

https://dspace.org/

manual

Diol, Joe

2005-08-17
DURASPACE
https://hdl.handle.net/10539/30217

Downloaded from a DSpace repository. https://dspace.org/



DSpace 1.8 Documentation

Author: The DSpace Developer Team
Date: 24 February 2012
URL: https://wiki.duraspace.org/display/DSDOC18

Table of Contents

1 Preface 15
1.1 Release Notes 15
2 Introduction 17
3 Functional Overview 19
3.1 Data Model 19
3.2 Plugin Manager 21
3.3 Metadata 21
3.4 Packager Plugins 22
3.5 Crosswalk Plugins 23
3.6 E-People and Groups 23
3.6.1 E-Person 23
3.6.2 Groups 24
3.7 Authentication 24
3.8 Authorization 24
3.9 Ingest Process and Workflow 26
3.9.1 Workflow Steps 27
3.10 Supervision and Collaboration 28
3.11 Handles 28
3.12 Bitstream 'Persistent' Identifiers 29
3.13 Storage Resource Broker (SRB) Support 30
3.14 Search and Browse 30
3.15 HTML Support 31
3.16 OAI Support 32
3.17 SWORD Support 32
3.18 OpenURL Support 32
3.19 Creative Commons Support 33
3.20 Subscriptions 33
3.21 Import and Export 33
3.22 Registration 33
3.23 Statistics 34
3.23.1 System Statistics 34
3.23.2 Item, Collection and Community Usage Statistics 34
3.24 Checksum Checker 35
3.25 Usage Instrumentation 35
3.26 Choice Management and Authority Control 35
3.26.1 Introduction and Motivation 36

4 Installation 38
4.1 For the Impatient 38
4.2 Prerequisite Software 59
4.2.1 UNIX-like OS or Microsoft Windows 38
4.2.2 Oracle Java JDK 6 (standard SDK is fine, you don't need J2EE) 39
4.2.3 Apache Maven 2.2.x or higher (Java build tool) 39
4.2.4 Apache Ant 1.8 or later (Java build tool) 40
4.2.5 Relational Database: (PostgreSQL or Oracle). 40
4.2.6 Servlet Engine: (Apache Tomcat 5.5 or 6, Jetty, Caucho Resin or equivalent). 41
4.2.7 Perl (only required for [dspace]/bin/dspace-info.pl) 42
4.3 Installation Instructions 42
4.3.1 Overview of Install Options 42
4.3.2 Overview of DSpace Directories 44
4.3.3 Installation 45
4.4 Advanced Installation 49
4.4.1 'cron' Jobs 49
4.4.2 Multilingual Installation 50
4.4.3 DSpace over HTTPS 50
4.4.4 The Handle Server 55
4.4.5 Google and HTML sitemaps 56
4.4.6 DSpace Statistics 57
4.4.7 Manually Installing/Updating GeoLite Database File 58
4.5 Windows Installation 59
4.5.1 Installation Steps 59
4.6 Checking Your Installation 61
4.7 Known Bugs 61
4.8 Common Problems 62
4.8.1 Common Installation Issues 62
4.8.2 General DSpace Issues 64

5 Upgrading a DSpace Installation 66
5.1 Upgrading From 1.8 to 1.8.x 66
5.1.1 Backup your DSpace 67
5.1.2 Upgrade Steps 68
5.2 Upgrading From 1.7.x to 1.8.x 69
5.2.1 Backup your DSpace 71
5.2.2 Upgrade Steps 72
5.3 Upgrading From 1.7 to 1.7.x 76
5.3.1 Upgrade Steps 77
5.4 Upgrading From 1.6.x to 1.7.x 78
5.4.1 Upgrade Steps 78
5.5 Upgrading From 1.6 to 1.6.x 88
5.5.1 Upgrade Steps 89
5.6 Upgrading From 1.5.x to 1.6.x 90
5.6.1 Upgrade Steps 91
5.7 Upgrading From 1.5 or 1.5.1 to 1.5.2 104
5.7.1 Upgrade Steps 104
5.8 Upgrading From 1.4.2 to 1.5 113
5.8.1 Upgrade Steps 113
5.9 Upgrading From 1.4.1 to 1.4.2 118
5.9.1 Upgrade Steps 118
5.10 Upgrading From 1.4 to 1.4.x 118
5.10.1 Upgrade Steps 118
5.11 Upgrading From 1.3.2 to 1.4.x 120
5.11.1 Upgrade Steps 120
5.12 Upgrading From 1.3.1 to 1.3.2 123
5.12.1 Upgrade Steps 123
5.13 Upgrading From 1.2.x to 1.3.x 124
5.13.1 Upgrade Steps 124
5.14 Upgrading From 1.2.1 to 1.2.2 125
5.14.1 Upgrade Steps 126
5.15 Upgrading From 1.2 to 1.2.1 127
5.15.1 Upgrade Steps 127
5.16 Upgrading From 1.1.x to 1.2 129
5.16.1 Upgrade Steps 129
5.17 Upgrading From 1.1 to 1.1.1 132
5.17.1 Upgrade Steps 133
5.18 Upgrading From 1.0.1 to 1.1 133
5.18.1 Upgrade Steps 133
6 Configuration 137
6.1 General Configuration 137
6.1.1 Input Conventions 137
6.1.2 Update Reminder 138

6.2 The dspace.cfg Configuration Properties File 139
6.2.1 The dspace.cfg file 139
6.2.2 Main DSpace Configurations 150
6.2.3 DSpace Database Configuration 151
6.2.4 DSpace Email Settings 153
6.2.5 File Storage 156
6.2.6 SRB (Storage Resource Brokerage) File Storage 157
6.2.7 Logging Configuration 160
6.2.8 Configuring Lucene Search Indexes 161
6.2.9 Handle Server Configuration 164
6.2.10 Delegation Administration : Authorization System Configuration 165
6.2.11 Restricted Item Visibility Settings 169
6.2.12 Proxy Settings 170
6.2.13 Configuring Media Filters 171
6.2.14 Crosswalk and Packager Plugin Settings 173
6.2.15 Event System Configuration 177
6.2.16 Embargo 180
6.2.17 Checksum Checker Settings 185
6.2.18 Item Export and Download Settings 186
6.2.19 Subscription Emails 187
6.2.20 Hiding Metadata 187
6.2.21 Settings for the Submission Process 188
6.2.22 Configuring Creative Commons License 188
6.2.23 WEB User Interface Configurations 190
6.2.24 Browse Index Configuration 194
6.2.25 Author (Multiple metadata value) Display 198
6.2.26 Links to Other Browse Contexts 199
6.2.27 Recent Submissions 200
6.2.28 Submission License Substitution Variables 201
6.2.29 Syndication Feed (RSS) Settings 201
6.2.30 OpenSearch Support 205
6.2.31 Content Inline Disposition Threshold 207
6.2.32 Multi-file HTML Document/Site Settings 208
6.2.33 Sitemap Settings 208
6.2.34 Authority Control Settings 209
6.2.35 JSPUI Upload File Settings 210
6.2.36 JSP Web Interface (JSPUI) Settings 211
6.2.37 JSPUI Configuring Multilingual Support 215
6.2.38 JSPUI Item Mapper 217
6.2.39 Display of Group Membership 217
6.2.40 JSPUI / XMLUI SFX Server 217
6.2.41 JSPUI Item Recommendation Setting 219
6.2.42 Controlled Vocabulary Settings 219

6.2.43 XMLUI Specific Configuration 221
6.2.44 DSpace SOLR Statistics Configuration 225
6.3 Optional or Advanced Configuration Settings 226
6.3.1 The Metadata Format and Bitstream Format Registries 227
6.3.2 XPDF Filter 228
6.3.3 Creating a new Media/Format Filter 231
6.3.4 Configuring Usage Instrumentation Plugins 233
6.4 Authentication Plugins 234
6.4.1 Stackable Authentication Method(s) 234
6.5 Batch Metadata Editing Configuration 256
6.6 Configurable Workflow 257
6.6.1 Introduction 257
6.6.2 Instructions for Enabling Configurable Reviewer Workflow in XMLUI 258
6.6.3 Data Migration (Backwards compatibility) 259
6.6.4 Configuration 260
6.6.5 Authorizations 266
6.6.6 Database 266
6.6.7 Additional workflow steps/actions and features 268
6.6.8 Known Issues 269
6.7 Discovery 270
6.7.1 What is DSpace Discovery 270
6.7.2 Discovery Features 271
6.7.3 DSpace 1.8 Improvements 271
6.7.4 Enabling Discovery 272
6.7.5 Configuration files 273
6.7.6 General Discovery settings (config/modules/discovery.cfg) 274
6.7.7 Modifying the Discovery User Interface (config/spring/spring-dspace-addon-discovery-configuration-services.xml) 274
6.7.8 Routine Discovery SOLR Index Maintenance 281
6.7.9 Advanced SOLR Configuration 281
6.8 DSpace Service Manager 282
6.8.1 Introduction 282
6.8.2 Configuration 282
6.8.3 Architectural Overview 285
6.8.4 Tutorials 285

6.9 DSpace Statistics 285
6.9.1 What is exactly being logged ? 285
6.9.2 Web user interface for DSpace statistics 286
6.9.3 Usage Event Logging and Usage Statistics Gathering 287
6.9.4 Configuration settings for Statistics 287
6.9.5 Older setting that are not related to the new 1.6 Statistics 290
6.9.6 Statistics Administration 291
6.9.7 Statistics differences between DSpace 1.7.x and 1.8.0 291
6.9.8 Statistics differences between DSpace 1.6.x and 1.7.0 292
6.9.9 Web UI Statistics Modification (XMLUI Only) 292
6.9.10 Custom Reporting - Querying SOLR Directly 293
6.10 Embargo 294
6.10.1 What is an embargo? 294
6.11 Google Scholar Metadata Mappings 298
6.12 OAI 299
6.12.1 OAI Interfaces 299
6.13 SWORDv1 Client 306
6.13.1 Enabling the SWORD Client 306
6.13.2 Configuring the SWORD Client 307
6.14 SWORDv1 Server 308
6.14.1 Enabling SWORD Server 308
6.14.2 Configuring SWORD Server 308
6.15 SWORDv2 Server 313
6.15.1 Enabling SWORD v2 Server 314
6.15.2 Configuring SWORD v2 Server 314
7 JSPUI Configuration and Customization 322
7.1 Configuration 322
7.2 Customizing the JSP pages 322

8 XMLUI Configuration and Customization 324
8.1 Overview of XMLUI / Manakin 324
8.1.1 Understanding the Flow of an XMLUI Request 325
8.2 Manakin Configuration Property Keys 327
8.3 Configuring Themes and Aspects 330
8.3.1 Aspects 330
8.3.2 Themes 332
8.4 Multilingual Support 332
8.5 Creating a New Theme 333
8.6 Customizing the News Document 334
8.7 Adding Static Content 335
8.8 Harvesting Items from XMLUI via OAI-ORE or OAI-PMH 336
8.8.1 Automatic Harvesting (Scheduler) 338
8.9 Additional XMLUI Learning Resources 338
8.10 Mirage Configuration and Customization 338
8.10.1 Introduction 339
8.10.2 Configuration Parameters 339
8.10.3 Technical Features 340
8.10.4 Troubleshooting 341
8.11 XMLUI Base Theme Templates (dri2xhtml) 342
8.11.1 dri2xhtml 342
8.11.2 dri2xhtml-alt 343
9 Advanced Customisation 346
9.1 Maven WAR Overlays 346
9.2 DSpace Source Release 346
10 System Administration 347
10.1 AIP Backup and Restore 347
10.1.1 Background & Overview 347
10.1.2 Makeup and Definition of AIPs 351
10.1.3 Running the Code 352
10.1.4 Additional Packager Options 363
10.1.5 Configuration in 'dspace.cfg' 368
10.1.6 Common Issues or Error Messages 371
10.1.7 DSpace AIP Format 372
10.2 Batch Metadata Editing 390
10.2.1 Batch Metadata Editing Tool 390

