyright issues arise with publisher. If the copyright issue is confirmed, the item will be permanently deleted or kept in the withdrawn state for future reference.
- Logical deletion delegated to community/collection admin, where permanent deletion is reserved to system administrators
- Logical deletion, where permanent deletion is not an option for an organization
- Removal of an old version of an item, forcing redirect to a new up-to-date version of the item (this use case is not currently implemented out-of-box in DSpace, see)

Private item

This state should only refer to the discoverable nature of the item. A private item will not be included in any system that aims to help users to find items. So it will not appear in:

- Browse
- Recent submission
- Search result
- OAI-PMH (at least for the ListRecords and ListIdentifiers verb; though the OAI-PMH specification is not clear about inconsistent implementation of the ListRecords and GetRecord verb)

- REST list and search methods

It should be accessible under the actual ACL rules of DSpace using direct URL or query method such as:

- Splash page access (i.e. /handle/<xxxxx>/<yyyyy>)
- OAI-PMH GetRecord verb
- REST direct access /rest/item/<item-id> or equivalent

Expected use cases:

- Provide a light rights awareness feature where discovery is not enabled for search and/or browse
- Hide “special items” such as repository presentations, guides or support materials
- Hide an old version of an Item in cases where real versioning is not appropriate or liked
- Hide specific types of item such as “Item used to record Journal record: Journal Title, ISSN, Publisher etc .” used as authority file for metadata (dc.relation.ispartof) of “normal item”

Archived/Published item

An item that is in a stable state, available in the repository under the defined ACL rule. Changes to these items are possible only for a restricted group of users (administrators) and should produce versioning according to the Institution's policy.

Embargoed Item (as in DSpace 3.0+):

Are a special case of Archived/Published Item. The item has some time based access policy attached to it and/or the underlying bitstreams. Specifically, read permission for someone (EPerson Group) starting from a defined date. Typically embargo is applied to the bitstreams so that "fulltext" has initially very limited access (normally administrators or other "repository staff" groups) and only after a defined date will the fulltext become visible to all users (Anonymous group). This scenario is used to implement typical "embargo requirements" from publishers -- see [Delayed Open Access](#).

If the metadata of the item should be visible only to a specific group of users, it is possible to define an embargo policy also for the ITEM itself. A READ policy for a specific group will mean that only the users in that group will be able to access the item splash page. Note that currently only some UIs (JSPUI/XMLUI) and in a very specific configuration (discovery enabled as search provider, and the SOLRBrowseDAOs is used for the Browse system) are fully rights aware. This means that in different UIs or with different configurations (legacy lucene search or DBMS browse) some metadata of a restricted item could be exposed to unauthorized users. When you need to work with UIs not fully rights aware, a workaround can be to use the "Private Item" flag to make the item undiscoverable so that metadata will be not exposed to unauthorized users. Please note that this workaround has several major limitations:

- No one, not even authorized users, is able to find the item by browsing or searching the repository.

- You need to manage externally a schedule that alerts you when the embargo is expired so that you may re-enable the discoverable nature of the item.