10.3 Curation System 394
10.3.1 Changes in 1.8 395
10.3.2 Tasks 395
10.3.3 Activation 395
10.3.4 Writing your own tasks 396
10.3.5 Task Invocation 397
10.3.6 Asynchronous (Deferred) Operation 400
10.3.7 Task Output and Reporting 401
10.3.8 Task Properties 402
10.3.9 Task Annotations 404
10.3.10Scripted Tasks 404
10.3.11Starter Tasks 406
10.4 Importing and Exporting Content via Packages 411
10.4.1 Package Importer and Exporter 411
10.5 Importing and Exporting Items via Simple Archive Format 417
10.5.1 Item Importer and Exporter 417
10.6 Importing Community and Collection Hierarchy 423
10.6.1 Community and Collection Structure Importer 423
10.7 Managing Community Hierarchy 425
10.7.1 Sub-Community Management 425
10.8 Managing Embargoed Content 427
10.8.1 Embargo Lifter 427
10.9 Managing Usage Statistics 427
10.9.1 DSpace Log Converter 428
10.9.2 Filtering and Pruning Spiders 429
10.9.3 Routine SOLR Index Maintenance 430
10.10Moving Items 430
10.10.1Moving Items via Web UI 430
10.10.2Moving Items via the Batch Metadata Editor 431
10.11Performance Tuning DSpace 431
10.11.1Give Tomcat (DSpace UIs) More Memory 431
10.11.2Give the Command Line Tools More Memory 433
10.11.3Give PostgreSQL Database More Memory 434
10.11.4SOLR Statistics Performance Tuning 434
10.12Registering (not Importing) Bitstreams via Simple Archive Format 434
10.12.1Overview 435
10.13ReIndexing Content (for Browse or Search) 437
10.13.1Overview 437
10.13.2Creating the Browse & Search Indexes 437
10.13.3Running the Indexing Programs 438
10.13.4Indexing Customization 439
10.14Testing Database Connection 440
10.14.1Test Database 440

10.15Transferring or Copying Content Between Repositories 441
10.15.1Transferring Content via Export and Import 441
10.15.2Transferring Items using Simple Archive Format 441
10.15.3Transferring Items using OAI-ORE/OAI-PMH Harvester 442
10.15.4Copying Items using the SWORD Client 442
10.16Transforming DSpace Content (MediaFilters) 442
10.16.1MediaFilters: Transforming DSpace Content 442
10.17Updating Items via Simple Archive Format 445
10.17.1Item Update Tool 445
10.18Validating CheckSums of Bitstreams 448
10.18.1Checksum Checker 448
11 Directories and Files 453
11.1 Overview 453
11.2 Source Directory Layout 453
11.3 Installed Directory Layout 455
11.4 Contents of JSPUI Web Application 455
11.5 Contents of XMLUI Web Application (aka Manakin) 455
11.6 Log Files 456
11.6.1 log4j.properties File. 458

12 Architecture 459
12.1 Overview 459
12.1.1 DSpace System Architecture 459
12.2 Application Layer 461
12.2.1 Web User Interface 461
12.2.2 OAI-PMH Data Provider 471
12.2.3 DSpace Command Launcher 475
12.3 Business Logic Layer 476
12.3.1 Core Classes 476
12.3.2 Content Management API 479
12.3.3 Plugin Manager 484
12.3.4 Workflow System 494
12.3.5 Administration Toolkit 495
12.3.6 E-person/Group Manager 495
12.3.7 Authorization 496
12.3.8 Handle Manager/Handle Plugin 497
12.3.9 Search 498
12.3.10Browse API 500
12.3.11Checksum checker 503
12.3.12OpenSearch Support 503
12.3.13Embargo Support 505
12.4 DSpace Services Framework 507
12.4.1 Architectural Overview 507
12.4.2 Basic Usage 509
12.4.3 Providers and Plugins 510
12.4.4 Core Services 511
12.4.5 Examples 512
12.4.6 Tutorials 513
12.5 Storage Layer 513
12.5.1 RDBMS / Database Structure 513
12.5.2 Bitstream Store 516

13 Submission User Interface 522
13.1 Understanding the Submission Configuration File 522
13.1.1 The Structure of item-submission.xml 522
13.1.2 Defining Steps (<step>) within the item-submission.xml 523
13.2 Reordering/Removing Submission Steps 525
13.3 Assigning a custom Submission Process to a Collection 526
13.3.1 Getting A Collection's Handle 529
13.4 Custom Metadata-entry Pages for Submission 527
13.4.1 Introduction 527
13.4.2 Describing Custom Metadata Forms 528
13.4.3 The Structure of input-forms.xml 528
13.4.4 Deploying Your Custom Forms 533
13.5 Configuring the File Upload step 533
13.6 Creating new Submission Steps 533
13.6.1 Creating a Non-Interactive Step 534
14 DRI Schema Reference 536
14.1 Introduction 536
14.1.1 The Purpose of DRI 536
14.1.2 The Development of DRI 536
14.2 DRI in Manakin 537
14.2.1 Themes 537
14.2.2 Aspect Chains 538
14.3 Common Design Patterns 538
14.3.1 Localization and Internationalization 538
14.3.2 Standard attribute triplet 539
14.3.3 Structure-oriented markup 539
14.4 Schema Overview 540
14.5 Merging of DRI Documents 541
14.6 Version Changes 542
14.6.1 Changes from 1.0 to 1.1 542

14.7 Element Reference 543
14.7.1 BODY 547
14.7.2 cell 548
14.7.3 div 549
14.7.4 DOCUMENT 551
14.7.5 field 551
14.7.6 figure 553
14.7.7 head 554
14.7.8 help 555
14.7.9 hi 556
14.7.10instance 557
14.7.11item 557
14.7.12label 558
14.7.13list 560
14.7.14META 561
14.7.15metadata 562
14.7.16OPTIONS 563
14.7.17p 564
14.7.18pageMeta 565
14.7.19params 567
14.7.20reference 568
14.7.21referenceSet 569
14.7.22repository 570
14.7.23repositoryMeta 570
14.7.24row 571
14.7.25table 572
14.7.26trail 574
14.7.27userMeta 574
14.7.28value 576
14.7.29xref 577
15 Appendices 579
15.1 Appendix A 579
15.1.1 Default Dublin Core Metadata Registry 579
15.1.2 Default Bitstream Format Registry 581

16 History 584
16.1 Changes in DSpace 1.8.2 584
16.1.1 General Improvements 637
16.1.2 Bug Fixes 623
16.2 Changes in DSpace 1.8.1 585
16.3 Changes in DSpace 1.8.0 586
16.3.1 New Features 621
16.4 Changes in DSpace 1.7.2 594
16.5 Changes in DSpace 1.7.1 595
16.6 Changes in DSpace 1.7.0 597
16.7 Changes in DSpace 1.6.2 607
16.8 Changes in DSpace 1.6.1 608
16.9 Changes in DSpace 1.6.0 611
16.10Changes in DSpace 1.5.2 621
16.11Changes in DSpace 1.5.1 628
16.11.1General Improvements and Bug Fixes 628
16.12Changes in DSpace 1.5 630
16.12.1Bug fixes and smaller patches 631
16.13Changes in DSpace 1.4.1 632
16.13.1Bug fixes 642
16.14Changes in DSpace 1.4 633
16.15Changes in DSpace 1.3.2 635
16.16Changes in DSpace 1.3.1 635
16.17Changes in DSpace 1.3 635
16.18Changes in DSpace 1.2.2 636
16.18.1Changes in JSPs 637
16.19Changes in DSpace 1.2.1 637
16.19.1Changed JSPs 638
16.20Changes in DSpace 1.2 639
16.20.1General Improvments 639
16.20.2Administration 639
16.20.3Import/Export/OAI 639
16.20.4Miscellaneous 639
16.20.5JSP file changes between 1.1 and 1.2 640
16.21Changes in DSpace 1.1.1 642
16.21.1Improvements 642
16.22Changes in DSpace 1.1 643

1 Preface

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 1.8.0 Documentation is also available at: https://wiki.duraspace.org/display/DSDOC18

1.1 Release Notes

Welcome to Release 1.8.0. The developers have volunteered many hours to fix, re-write and contribute new software code for this release. Documentation has also been updated.

The following is a list of the new features included for release 1.8.0 (not an exhaustive list):

- Improvements to the upgrade and configuration process (see page 69) (sections of dspace.cfg have been split into separate configuration files).
- More Curation Tools/Plugins (see page 394).
- Configurable Workflow (see page 257).
- SWORD Client (see page 306).
- SWORDv2 Server Module (see page 313).
- RSS feeds now support richer features, such as iTunes podcast and publishing to iTunesU (see new "webui.feed.podcast.*" settings (see page 201)).
- Enhancements to Discovery (see page 270).
- Rewrite of Creative Commons licensing (see page 188) for the XMLUI.
- Reordering of bitstreams.
- Enable virus checking during submission (see page 407).
- Ability to Withdraw/Reinstate/Delete Items in Bulk, via Batch Metadata Editing (see page 394).

A full list of all changes / bug fixes in 1.8.0 is available in the History (see page 584) section.

The following people have contributed directly to this release of DSpace: Alex Lemann, Álvaro López, Andrea Schweer, Ben Bosman, Bill Hays, Bram De Schouwer, Bram Luyten, Brian Freels-Stendel, Claudia Jürgen, David Chandek-Stark, Denys Slipetskyy, Fabio Bolognesi, Gareth Waller, Hardik Mishra, Hardy Pottinger, Ivan Masár, James Russell, Janne Pietarila, Jason Stirnaman, Joonas Kesäniemi, Jordan Pišanc, Jose Blanco, Juan García, Kevin Van de Velde, Kim Shepherd, Konstantinos V. Paraskevopoulos 1, Lighton Phiri, Mark Diggory, Mark H. Wood, Nicholas Riley, Onivaldo Rosa Junior, Peter Dietz, Richard Rodgers, Robin Taylor, Ronee Francis, Samuel Ottenhoff, Scott Phillips, Stuart Lewis, Stuart Yeates, Terry Burton, Tim Donohue, Timo Aalto, Vladislav Zhivkov, Wendy Bossons. Many of them 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: Amy Lana, Augustine Gitonga, Bram Luyten, Ciarán Walsh, Claire Bundy, Dibyendra Hyoju, Elena Feinstein, Elin Stangeland, Imma Subirats, Iryna Kuchma, Jennifer Laherty, Jim Ottaviani, Leonie Hayes, Maureen Walsh, Michael Guthrie, Sarah Shreeves, Sue Kunda, and Valorie Hollister.

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. Acknowledgements 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 Documentation Gardener for this release was Jeffrey Trimble with input from everyone. All typos are his fault.

Robin Taylor was the Release Coordinator of this release with immeasurable help from the DSpace Technical Lead Tim Donohue.

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

2 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 (see page 19), 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 (see page 38), and sections on configuration (see page 137) and the directory structure (see page 453).

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 (see page 459).

Other good sources of information are:

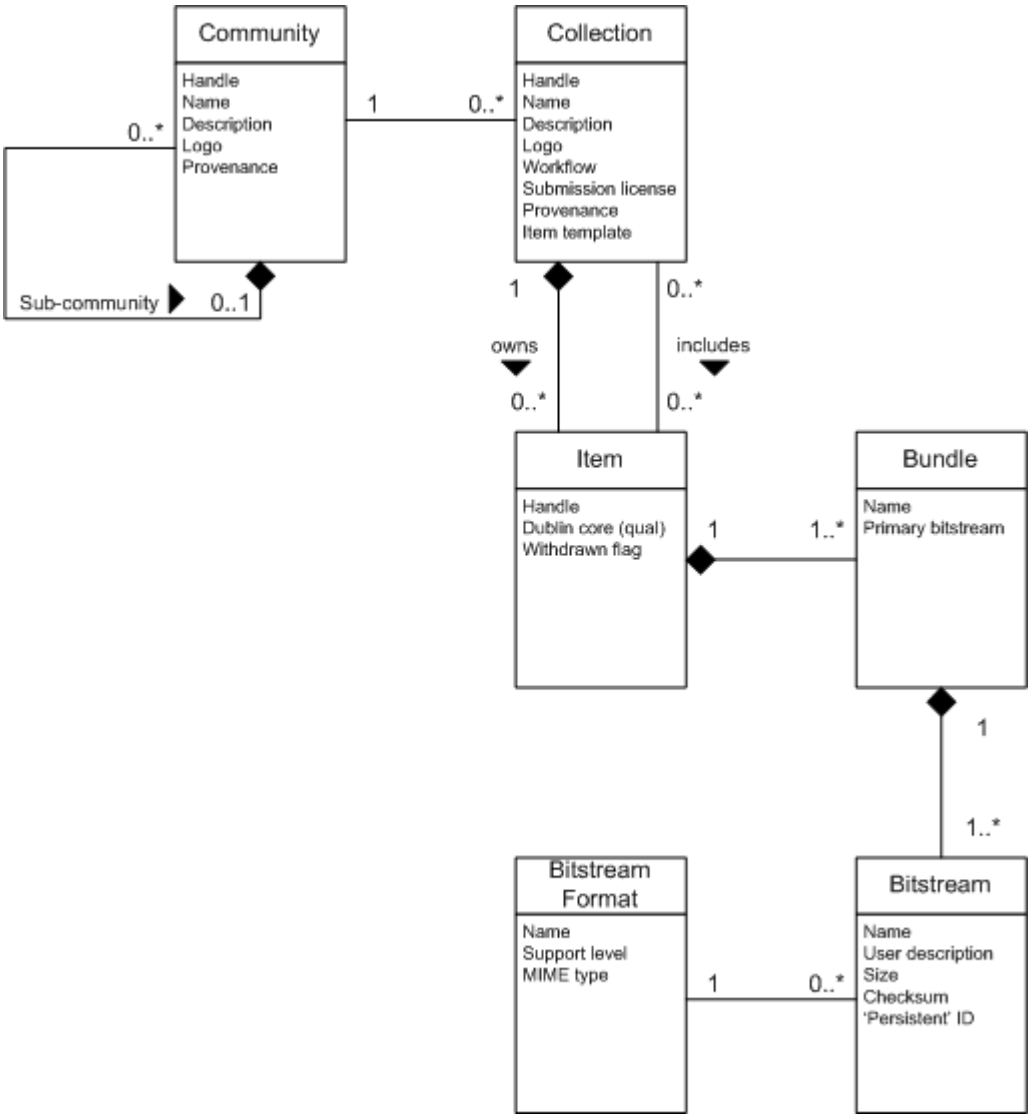
- The DSpace Public API Javadocs. Build these with the command `mvn javadoc:javadoc`
- The DSpace Wiki contains stacks of useful information about the DSpace platform and the work people are doing with it. You are strongly encouraged to visit this site and add information about your own work. Useful 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.

3 Functional Overview

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

3.1 Data Model



Data Model Diagram

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

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

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

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

3.2 Plugin Manager

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

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

The mediafilter is a simple example of a plugin implementation. Refer to the Business Logic Layer (see page 476) for more details on Plugins.

3.3 Metadata

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

- **Descriptive Metadata**: DSpace can support multiple flat metadata schemas for describing an item. A qualified Dublin Core metadata schema loosely based on the 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.

3.4 Packager Plugins

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

A package is typically an archive file such as a Zip or "tar" file, including a *manifest* document which contains metadata and a description of the package contents. The IMS Content Package 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 (see page 21)), 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 (see page 23) 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 (see page) section of the System Administration documentation.

3.5 Crosswalk Plugins

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

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

Crosswalk plugins are named plugins (see Plugin Manager (see page 21)), 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.

3.6 E-People and Groups

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

3.6.1 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.

3.6.2 Groups

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

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

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

3.7 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.

3.8 Authorization

DSpace's authorization system is based on associating actions with objects and the lists of EPeople who can perform them. The associations are called Resource Policies, and the lists of EPeople are called Groups. There are two 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 *not* 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

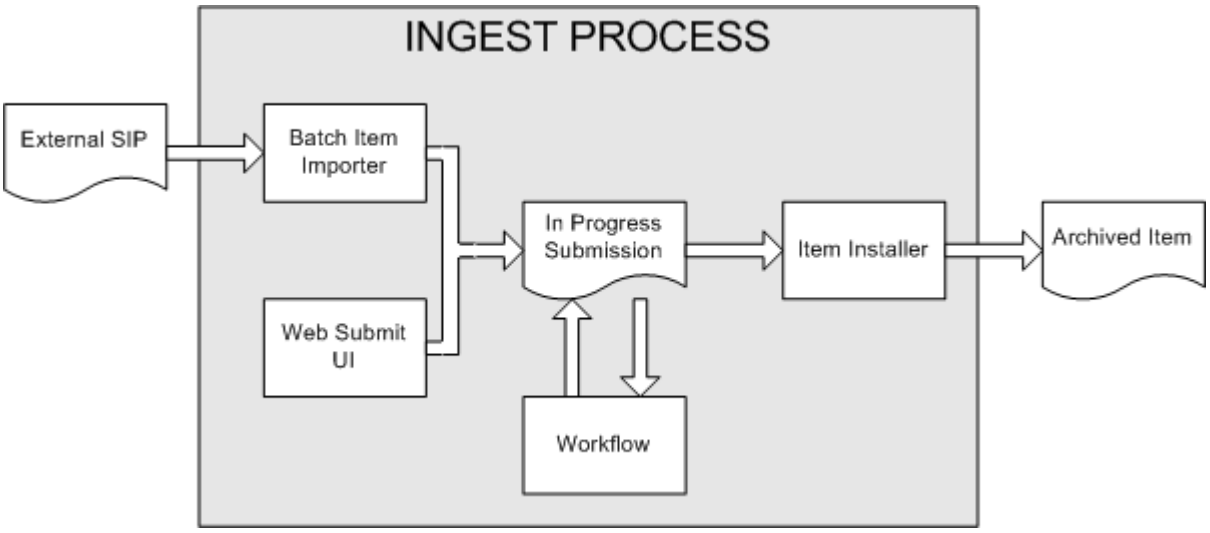
READ	view bitstream
WRITE	modify bitstream

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.

3.9 Ingest Process and Workflow

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



DSpace Ingest Process

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

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

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

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

3.9.1 Workflow Steps

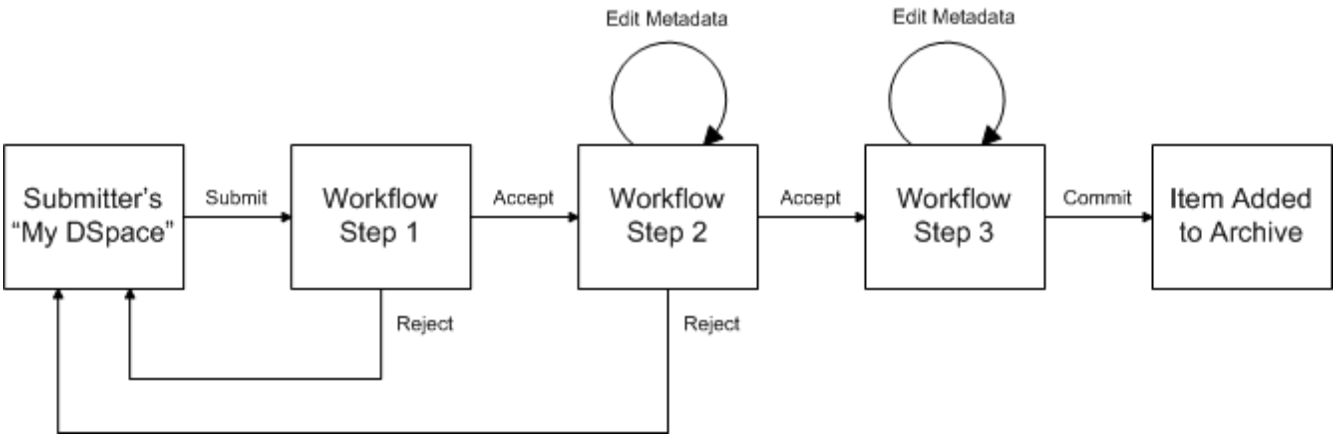
A collection's workflow can have up to three steps. Each collection may have an associated e-person group for performing each step; if no group is associated with a certain step, that step is skipped. If a collection has no e-person groups associated with any step, submissions to that collection are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

3.10 Supervision and Collaboration

In order to facilitate, as a primary objective, the opportunity for thesis authors to be supervised in the preparation of their e-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.

3.11 Handles

Researchers require a stable point of reference for their works. The simple evolution from sharing of citations to emailing of URLs broke when Web users learned that sites can disappear or be reconfigured without notice, and that their bookmark files containing critical links to research results couldn't be trusted 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.

3.12 Bitstream 'Persistent' Identifiers

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

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

dspace url/bitstream/handle/sequence ID/filename

For example:

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

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

3.13 Storage Resource Broker (SRB) Support

DSpace offers two means for storing bitstreams. The first is in the file system on the server. The second is using 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.

3.14 Search and Browse

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

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

- Browsing though title, author, date or subject indices, with optional image thumbnails
  Search is an essential component of discovery in DSpace. Users' expectations from a search engine are quite high, so a goal for DSpace is to supply as many search features as possible. DSpace's indexing and search module has a very simple API which allows for indexing new content, regenerating the index, and performing searches on the entire corpus, a community, or collection. Behind the API is the Java freeware search engine Lucene. Lucene gives us fielded searching, stop word removal, stemming, and the ability to incrementally add new indexed content without regenerating the entire index. The specific Lucene search indexes are configurable enabling institutions to customize which DSpace metadata fields are indexed.

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

3.15 HTML Support

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

- Web pages tend to consist of several files – one or more HTML files that contain references to each other, and stylesheets and image files that are referenced by the HTML files.
- Web pages also link to or include content from other sites, often imperceptibly to the end-user. Thus, in a few year's time, when someone views the preserved Web site, they will probably find that many links are now broken or refer to other sites than are now out of context.In fact, it may be unclear to an end-user when they are viewing content stored in DSpace and when they are seeing content included from another site, or have navigated to a page that is not stored in DSpace. This problem can manifest when a submitter uploads some HTML content. For example, the HTML document may include an image from an external Web site, or even their local hard drive. When the submitter views the HTML in DSpace, their browser is able to use the reference in the HTML to retrieve the appropriate image, and so to the submitter, the whole HTML document appears to have been deposited correctly. However, later on, when another user tries to view that HTML, their browser might not be able to retrieve the included image since it may have been removed from the external server. Hence the HTML will seem broken.
- Often Web pages are produced dynamically by software running on the Web server, and represent the state of a changing database underneath it.
  Dealing with these issues is the topic of much active research. Currently, DSpace bites off a small, tractable chunk of this problem. DSpace can store and provide on-line browsing capability for *self-contained, non-dynamic* HTML documents. In practical terms, this means:

- No dynamic content (CGI scripts and so forth)
- All links to preserved content must be *relative links*, that do not refer to 'parents' above the 'root' of the HTML document/site:
  - *diagram.gif* is OK
  - *image/foo.gif* is OK
  - *../index.html* is only OK in a file that is at least a directory deep in the HTML document/site hierarchy
  - */stylesheet.css* is not OK (the link will break)
  - *http://somedomain.com/content.html* is not OK (the link will continue to link to the external site which may change or disappear)
- Any 'absolute links' (e.g. *http://somedomain.com/content.html*) are stored 'as is', and will continue to link to the external content (as opposed to relative links, which will link to the copy of the content stored in DSpace.) Thus, over time, the content referred to by the absolute link may change or disappear.

3.16 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.

3.17 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.

3.18 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.

3.19 Creative Commons Support

DSpace provides support for Creative Commons licenses to be attached to items in the repository. They represent an alternative to traditional copyright. To learn more about Creative Commons, visit their website. 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.

3.20 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.

3.21 Import and Export

DSpace includes batch tools to import and export items in a simple directory structure, where the Dublin Core metadata is stored in an XML file. This may be used as the basis for moving content between DSpace and other systems. For more information see Item Importer and Exporter (see page).

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

3.22 Registration

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

3.23 Statistics

DSpace offers system statistics for administrator usage, as well as usage statistics on the level of items, communities and collections.

3.23.1 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

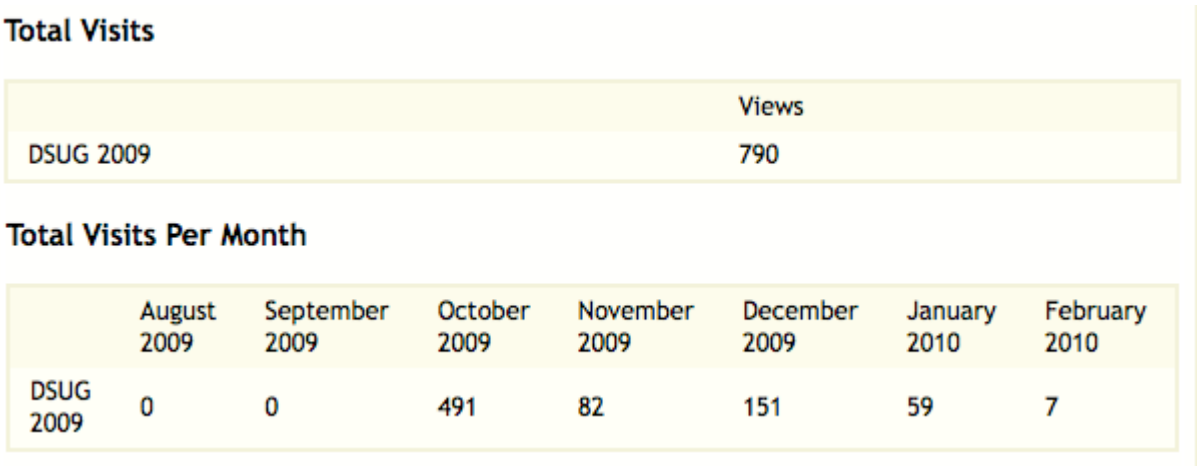
- A customizable general overview of activities in the archive, by default including:
  - Number of items archived
  - Number of bitstream views
  - Number of item page views
  - Number of collection page views
  - Number of community page views
  - Number of user logins
  - Number of searches performed
  - 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.

3.23.2 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

3.24 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.

3.25 Usage Instrumentation

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

3.26 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.

3.26.1 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.

4 Installation

4.1 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 (see page 42)

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

4.2 Prerequisite Software

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

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

4.2.1 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 distributions documentation or local system administrators to determine what is already available.
- Microsoft Windows: After verifying all prerequisites below, see the Windows Installation (see page 59) section for Windows tailored instructions

4.2.2 Oracle Java JDK 6 (standard SDK is fine, you don't need J2EE)

DSpace requires Oracle Java 6 (standard SDK is fine, you don't need J2EE). Please note, at this time, DSpace does not function properly with Java JDK 7 (see warning below).

Oracle's Java can be downloaded from the following location: http://www.oracle.com/technetwork/java/javase/downloads/index.html. Again, you can just download the Java SE JDK version.

Java 7 is currently unsupported

DSpace does not currently support Java 7, as there is a known issue with Java 7 and Lucene/SOLR (which DSpace uses for search & browse functionality). For more details, see this article on the Apache site: "WARNING: Index corruption and crashes in Apache Lucene Core / Apache Solr with Java 7" as well as this Java bug report: 7073868

Other flavors of Java may cause issues

Only Oracle's Java has been tested with each release and is known to work correctly. Other flavors of Java may pose problems.

4.2.3 Apache Maven 2.2.x or higher (Java build tool)

DSpace 1.7.x requires usage of Maven 2.2.x

DSpace 1.7.x required usage of Maven 2.2.x, as it did not build properly when using Maven 2.0.x or Maven 3.x. This was a known issue (see DS-788). However, DSpace 1.8.x resolved this issue so that DSpace now builds properly with Maven 2.2.x or above.

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 2.0. 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>
```

4.2.4 Apache Ant 1.8 or later (Java build tool)

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

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

4.2.5 Relational Database: (PostgreSQL or Oracle).

- **PostgreSQL 8.2 to 8.4** PostgreSQL can be downloaded from the following location: http://www.postgresql.org/ . It is highly recommended that you try to work with Postgres 8.4 or greater, however, 8.2 or greater should still work. Unicode (specifically UTF-8) support must be enabled. This is enabled by default in 8.0+. Once installed, you need to enable TCP/IP connections (DSpace uses JDBC). In *postgresql.conf*: uncomment the line starting: *listen_addresses = 'localhost'*. Then tighten up security a bit by editing *pg_hba.conf* and adding this line: *host dspace dspace 127.0.0.1 255.255.255.255 md5*. Then restart PostgreSQL.
- **Oracle 10g or greater** Details on acquiring Oracle can be downloaded from the following location: http://www.oracle.com/database/. You will need to create a database for DSpace. Make sure that the character set is one of the Unicode character sets. DSpace uses UTF-8 natively, and it is suggested that the Oracle database use the same character set. You will also need to create a user account for DSpace (e.g. *dspace*) and ensure that it has permissions to add and remove tables in the database. Refer to the Quick Installation for more details.
    - **NOTE:** DSpace uses sequences to generate unique object IDs — beware Oracle sequences, which are said to lose their values when doing a database export/import, say restoring from a backup. Be sure to run the script *etc/update-sequences.sql*.
    - For people interested in switching from Postgres to Oracle, I know of no tools that would do this automatically. You will need to recreate the community, collection, and eperson structure in the Oracle system, and then use the item export and import tools to move your content over.

4.2.6 Servlet Engine: (Apache Tomcat 5.5 or 6, Jetty, Caucho Resin or equivalent).

- **Apache Tomcat 5.5 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*.
- **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.

4.2.7 Perl (only required for [dspace]/bin/dspace-info.pl)

4.3 Installation Instructions

4.3.1 Overview of Install Options

With the advent of a new Apache Maven 2 based build architecture (first introduced inDSpace 1.5.x), you now have two options in how you may wish to install and manage your local installation of DSpace. If you've used DSpace 1.4.x, please recognize that the initial build procedure has changed to allow for more customization. You will find the later 'Ant based' stages of the installation procedure familiar. Maven is used to resolve the dependencies of DSpace online from the 'Maven Central Repository' server.

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

- 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-discovery* - Discovery source module
      - *dspace-jspui/* - JSP-UI source module
      - *dspace-oai* - OAI-PMH source module
      - *dspace-xmlui* - XML-UI (Manakin) source module
      - *dspace-lni* - Lightweight Network Interface source module
      - *dspace-stats* - Statistics source module
      - *dspace-sword* - SWORD (Simple Web-serve Offering Repository Deposit) deposit service source module
      - *dspace-swordv2* - SWORDv2 source module
      - *dspace-sword-client* - XMLUI client for SWORD
      - *pom.xml* - DSpace Parent Project definition

4.3.2 Overview of DSpace Directories

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

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

1. **The installation directory**, referred to as `[dspace]`. This is the location where DSpace is installed and running off of it is the location that gets defined in the `dspace.cfg` as "dspace.dir". It is where all the DSpace configuration files, command line scripts, documentation and webapps will be installed to.
2. **The source directory**, referred to as `[dspace-source]` . This is the location where the DSpace release distribution has been unzipped into. It usually has the name of the archive that you expanded such as `dspace-<version>-release` or `dspace-<version>-src-release`. 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). In DSpace 1.5.x and above, this corresponds to `[dspace]/webapps` by default. However, if you are using Tomcat, you may decide to copy your DSpace web applications from `[dspace]/webapps/` to `[tomcat]/webapps/` (with `[tomcat]` being wherever you installed Tomcat, also known as `$CATALINA_HOME`).
   For details on the contents of these separate directory trees, refer to directories.html. *Note that the* `[dspace-source]` *and* `[dspace]` *directories are always separate!*

4.3.3 Installation

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

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

```
useradd -m dspace
```

2. **Download** the latest DSpace release There are two version available with each release of DSpace: (*dspace-1.x-release.* and *dspace-1.x-src-release.xxx*); you only need to choose one. If you want a copy of all underlying Java source code, you should download the *dspace-1.x-src-release.xxx* Within each version, you have a choice of compressed file format. Choose the one that best fits your environment.
3. **Unpack the DSpace software**. After downloading the software, based on the compression file format, choose one of the following methods to unpack your software:
   1. **Zip file**. If you downloaded *dspace-1.8-release.zip* do the following:

```
unzip dspace-1.8-release.zip
```

   2. **.gz file**. If you downloaded *dspace-1.8-release.tar.gz* do the following:

```
gunzip -c dspace-1.8-release.tar.gz | tar -xf -
```

   3. **.bz2 file**. If you downloaded _dspace-1.8-release.tar.bz2_do the following:

```
bunzip2 dspace-1.8-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 which to change the ownership of the *dspace-1.6-release* to the 'dspace' user. (And you may need to change the group).

4. **Database Setup**
    - *PostgreSQL:*
        - A 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.

          ```
          createuser -U postgres -d -A -P dspace
          ```

          You will be prompted for the password of the PostgreSQL superuser (`postgres`). Then you'll be prompted (twice) for a password for the new `dspace` user.
        - Create a `dspace` database, owned by the `dspace` PostgreSQL user *(you are still logged in at 'root')*:

          ```
          createdb -U dspace -E UNICODE dspace
          ```

          You will be prompted for the password of the DSpace database user. (This isn't the same as the *dspace* user's UNIX password.)
    - *Oracle:*
        - Setting up to use Oracle is a bit different now. You will need still need to get a copy of the Oracle JDBC driver, but instead of copying it into a lib directory you will need to install it into your local Maven repository. (You'll need to download it first from this location: http://www.oracle.com/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=11.2.0.2.0
              -Dpackaging=jar
              -DgeneratePom=true
          ```

        - 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.
        - Edit the *[dspace-source]/dspace/config/dspace.cfg* database settings:

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

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

   > You can interpolate the value of one configuration variable in the value of another one. For example, to set *feedback.recipient* to the same value as *mail.admin*, the line would look like:
   > `feedback.recipient = ${mail.admin}`
   > Refer to the General Configuration (see page 137) section for details and examples of the above.

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

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

   (Assuming the *dspace* UNIX username.)

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

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

   > **Defaults to PostgreSQL settings**
   >
   > Without any extra arguments, the DSpace installation package is initialized for PostgreSQL. If you want to use Oracle instead, you should build the DSpace installation package as follows:
   > `mvn -Ddb.name=oracle package`

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

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

> To see a complete list of build targets, run: `ant help` *The most likely thing to go wrong here is the database connection. See the* *Common Problems (see page 62)* *Section*.

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

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

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

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

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

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

4.4 Advanced Installation

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

4.4.1 'cron' Jobs

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

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

```
crontab -e
```

Then add the following lines:

```
# Send out subscription e-mails at 01:00 every day
0 1 * * *  [dspace]/bin/dspace sub-daily
# Run the media filter at 02:00 every day
0 2 * * *  [dspace]/bin/dspace filter-media
# Run the checksum checker at 03:00
0 3 * * *  [dspace]/bin/dspace checker -lp
# Mail the results to the sysadmin at 04:00
0 4 * * *  [dspace]/bin/dspace checker-emailer -c
```

Naturally you should change the frequencies to suit your environment.

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

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

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

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

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

4.4.2 Multilingual Installation

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

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

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

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

4.4.3 DSpace over HTTPS

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

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

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

To enable the HTTPS support in Tomcat 5.0:

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

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

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

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

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

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

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

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

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

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

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

1. Optional – ONLY if you don't already have a server certificate. Follow this sub-procedure to request a new, signed server certificate from your Certifying Authority (CA):
    - Create a new key pair under the alias name *"tomcat"*. When generating your key, give the Distinguished Name fields the appropriate values for your server and institution. CN should be the fully-qualified domain name of your server host. Here is an example:

      ```
      $JAVA_HOME/bin/keytool -genkey -alias tomcat -keyalg RSA -keysize
       1024 \
        -keystore $CATALINA_BASE/conf/keystore -storepass changeit
       -validity 365 \
        -dname 'CN=dspace.myuni.edu, OU=MIT Libraries, O=Massachusetts
       Institute of Technology, L=Cambridge, S=MA, C=US'
      ```

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

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

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

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

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

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

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

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

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

   When answering the questions to identify the certificate, be sure to respond to "First and last name" with the fully-qualified domain name of your server (e.g. *test-dspace.myuni.edu*). The other questions are not important.
3. Optional – ONLY if you need to accept client certificates for the X.509 certificate stackable authentication module See the configuration section for instructions on enabling the X.509 authentication method. Load the keystore with the CA (certifying authority) certificates for the authorities of any clients whose certificates you wish to accept. For example, assuming the client CA certificate is in *client1.pem*:

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

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

To use SSL on Apache HTTPD with mod_jk:

If you choose Apache HTTPD 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.

4.4.4 The Handle Server

First a few facts to clear up some common misconceptions:

- You don't **have** to use CNRI's Handle system. At the moment, you need to change the code a little to use something else (e.g PURLs) but that should change soon.
- You'll notice that while you've been playing around with a test server, DSpace has apparently been creating handles for you looking like *hdl:123456789/24* and so forth. These aren't really Handles, since the global Handle system doesn't actually know about them, and lots of other DSpace test installs will have created the same IDs. They're only really Handles once you've registered a prefix with CNRI (see below) and have correctly set up the Handle server included in the DSpace distribution. This Handle server communicates with the rest of the global Handle infrastructure so that anyone that understands Handles can find the Handles your DSpace has created.
  If you want to use the Handle system, you'll need to set up a Handle server. This is included with DSpace. Note that this is not required in order to evaluate DSpace; you only need one if you are running a production service. You'll need to obtain a Handle prefix from 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 broadcast and receive TCP on port 2641.

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

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

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

2. Edit the resulting *[dspace]/handle-server/config.dct* file to include the following lines in the *"server_config"* clause:

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

   This tells the Handle server to get information about individual Handles from the DSpace code.
3. Once the configuration file has been generated, you will need to go to http://hdl.handle.net/4263537/5014 to upload the generated sitebndl.zip file. The upload page will ask you for your contact information. An administrator will then create the naming authority/prefix on the root service (known as the Global Handle Registry), and notify you when this has been completed. You will not be able to continue the handle server installation until you receive further information concerning your naming authority.
4. When CNRI has sent you your naming authority prefix, you will need to edit the *config.dct* file. The file will be found in */[dspace]/handle-server*. Look for *"300:0.NA/YOUR_NAMING_AUTHORITY"*. Replace *YOUR_NAMING_AUTHORITY* with the assigned naming authority prefix sent to you.
5. Now start your handle server (as the dspace user):

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

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

Updating Existing Handle Prefixes

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

4.4.5 Google and HTML sitemaps

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

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

To generate the sitemaps, you need to run *[dspace]/bin/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-space]/config/dspace.cfg* using the *sitemap.engineurls* setting where you can provide a comma-separated list of URLs to 'ping'.

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

```
# Generate sitemaps


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

4.4.6 DSpace Statistics

DSpace uses the Apache Solr application underlaying 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 (see page 285) for detailed information.

1. **DSpace Configuration for Accessing Solr.** In the *dspace.cfg* file review the following fields to make sure they are uncommented:

```
solr.log.server = ${dspace.baseUrl}/solr/statistics
solr.dbfile = ${dspace.dir}/config/GeoLiteCity.dat
solr.spiderips.urls = 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
```

2. **DSpace logging configuration for Solr.** If your DSpace instance is protected by a proxy server, in order for Solr 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 the following setting is uncommented in the logging section of *dspace.cfg*:

```
useProxies = true
```

3. **Configuration Control.** In the *dspace.cfg* set the following property key:_statistics.item.authorization.admin=true_This will require the user to sign on to see that statistics. Setting the statistics to "false" will make them publicly available.
4. Final steps.
    - Perform the following step:

```
cd [dspace-source]/dspace
mvn package
cd [dspace-source]/dspace/target/dspace-<version>-build.dir
ant -Dconfig=[dspace]/config/dspace.cfg update
cp -R [dspace]/webapps/* [TOMCAT]/webapps
```

      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*
    - Restart your webapps (Tomcat/Jetty/Resin)

4.4.7 Manually Installing/Updating GeoLite Database File

The GeoLite Database file (at [dspace]/config/GeoLiteCity.dat) is used by the DSpace Statistics (see page 57) 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 two 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`.

4.5 Windows Installation

4.5.1 Pre-requisite Software

If you are installing DSpace on Windows, you will still need to install all the same Prerequisite Software (see page 59), as listed above.

- 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.

4.5.2 Installation Steps

1. Download the DSpace source from SourceForge and unzip it (WinZip will do this)
2. Ensure the PostgreSQL service is running, and then run pgAdmin III (Start -> PostgreSQL 8.0 -> pgAdmin III). Connect to the local database as the postgres user and:
   - Create a 'Login Role' (user) called *dspace* with the password *dspace*
   - Create a database called *dspace* owned by the user *dspace*, with UTF-8 encoding

3. Update paths in *[dspace-source]\dspace\config\dspace.cfg*
    - **Note:** Use forward slashes / for path separators, though you can still use drive letters, e.g.: *dspace.dir = C:/DSpace* Also, make sure you change all of the parameters with file paths to suit, specifically:

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

4. Create the directory for the DSpace installation (e.g. *C:/DSpace*)
5. Generate the DSpace installation package by running the following from command line (cmd) :

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

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

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

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

```
ant fresh_install
```

    - *Note:* to see a complete list of build targets, run: `ant help`
7. Create an administrator account, by running the following from your *[dspace]* (e.g. *C:\DSpace*) directory:

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

8. Copy the Web application directories from *[dspace]\webapps* to Tomcat's webapps dir, which should be somewhere like *C:\Program Files\Apache Software Foundation\Tomcat\webapps*
    - Alternatively, Tell your Tomcat installation where to find your DSpace web application(s). As an example, in the *<Host>* section of your *[tomcat]/conf/server.xml* you could add lines similar to the following (but replace *[dspace]* with your installation location):

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

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

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

4.6 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 test database command to see if other issues are being report: [dspace]/bin/dspace test-database*
- Email subsystem is running. The user can issue the following command to test the email system. t 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, will refer you to the DSpace documentation. *[dspace]/bin/test-email*

4.7 Known Bugs

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

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

Please see the 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.

4.8 Common Problems

In an ideal world everyone would follow the above steps and have a fully functioning DSpace. Of course, in the real world it doesn't always seem to work out that way. This section lists common problems that people encounter when installing DSpace, and likely causes and fixes. This is likely to grow over time as we learn about users' experiences.

4.8.1 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
bc1Connection.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-1.8.0-build.dir/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. You should be able to resolve this issue by following the "Manually Installing/Updating GeoLite Database File" (see page 58) instructions above.
  - 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" (see page 58) instructions above.

4.8.2 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.

5 Upgrading a DSpace Installation

This section describes how to upgrade a DSpace installation from one version to the next. Details of the differences between the functionality of each version are given in the Version History (see page 584) section.

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.

If you are upgrading across multiple versions

You should perform all of the steps of each upgrade between the version from which you are starting and the version to which you are upgrading. You do *not* need to install each intervening version, but you do need to carry out all of the configuration changes and additions, and all of the database updates, for each one. For example, when upgrading from 1.6.x to 1.8.x, you need to perform the configuration & database upgrade steps detailed in Upgrading From 1.6.x to 1.7.x (see page 78) followed by those detailed in Upgrading From 1.7.x to 1.8.x (see page 69).

5.1 Upgrading From 1.8 to 1.8.x

In the notes below `[dspace]` refers to the install directory for your existing DSpace installation, and `[dspace-source]` to the source directory for DSpace 1.8. Whenever you see these path references, be sure to replace them with the actual path names on your local system. You should also check the DSpace Release 1.8.0 Notes to see what changes are in this version.

Changes to the DSpace 1.8 Upgrade / Configuration Process

In DSpace 1.8.0, there have been a few significant changes to how you upgrade and configure DSpace. Notably:

- **The dspace.cfg has been "split up":** Many "module" configurations have now been moved out of the 'dspace.cfg' and into separate configuration files in the `[dspace]/config/modules/` directory.
  - Authentication Configurations (see page 234) are now in `[dspace]/config/modules/authenticate*.cfg` files
  - Batch Metadata Editing Configurations (see page 256) are now in the `[dspace]/config/modules/bulkedit.cfg` file
  - Discovery Configurations (see page 270) are now in the `[dspace]/config/modules/discovery.cfg` file
  - OAI-PMH / OAI-ORE Configurations (see page 299) are now in the `[dspace]/config/modules/oai.cfg` file
  - Solr Statistics Configurations (see page 285) are now in the `[dspace]/config/modules/solr-statistics.cfg` file
  - SWORD Configurations (see page 308) are now in `[dspace]/config/modules/sword*.cfg` files
  - *All other DSpace configurations are still in the* `dspace.cfg` *configuration file.*
- **Behavior of 'ant update' has changed:** The `ant update` upgrade command now defaults to **replacing** any existing configuration files (though the existing configuration files will first be backed up to a file with the suffix `*.old`).
  - In prior versions of DSpace (before 1.8.0), this `ant update` command would leave existing configuration files intact (and you would have to manually merge in new configuration settings, which would be in a file with the suffix `*.new`). If you prefer this previous behavior, you can still achieve the same result by running:
    - `ant -Doverwrite=false update`
  - **WARNING:** If you choose to run `ant -Doverwrite=false update` please be aware that this will not auto-upgrade any of your configuration files. This means you must closely watch the output of this command, and ensure you manually upgrade all configuration files in the `[dspace]/config/` directory as well as all Solr configurations/schemas in the `[dspace]/solr/search/conf/` and `[dspace]/solr/statistics/conf/` directories.
- **The structure of the source release has now been changed:** Please see Advanced Customisation (see page 346) for more details.

5.1.1 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.

5.1.2 Upgrade Steps

1. **Download DSpace 1.8.2** Either download DSpace 1.8.2 from DSpace.org or check it out directly from the SVN code repository. If you downloaded DSpace do not unpack it on top of your existing installation. Refer to Installation Instructions, Step 3 (see page 45) for unpacking directives.
2. **Merge any customizations.** If you have made any local customizations to your DSpace installation they will need to be migrated over to the new DSpace. Customizations are typically housed in one of the following places:
   - JSPUI modifications: `[dspace-source]/dspace/modules/jspui/src/main/webapp/`
   - XMLUI modifications: `[dspace-source]/dspace/modules/xmlui/src/main/webapp/`
   - Config modifications: `[dspace]/config`
3. **Build DSpace.** Run the following commands to compile DSpace:

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

   You will find the result in `[dspace-source]/dspace/target/dspace-[version]-build.dir` . Inside this directory is the compiled binary distribution of DSpace. Before rebuilding DSpace ('package'), the above command will clean out any previously compiled code ('clean') and ensure that your local DSpace JAR files are updated from the remote maven repository.

4. **Stop Tomcat.** Take down your servlet container. 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.)
5. **Update DSpace.**
    1. Update the DSpace installed directory with the new code and libraries. Issue the following commands:

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

        > **Changes to the behavior of the 'ant update' script**
        >
        > The `ant update` script has changed slightly as of DSpace 1.8. It now defaults to replacing your existing configuration files (after backing them up first). See the Changes to the DSpace 1.8 Upgrade / Configuration Process (see page 66) note at the top of this page for more details.

    2. No database changes have been made in either 1.8.1 or 1.8.2. So there is no need to update your existing 1.8 database.
6. **Update your DSpace Configurations.**
    1. There are no new configurations for the 1.8.1 or 1.8.2 releases. Your existing 1.8 configuration files should work fine
7. **Refresh Browse and Search Indexes**. Though there are not any database changes, it is a good policy to rebuild your search and browse indexes when upgrading to a new release. To do this, run the following command from your DSpace install directory (as the dspace user):

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

8. **Deploy Web Applications**. If necessary, copy the web applications files from your `[dspace]/webapps` directory to the subdirectory of your servlet container (e.g. tomcat):

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

    See the installation guide (see page 48) for full details.
9. **Restart servlet container.** Now restart your Tomcat/Jetty/Resin server program and test out the upgrade.

5.2 Upgrading From 1.7.x to 1.8.x



In the notes below `[dspace]` refers to the install directory for your existing DSpace installation, and `[dspace-source]` to the source directory for DSpace 1.8. Whenever you see these path references, be sure to replace them with the actual path names on your local system. You should also check the DSpace Release 1.8.0 Notes to see what changes are in this version.

Changes to the DSpace 1.8 Upgrade / Configuration Process

In DSpace 1.8.0, there have been a few significant changes to how you upgrade and configure DSpace. Notably:

- **The dspace.cfg has been "split up":** Many "module" configurations have now been moved out of the 'dspace.cfg' and into separate configuration files in the `[dspace]/config/modules/` directory.
  - Authentication Configurations (see page 234) are now in `[dspace]/config/modules/authenticate*.cfg` files
  - Batch Metadata Editing Configurations (see page 256) are now in the `[dspace]/config/modules/bulkedit.cfg` file
  - Discovery Configurations (see page 270) are now in the `[dspace]/config/modules/discovery.cfg` file
  - OAI-PMH / OAI-ORE Configurations (see page 299) are now in the `[dspace]/config/modules/oai.cfg` file
  - Solr Statistics Configurations (see page 285) are now in the `[dspace]/config/modules/solr-statistics.cfg` file
  - SWORD Configurations (see page 308) are now in `[dspace]/config/modules/sword*.cfg` files
  - *All other DSpace configurations are still in the* `dspace.cfg` *configuration file.*
- **Behavior of 'ant update' has changed:** The `ant update` upgrade command now defaults to **replacing** any existing configuration files (though the existing configuration files will first be backed up to a file with the suffix `*.old`).
  - In prior versions of DSpace (before 1.8.0), this `ant update` command would leave existing configuration files intact (and you would have to manually merge in new configuration settings, which would be in a file with the suffix `*.new`). If you prefer this previous behavior, you can still achieve the same result by running:
    - `ant -Doverwrite=false update`
  - **WARNING:** If you choose to run `ant -Doverwrite=false update` please be aware that this will not auto-upgrade any of your configuration files. This means you must closely watch the output of this command, and ensure you manually upgrade all configuration files in the `[dspace]/config/` directory as well as all Solr configurations/schemas in the `[dspace]/solr/search/conf/` and `[dspace]/solr/statistics/conf/` directories.
- **The structure of the source release has now been changed:** Please see Advanced Customisation (see page 346) for more details.

5.2.1 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.

5.2.2 Upgrade Steps

1. **Download DSpace 1.8** Either download DSpace 1.8 from DSpace.org or check it out directly from the SVN code repository. If you downloaded DSpace do not unpack it on top of your existing installation. Refer to Installation Instructions, Step 3 (see page 45) for unpacking directives.
2. **Merge any customizations.** If you have made any local customizations to your DSpace installation they will need to be migrated over to the new DSpace. Customizations are typically housed in one of the following places:
    - JSPUI modifications: `[dspace-source]/dspace/modules/jspui/src/main/webapp/`
    - XMLUI modifications: `[dspace-source]/dspace/modules/xmlui/src/main/webapp/`
    - Config modifications: `[dspace]/config`
3. **Build DSpace.** Run the following commands to compile DSpace:

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

   You will find the result in `[dspace-source]/dspace/target/dspace-[version]-build.dir` . Inside this directory is the compiled binary distribution of DSpace. Before rebuilding DSpace ('package'), the above command will clean out any previously compiled code ('clean') and ensure that your local DSpace JAR files are updated from the remote maven repository.

4. **Stop Tomcat.** Take down your servlet container. 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.)
5. **Update DSpace.**
    1. Update the DSpace installed directory with the new code and libraries. Issue the following commands:

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

> **Changes to the behavior of the 'ant update' script**
>
> The `ant update` script has changed slightly as of DSpace 1.8.0. It now defaults to replacing your existing configuration files (after backing them up first). See the Changes to the DSpace 1.8 Upgrade / Configuration Process (see page 70) note at the top of this page for more details.

    2. Apply database changes to your database by running one of the following database schema upgrade scripts.

> **Backup Your Database First**
>
> Applying a database change will alter your database! The database upgrade scripts have been tested, however, there is always a chance something could go wrong. So, do yourself a favor and create a backup of your database before you run a script that will alter your database.

        1. PostgreSQL: [dspace-source]/dspace/etc/postgres/database_schema_17-18.sql
        2. Oracle: [dspace-source]/dspace/etc/oracle/database_schema_17-18.sql
6. **Update your DSpace Configurations.**
    1. Merge existing configurations: After updating DSpace, you may notice a series of `*.old` files in your newly updated `[dspace]/config/` directory (and all sub-directories). During the update process, if there is a difference between your old 1.7-compatible configuration file and the new 1.8-compatible configuration file, your previous settings will be moved to a `*.old` file. You may want to review the differences between the `*.old` file and the new version of that file, and ensure your previous configurations/settings are merged into the new configuration file. One way to compare these files is by using a comparison-utility like `diff` or a text editor that supports file comparison.

2. Set New Configurations: There are new configuration settings in the new release that add or change functionality. You should review these new settings and ensure that they are set according to your needs.

1. New settings for Creative Commons licensing (see page 188) in `dspace.cfg`
2. New settings for RSS feeds (see "webui.feed.podcast.*" (see page 201)) in `dspace.cfg` which now support richer features, such as iTunes podcast and publishing to iTunesU
3. **Several major configuration sections have now been removed from the `dspace.cfg` and separated into their own config files.** Configuration sections which have been moved include Authentication settings, Batch Metadata Editing settings, Discovery settings, OAI-PMH/OAI-ORE settings, Statistics settings and SWORD settings. So, any configurations from these sections should be removed from your existing dspace.cfg file, as they will be ignored. For more information, see the Changes to the DSpace 1.8 Upgrade / Configuration Process (see page 70) note at the top of this page.
4. Several new configurations files have been created in the `[dspace]/config/modules/` directory. Each of these corresponds to a new feature in 1.8.0 (or a configuration section which has now been moved out of the dspace.cfg file):
    - `authentication-*.cfg` files : new location for Authentication Configurations (see page 234).
    - `bulkedit.cfg` : new location for Batch Metadata Editing Configurations (see page 256).
    - `discovery.cfg` : new location for Discovery Configurations (see page 270).
    - `fetchccdata.cfg` : configuration for new "Fetch CC Data" Curation Task (see page 394).
    - `oai.cfg` : new location for OAI-PMH / OAI-ORE Configurations (see page 299).
    - `solr-statistics.cfg` : new location for Solr Statistics Configurations (see page 285).
    - `spring.cfg` : configuration file for DSpace Service Manager (should not need modification).
    - `submission-curation.cfg` - configuration file for new Virus Scanning on Submission feature.
    - `sword-client.cfg` : configuration file for new SWORDv1 Client (see page 306) feature.
    - `sword-server.cfg` : new location for SWORDv1 Server Configurations (see page 308).
    - `swordv2-server.cfg` : configuration file for new SWORDv2 Server (see page 313) feature.
    - `translator.cfg` : configuration for new "Microsoft Translator" Curation Task (see page 409).
    - `workflow.cfg` : configuration for new Configurable Workflow (see page 257) feature.
5. Finally, there is a new `[dspace]/config/spring/` directory which holds Spring Framework configuration files. *The vast majority of users should never need to modify these settings, but they are available for hardcore developers who wish to add new features via the DSpace Services Framework (see page 507) (based on Spring Framework).*

7. **Generate Browse and Search Indexes**. The search mechanism has been updated in 1.8, so you must perform a full reindex of your site for searching and browsing to work. To do this, run the following command from your DSpace install directory (as the dspace user):

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

8. **Deploy Web Applications**. If necessary, copy the web applications files from your `[dspace]/webapps` directory to the subdirectory of your servlet container (e.g. tomcat):

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

   See the installation guide (see page 48) for full details.
9. **Restart servlet container.** Now restart your Tomcat/Jetty/Resin server program and test out the upgrade.

Optional Upgrade Step: Fix Broken File Statistics

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.

Updating the file statistics will ensure that old file downloads statistics data will also be filterable using the filter bundle feature. The benefit of upgrading is that only files within, for example, the "ORIGINAL" bundle are shown as opposed to also showing statistics from the LICENSE bundle. More information about this feature can be found at Statistics differences between DSpace 1.7.x and 1.8.0 (see page 291)



Backup Your statistics data first

Applying this change will involve dumping all the old file statistics into a file and re-loading them. Therefore it is wise to create a backup of the [DSpace]/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 option (`-r`) which will, if given, not only update the broken file statistics but also delete 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
```

5.3 Upgrading From 1.7 to 1.7.x

In the notes below `[dspace]` refers to the install directory for your existing DSpace installation, and `[dspace-source]` to the source directory for DSpace 1.7.2. Whenever you see these path references, be sure to replace them with the actual path names on your local system. Additionally, be sure to backup your configs, source code modifications, and database before doing a step that could destroy your instance.

5.3.1 Upgrade Steps

1. **Download DSpace 1.7.2** Either download DSpace 1.7.2 from DSpace.org or check it out directly from the SVN code repository. If you downloaded DSpace do not unpack it on top of your existing installation. Refer to Installation Instructions, Step 3 (see page 45) for unpacking directives.
2. **Apply any customizations.** If you have made any local customizations to your DSpace installation they will need to be migrated over to the new DSpace. These are housed in one of the following places:
   - JSPUI modifications: `[dspace-source]/dspace/modules/jspui/src/main/webapp/`
   - XMLUI modifications: `[dspace-source]/dspace/modules/xmlui/src/main/webapp/`
   - Config modifications: `[dspace]/config`
3. **Build DSpace.** Run the following commands to compile DSpace:

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

   You will find the result in `[dspace-source]/dspace/target/dspace-[version]-build.dir` . Inside this directory is the compiled binary distribution of DSpace. Before rebuilding DSpace ('package'), the above command will clean out any previously compiled code ('clean') and ensure that your local DSpace JAR files are updated from the remote maven repository.
4. **Stop Tomcat.** Take down your servlet container. For Tomcat, use the `$CATALINA/shutdown.sh` script. (Many Unix-based installations will have a startup/shutdown script in the `/etc/init.d` or `/etc/rc.d` directories.)
5. **Backup Your DSpace.** Make a complete backup of your system, including:
   - A snapshot of the database. *To have a "snapshot" of the PostgreSQL database use Postgres' pg_dump command.*
   - The asset store (`[dspace]/assetstore` by default, and any other assetstores configured in the `[dspace]/config/dspace.cfg` "assetstore.dir" and "assetstore.dir.#" settings)
   - Your configuration files and customizations to DSpace (including any customized scripts).

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

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

7. **Generate Browse and Search Indexes**. Though there are not any database changes between 1.7 and 1.7.1 release, it makes good policy to rebuild your search and browse indexes when upgrading to a new release. To do this, run the following command from your DSpace install directory (as the dspace user):

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

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

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

9. **Restart servlet container.** Now restart your Tomcat/Jetty/Resin server program and test out the upgrade.

5.4 Upgrading From 1.6.x to 1.7.x



In the notes below `[dspace]` refers to the install directory for your existing DSpace installation, and *[dspace-source]* to the source directory for DSpace 1.7.x. Whenever you see these path references, be sure to replace them with the actual path names on your local system.

5.4.1 Upgrade Steps



Before upgrading you need to check you are using the current recommended minimum versions of Java (1.6), Maven (2.0.8 or above) and ant (1.7 or above). For more details, see the current listing of Prerequisite Software (see page 59)

1. **Backup Your DSpace.** First, and foremost, make a complete backup of your system, including:
   - A snapshot of the database. *To have a "snapshot" of the PostgreSQL database, you need to shut it down during the backup. You should also have your regular PostgreSQL Backup output (using Postgres' pg_dump command).*
   - The asset store (`[dspace]/assetstore` by default)
   - Your configuration files and customizations to DSpace (including any customized scripts).
2. **Download DSpace 1.7.x** Retrieve the new DSpace 1.7.x source code either as a download from DSpace.org or check it out directly from the SVN code repository. If you downloaded DSpace do not unpack it on top of your existing installation. Refer to Installation Instructions, Step 3 (see page 45) for unpacking directives.
3. **Stop Tomcat.** Take down your servlet container. For Tomcat, use the `$CATALINA_HOME/bin/shutdown.sh` script. (Many Unix-based installations will have a startup/shutdown script in the `/etc/init.d` or `/etc/rc.d` directories).
4. **Apply any customizations.** If you have made any local customizations to your DSpace installation they will need to be migrated over to the new DSpace. These are normally housed in one of the following places:
   - JSPUI modifications: `[dspace-source]/dspace/modules/jspui/src/main/webapp/`
   - XMLUI modifications: `[dspace-source]/dspace/modules/xmlui/src/main/webapp/`
5. **Update Configuration Files.** Some parameters have changed and some are new. You can either attempt to make these changes in your current 1.6.x `dspace.cfg` file, or you can start with a new 1.7 `dspace.cfg` and re-modify it as needed. Configuration changes are noted below:
   - ***CORRECTION*** There was a missing hyphen "-" in the property key for mail character set:

     ```
     # Set the default mail character set. This may be over ridden by providing a line
     # inside the email template "charset: <encoding>", otherwise this default is used.
     #mail.charset = UTF-8
     ```

   - ***CORRECTION*** This was moved from the end of the solr configuration section to just under Logging Configurations:

     ```
     # If enabled, the logging and the solr statistics system will look for
     # an X-Forward header. If it finds it, it will use this for the user IP Address
     # useProxies = true
     ```

- *__CHANGE__* The MediaFilter is now able to process Power Point Text Extracter

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

#Assign 'human-understandable' names to each filter
plugin.named.org.dspace.app.mediafilter.FormatFilter = \
  org.dspace.app.mediafilter.PDFFilter = PDF Text Extractor, \
  org.dspace.app.mediafilter.HTMLFilter = HTML Text Extractor, \
  org.dspace.app.mediafilter.WordFilter = Word Text Extractor, \
  org.dspace.app.mediafilter.PowerPointFilter = PowerPoint Text Extractor, \
  org.dspace.app.mediafilter.JPEGFilter = JPEG Thumbnail, \
  org.dspace.app.mediafilter.BrandedPreviewJPEGFilter = Branded Preview JPEG

#Configure each filter's input format(s)
filter.org.dspace.app.mediafilter.PDFFilter.inputFormats = Adobe PDF
filter.org.dspace.app.mediafilter.HTMLFilter.inputFormats = HTML, Text
filter.org.dspace.app.mediafilter.WordFilter.inputFormats = Microsoft Word
filter.org.dspace.app.mediafilter.PowerPointFilter.inputFormats = Microsoft Powerpoint,
Microsoft Powerpoint XML
filter.org.dspace.app.mediafilter.JPEGFilter.inputFormats = BMP, GIF, JPEG, image/png
filter.org.dspace.app.mediafilter.BrandedPreviewJPEGFilter.inputFormats = BMP, GIF,
JPEG, image/png
```

- *__CHANGE__* The Crosswalk Plugin Configuration has changed with additional lines. Edit your file accordingly:

```
# Crosswalk Plugin Configuration:
#   The purpose of Crosswalks is to translate an external metadata format to/from
#   the DSpace Internal Metadata format (DIM) or the DSpace Database.
#   Crosswalks are often used by one or more Packager plugins (see below).
plugin.named.org.dspace.content.crosswalk.IngestionCrosswalk = \
  org.dspace.content.crosswalk.AIPDIMCrosswalk = DIM, \
  org.dspace.content.crosswalk.AIPTechMDCrosswalk = AIP-TECHMD, \
  org.dspace.content.crosswalk.PREMISCrosswalk = PREMIS, \
  org.dspace.content.crosswalk.OREIngestionCrosswalk = ore, \
  org.dspace.content.crosswalk.NullIngestionCrosswalk = NIL, \
  org.dspace.content.crosswalk.OAIDCIngestionCrosswalk = dc, \
  org.dspace.content.crosswalk.DIMIngestionCrosswalk = dim, \
  org.dspace.content.crosswalk.METSRightsCrosswalk = METSRIGHTS, \
  org.dspace.content.crosswalk.RoleCrosswalk = DSPACE-ROLES

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

plugin.named.org.dspace.content.crosswalk.StreamIngestionCrosswalk = \
  org.dspace.content.crosswalk.NullStreamIngestionCrosswalk = NULLSTREAM, \
  org.dspace.content.crosswalk.CreativeCommonsRDFStreamIngestionCrosswalk =
DSPACE_CCRDF, \
  org.dspace.content.crosswalk.LicenseStreamIngestionCrosswalk = DSPACE_DEPLICENSE

plugin.named.org.dspace.content.crosswalk.DisseminationCrosswalk = \
  org.dspace.content.crosswalk.AIPDIMCrosswalk = DIM, \
  org.dspace.content.crosswalk.AIPTechMDCrosswalk = AIP-TECHMD, \
  org.dspace.content.crosswalk.SimpleDCDisseminationCrosswalk = DC, \
  org.dspace.content.crosswalk.SimpleDCDisseminationCrosswalk = dc, \
  org.dspace.content.crosswalk.PREMISCrosswalk = PREMIS, \
  org.dspace.content.crosswalk.METSDisseminationCrosswalk = METS, \
  org.dspace.content.crosswalk.METSDisseminationCrosswalk = mets, \
  org.dspace.content.crosswalk.METSRightsCrosswalk = METSRIGHTS, \
  org.dspace.content.crosswalk.OREDisseminationCrosswalk = ore, \
  org.dspace.content.crosswalk.DIMDisseminationCrosswalk = dim, \
  org.dspace.content.crosswalk.RoleCrosswalk = DSPACE-ROLES
```

- *__NEW__*

```
plugin.named.org.dspace.content.crosswalk.StreamDisseminationCrosswalk = \
  org.dspace.content.crosswalk.CreativeCommonsRDFStreamDisseminationCrosswalk =
DSPACE_CCRDF, \
  org.dspace.content.crosswalk.CreativeCommonsTextStreamDisseminationCrosswalk =
DSPACE_CCTEXT, \
  org.dspace.content.crosswalk.LicenseStreamDisseminationCrosswalk = DSPACE_DEPLICENSE
```

- *__CHANGE__* The Packager Plugin Configuration has changed considerably. Carefully revise your configuration file:

```
Packager Plugin Configuration:
#   Configures the ingest and dissemination packages that DSpace supports.
#   These Ingester and Disseminator classes support a specific package file format
#   (e.g. METS) which DSpace understands how to import/export.  Each Packager
#   plugin often will use one (or more) Crosswalk plugins to translate metadata (see
above).
plugin.named.org.dspace.content.packager.PackageDisseminator = \
  org.dspace.content.packager.DSpaceAIPDisseminator = AIP, \
  org.dspace.content.packager.DSpaceMETSDisseminator = METS, \
  org.dspace.content.packager.RoleDisseminator = DSPACE-ROLES

plugin.named.org.dspace.content.packager.PackageIngester = \
  org.dspace.content.packager.DSpaceAIPIngester = AIP, \
  org.dspace.content.packager.PDFPackager  = Adobe PDF, PDF, \
  org.dspace.content.packager.DSpaceMETSIngester = METS, \
  org.dspace.content.packager.RoleIngester = DSPACE-ROLES
```

- *__CHANGE__* The Mets Ingester configuration has change and been updated. Carefully edit:

```
#### METS ingester configuration:
# These settings configure how DSpace will ingest a METS-based package

# Configures the METS-specific package ingesters (defined above)
# 'default' settings are specified by 'default' key

# Default Option to save METS manifest in the item: (default is false)
mets.default.ingest.preserveManifest = false

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

# Default crosswalk mappings
# Maps a METS 'mdtype' value to a DSpace crosswalk for processing.
# When the 'mdtype' value is same as the name of a crosswalk, that crosswalk
# will be called automatically (e.g. mdtype='PREMIS' calls the crosswalk named
# 'PREMIS', unless specified differently in below mapping)
# Format is 'mets.default.ingest.crosswalk.<mdType> = <DSpace-crosswalk-name>'
mets.default.ingest.crosswalk.DC = QDC
mets.default.ingest.crosswalk.DSpaceDepositLicense = DSPACE_DEPLICENSE
mets.default.ingest.crosswalk.Creative\ Commons = DSPACE_CCRDF
mets.default.ingest.crosswalk.CreativeCommonsRDF = DSPACE_CCRDF
mets.default.ingest.crosswalk.CreativeCommonsText = NULLSTREAM

# Locally cached copies of METS schema documents to save time on ingest.  This
# will often speed up validation & ingest significantly.  Before enabling
# these settings, you must manually cache all METS schemas in
```

```
# [dspace]/config/schemas/ (does not exist by default).  Most schema documents
# can be found on the http://www.loc.gov/ website.
# Enable the below settings to pull these *.xsd files from your local cache.
# (Setting format: mets.xsd.<abbreviation> = <namespace> <local-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

#### AIP Ingester & Disseminator Configuration
# These settings configure how DSpace will ingest/export its own
# AIP (Archival Information Package) format for backups and restores
# (Please note, as the DSpace AIP format is also METS based, it will also
# use many of the 'METS ingester configuration' settings directly above)

# AIP-specific ingestion crosswalk mappings
# (overrides 'mets.default.ingest.crosswalk' settings)
# Format is 'mets.dspaceAIP.ingest.crosswalk.<mdType> = <DSpace-crosswalk-name>'
mets.dspaceAIP.ingest.crosswalk.DSpaceDepositLicense = NULLSTREAM
mets.dspaceAIP.ingest.crosswalk.CreativeCommonsRDF = NULLSTREAM
mets.dspaceAIP.ingest.crosswalk.CreativeCommonsText = NULLSTREAM

# Create EPerson if necessary for Submitter when ingesting AIP (default=false)
# (by default, EPerson creation is already handled by 'DSPACE-ROLES' Crosswalk)
#mets.dspaceAIP.ingest.createSubmitter = false

## AIP-specific Disseminator settings
# These settings allow you to customize which metadata formats are exported in AIPs

# Technical metadata in AIP (exported to METS <techMD> section)
# Format is <label-for-METS>:<DSpace-crosswalk-name> [, ...] (label is optional)
# If unspecfied, defaults to "PREMIS"
aip.disseminate.techMD = PREMIS, DSPACE-ROLES

# Source metadata in AIP (exported to METS <sourceMD> section)
# Format is <label-for-METS>:<DSpace-crosswalk-name> [, ...] (label is optional)
# If unspecfied, defaults to "AIP-TECHMD"
aip.disseminate.sourceMD = AIP-TECHMD

# Preservation metadata in AIP (exported to METS <digipovMD> section)
# Format is <label-for-METS>:<DSpace-crosswalk-name> [, ...] (label is optional)
# If unspecified, defaults to nothing in <digiprovMD> section
#aip.disseminate.digiprovMD =

# Rights metadata in AIP (exported to METS <rightsMD> section)
# Format is <label-for-METS>:<DSpace-crosswalk-name> [, ...] (label is optional)
```

```
# If unspecified, default to adding all Licenses (CC and Deposit licenses),
# as well as METSRights information
aip.disseminate.rightsMD = DSpaceDepositLicense:DSPACE_DEPLICENSE, \
    CreativeCommonsRDF:DSPACE_CCRDF, CreativeCommonsText:DSPACE_CCTEXT, METSRIGHTS

# Descriptive metadata in AIP (exported to METS <dmdSec> section)
# Format is <label-for-METS>:<DSpace-crosswalk-name> [, ...] (label is optional)
# If unspecfied, defaults to "MODS, DIM"
aip.disseminate.dmd = MODS, DIM
```

- *__NEW__* A new property has been added to control the discovery index for the Event System Configuration:

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

- *__NEW__* License bundle display is now configurable. You are able to either display or suppress.

```
# whether to display the contents of the licence bundle (often just the deposit
# licence in standard DSpace installation
webui.licence_bundle.show = false
```

- *__CORRECTION__* Thumbnail generation. The width and height of generated thumbnails had a missing equal sign.

```
# maximum width and height of generated thumbnails
thumbnail.maxwidth  = 80
thumbnail.maxheight = 80
```

- *__CORRECTION and ADDITION__* Authority Control Settings have changed. Formerly called ChoiceAuthority, it is now referred to as DCInputAuthority.

```
## The DCInputAuthority plugin is automatically configured with every
## value-pairs element in input-forms.xml, namely:
##   common_identifiers, common_types, common_iso_languages
##
## The DSpaceControlledVocabulary plugin is automatically configured
## with every *.xml file in [dspace]/config/controlled-vocabularies,
## and creates a plugin instance for each, using base filename as the name.
## eg: nsi, srsc.
## Each DSpaceControlledVocabulary plugin comes with three configuration options:
# vocabulary.plugin._plugin_.hierarchy.store = <true|false>    # default: true
# vocabulary.plugin._plugin_.hierarchy.suggest = <true|false>  # default: true
# vocabulary.plugin._plugin_.delimiter = "<string>"            # default: "::"
##
## An example using "srsc" can be found later in this section

#plugin.selfnamed.org.dspace.content.authority.ChoiceAuthority = \
# org.dspace.content.authority.DCInputAuthority, \
# org.dspace.content.authority.DSpaceControlledVocabulary
```

- *__NEW__* Controls autocomplete for authority control

```
## demo: subject code autocomplete, using srsc as authority
## (DSpaceControlledVocabulary plugin must be enabled)
#choices.plugin.dc.subject = srsc
#choices.presentation.dc.subject = select
#vocabulary.plugin.srsc.hierarchy.store = true
#vocabulary.plugin.srsc.hierarchy.suggest = true
#vocabulary.plugin.srsc.delimiter = "::"
```

- *__NEW__* You are now able to order your bitstreams by sequence id or file name.

```
#### Ordering of bitstreams ####

## Specify the ordering that bitstreams are listed.
##
## Bitstream field to sort on.  Values: sequence_id or name. Default: sequence_id
#webui.bitstream.order.field = "sequence_id"

## Direction of sorting order. Values: DESC or ASC. Default: ASC
#webui.bitstream.order.direction = ASC
```

- *__NEW__* DSpace now includes a metadata mapping feature that makes repository content discoverable by Google Scholar:

```
##### Google Scholar Metadata Configuration #####
google-metadata.config = ${dspace.dir}/config/crosswalks/google-metadata.properties
google-metadata.enable = true
```

- *__NEW__* XMLUI is now able to concatenate CSS, JS and JSON files:

```
# 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.
#xmlui.theme.enableConcatenation = false

# 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.
#xmlui.theme.enableMinification = false
```

- *__NEW__* XMLUI Mirage Theme. This is a new theme with it's own configuration:

```
### Setings for Item lists in Mirage theme ###
# What should the emphasis be in the display of item lists?
# Possible values : 'file', 'metadata'. If your repository is
# used mainly for scientific papers 'metadata' is probably the
# best way. If you have a lot of images and other files 'file'
# will be the best starting point
# (metdata is the default value if this option is not specified)
#xmlui.theme.mirage.item-list.emphasis = file
```

- *__NEW__* OAI Response default change.

```
# DSpace by default uses 100 records as the limit for the oai responses.
# This can be altered by enabling the oai.response.max-records parameter
# and setting the desired amount of results.
oai.response.max-records = 100
```

- *__CHANGE__* EPDCX property key has been renamed.

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

- *__NEW__* New SOLR Statistic Property keys:

```
# Timeout for the resolver in the dns lookup
# Time in milliseconds, defaults to 200 for backward compatibility
# Your systems default is usually set in /etc/resolv.conf and varies
# between 2 to 5 seconds, to high a value might result in solr exhausting
# your connection pool
solr.resolver.timeout = 200

----

# Enable/disable logging of spiders in solr statistics.
# If false, and IP matches an address in solr.spiderips.urls, event is not logged.
# If true, event will be logged with the 'isBot' field set to true
# (see solr.statistics.query.filter.* for query filter options)
# Default value is true.
#solr.statistics.logBots = true
```

6. **Build DSpace.** Run the following commands to compile DSpace.:

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

   You will find the result in `[dspace-source]/dspace/target/dspace-[version]-build.dir` . Inside this directory is the compiled binary distribution of DSpace. Before rebuilding DSpace, the above command will clean out any previously compiled code ('clean') and ensure that your local DSpace JAR files are updated from the remote maven code repository.
7. **Update DSpace.** Update the DSpace installed directory with the new code and libraries. Issue the following commands:

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

8. **Update the Database.** You will need to run the 1.6.x to 1.7.x database upgrade script.
    - For PostgreSQL:

      ```
      psql -U [dspace-user] -f [dspace-source]/dspace/etc/postgres/database_schema_16-17.sql
      [database name]
      ```

      (Your database name is by default 'dspace'). Example:

      ```
      psql -U dspace -f [dspace-source]/dspace/etc/postgres/database_schema_16-17.sql dspace
      ```

    - For Oracle: Execute the upgrade script, e.g. with sqlplus, recording the output:
        1. Start SQL*Plus with `sqlplus [connect args]`
        2. Record the output: `SQL> spool 'upgrade.lst'`
        3. Run the upgrade script `SQL> @[dspace-source]/dspace/etc/oracle/database_schema_16-17.sql`
        4. Turn off recording of output: `SQL> spool off`
9. **Generate Browse and Search Indexes**. It's always good policy to rebuild your search and browse indexes when upgrading to a new release. To do this, run the following command from your DSpace install directory (as the 'dspace' user):

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

10. **Deploy Web Applications**. If your servlet container (e.g. Tomcat) is not configured to look for new web applications in your `[dspace]/webapps` directory, then you will need to copy the web applications files into the appropriate subdirectory of your servlet container. For example:

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

11. **Restart servlet container.** Now restart your Tomcat/Jetty/Resin server program and test out the upgrade.
12. **Add a new crontab entry**, or add to your system's scheduler, the following, run as the DSpace user, to enable routine maintenance of your SOLR indexes. If you do not run this command daily, it is likely your production instances of DSpace will exhaust the available memory in your servlet container

    ```
    [dspace]/bin/dspace stats-util -o
    ```

5.5 Upgrading From 1.6 to 1.6.x

In the notes below `[dspace]` refers to the install directory for your existing DSpace installation, and `[dspace-source]` to the source directory for DSpace 1.6.1. Whenever you see these path references, be sure to replace them with the actual path names on your local system.

5.5.1 Upgrade Steps

1. **Backup Your DSpace.** First, and foremost, make a complete backup of your system, including:
   - A snapshot of the database. *To have a "snapshot" of the PostgreSQL database, you need to shut it down during the backup. You should also have your regular PostgreSQL Backup output (using pg_dump commands).*
   - The asset store (*[dspace]/assetstore* by default)
   - Your configuration files and customizations to DSpace (including any customized scripts).
2. **Download DSpace 1.6.2** Retrieve the new DSpace 1.6.2 source code either as a download from DSpace.org or check it out directly from the SVN code repository. If you downloaded DSpace do not unpack it on top of your existing installation. Refer to Installation Instructions, Step 3 (see page 45) for unpacking directives.
3. **Stop Tomcat.** Take down your servlet container. For Tomcat, use the *$CATALINA/shutdown.sh* script. (Many installations will have a startup/shutdown script in the */etc/init.d* or */etc/rc.d* directories.
4. **Apply any customizations.** If you have made any local customizations to your DSpace installation they will need to be migrated over to the new DSpace. These are housed in one of the following places:
   - JSPUI modifications: `[dspace-source]/dspace/modules/jspui/src/main/webapp/`
   - XMLUI modifications: `[dspace-source]/dspace/modules/xmlui/src/main/webapp/`
5. **Update Configuration Files.** There are no additions to this release. So you do not have to update the configuration files.
6. **Build DSpace.** Run the following commands to compile DSpace.:

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

   You will find the result in *[dspace-source]/dspace/target/dspace-[version]-build.dir* . Inside this directory is the compiled binary distribution of DSpace. Before rebuilding DSpace, the above command will clean out any previously compiled code ('clean') and ensure that your local DSpace JAR files are updated from the remote maven repository.
7. **Update DSpace.** Update the DSpace installed directory with the new code and libraries. Issue the following commands:

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

8. **Run Registry Format Update for CC License.** Creative Commons licenses have been assigned the wrong mime-type in past versions of DSpace. Even if you are not currently using CC Licenses, you should update your Bitstream Format Registry to include a new entry with the proper mime-type. To update your registry, run the following command: *dspace]/bin/dspace registry-loader -bitstream [dspace]/etc/upgrades/15-16/new-bitstream-formats.xml*
9. **Update the Database.** If you are using Creative Commons Licenses in your DSpace submission process, you will need to run the 1.5.x to 1.6.x database upgrade script again. In 1.6.0 the improper mime-type was being assigned to all CC Licenses. This has now been resolved, and rerunning the upgrade script will now assign the proper mime-type to all existing CC Licenses in your DSpace installation. NOTE: You will receive messages that most of the script additions already exist. This is normal, and nothing to be worried about.
    - For PostgreSQL: *psql -U [dspace-user] -f [dspace-source]/dspace/etc/postgres/database_schema_15-16.sql [database name]* (Your database name is by default 'dspace'). Example:

      ```
      psql -U dspace -f
         /dspace-1.6-1-src-release/dspace/etc/postgres/database_schema_15-16.sql dspace
      ```

      (The line break above is cosmetic. Please place your command in one line.
    - For Oracle: Execute the upgrade script, e.g. with sqlplus, recording the output:
        1. Start SQL*Plus with `sqlplus [connect args]`
        2. Record the output: `SQL> spool 'upgrade.lst'`
        3. Run the upgrade script `SQL> @[dspace-source]/dspace/etc/oracle/database_schema_15-16.sql`
        4. Turn off recording of output: `SQL> spool off`
        5. Please note: The final few statements WILL FAIL. That is because you have run some queries and use the results to construct the statements to remove the constraints, manually, Oracle doesn't have any easy way to automate this (unless you know PL/SQL). So, look for the comment line beginning:

           ```
           --You need to remove the already in place constraints
           ```

           and follow the instructions in the actual SQL file. Refer to the contents of the spool file "upgrade.lst" for the output of the queries you'll need.
10. **Generate Browse and Search Indexes**. Though there are not any database changes in the 1.6 to 1.6.1 release, it makes good policy to rebuild your search and browse indexes when upgrading to a new release. To do this, run the following command from your DSpace install directory (as the dspace user): *[dspace]/bin/dspace index-init*
11. **Deploy Web Applications**. Copy the web applications files from your *[dspace]/webapps* directory to the subdirectory of your servlet container (e.g. tomcat):*cp -R [dspace]/webapps/* [tomcat]/webapps/*
12. **Restart servlet.** Now restart your Tomcat/Jetty/Resin server program and test out the upgrade.

5.6 Upgrading From 1.5.x to 1.6.x



In the notes below `[dspace]` refers to the install directory for your existing DSpace installation, and *[dspace-source]* to the source directory for DSpace 1.6. Whenever you see these path references, be sure to replace them with the actual path names on your local system.

5.6.1 Upgrade Steps

1. **Backup Your DSpace.** First, and foremost, make a complete backup of your system, including:
   - A snapshot of the database. _To have a "snapshot" of the PostgreSQL database, you need to shut it down during the backup. You should also have your regular PostgreSQL Backup output (using pg_dump commands). _
   - The asset store (*[dspace]/assetstore* by default)
   - Your configuration files and customizations to DSpace (including any customized scripts).
2. **Download DSpace 1.6.x** Retrieve the new DSpace 1.6.x source code either as a download from DSpace.org or check it out directly from the SVN code repository. If you downloaded DSpace do not unpack it on top of your existing installation. Refer to Installation Instructions, Step 3 (see page 45) for unpacking directives.
3. **Stop Tomcat**. Take down your servlet container. For Tomcat, use the *$CATALINA/shutdown.sh* script. (Many installations will have a startup/shutdown script in the */etc/init.d* or */etc/rc.d* directories.
4. **Apply any customizations**. If you have made any local customizations to your DSpace installation they will need to be migrated over to the new DSpace. These are housed in one of the following places:
   - JSPUI modifications: `[dspace-source]/dspace/modules/jspui/src/main/webapp/`
   - XMLUI modifications: `[dspace-source]/dspace/modules/xmlui/src/main/webapp/`
5. **Update Configuration Files**. Some of the parameters have change and some are new. Changes will be noted below:

- **CHANGE** The base url and oai urls property keys are set differently

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

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

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

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

- **NEW** New email options (Add these at the end of the "Email Settings" sub-section):

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

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

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

- **NEW**New Authorization levels and parameters. See the Configuration (see page 137) documentation, "Delegation Administration" section for further information.

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

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

# COLLECTION ADMIN
#core.authorization.collection-admin.policies = true
#core.authorization.collection-admin.template-item = true
#core.authorization.collection-admin.submitters = true
#core.authorization.collection-admin.workflows = true
#core.authorization.collection-admin.admin-group = true
# item owned by his collection
#core.authorization.collection-admin.item.delete = true
#core.authorization.collection-admin.item.withdraw = true
#core.authorization.collection-admin.item.reinstatiate = true
#core.authorization.collection-admin.item.policies = true
# also bundle...
#core.authorization.collection-admin.item.create-bitstream = true
#core.authorization.collection-admin.item.delete-bitstream = true
#core.authorization.collection-admin.item-admin.cc-license = true

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

- **CHANGE** METS ingester has been revised. (Modify In "*Crosswalk and Packager Plugin Settings*")

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

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

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

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

- **CHANGE** Event Settings have had the following revision with the addition of 'harvester' (modify in "*Event System Configuration*"):

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

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

also:

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

- **NEW** New option for the Embargo of Thesis and Dissertations.

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

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

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

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

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

- **NEW** New option for using the Batch Editing capabilities. See Batch Metadata Editing Configuration (see page) and also System Administration : Batch Metadata Editing (see page)

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

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

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

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

- **NEW** Ability to hide metadata fields is now available. (Look for "*JSPUI & XMLUI Configurations* " Section)

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

- **NEW**Choice Control and Authority Control options are available (Look for "*JSPUI & XMLUI Configurations*" Section):

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

And also:

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

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

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

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

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

##
## This sets the default lowest confidence level at which a metadata value is included
```

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

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


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

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

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

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

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

- **REPLACE** RSS Feeds now support Atom 1.0. *Replace its previous configuration with the one below:*

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

```
# enable syndication feeds - links display on community and collection home pages
# (This setting is not used by XMLUI, as you enable feeds in your theme)
webui.feed.enable = false
# number of DSpace items per feed (the most recent submissions)
webui.feed.items = 4
# maximum number of feeds in memory cache
# value of 0 will disable caching
webui.feed.cache.size = 100
# number of hours to keep cached feeds before checking currency
# value of 0 will force a check with each request
webui.feed.cache.age = 48
# which syndication formats to offer
# use one or more (comma-separated) values from list:
# rss_0.90, rss_0.91, rss_0.92, rss_0.93, rss_0.94, rss_1.0, rss_2.0
webui.feed.formats = rss_1.0,rss_2.0,atom_1.0
# URLs returned by the feed will point at the global handle server (e.g.
http://hdl.handle.net/123456789/1)
# Set to true to use local server URLs (i.e. http://myserver.myorg/handle/123456789/1)
webui.feed.localresolve = false

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

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

# Customize the extra namespaced DC elements added to the item (RSS) or entry
# (Atom) element.  These let you include individual metadata values in a
# structured format for easy extraction by the recipient, instead of (or in
# addition to) appending these values to the Description field.
## dc:creator value(s)
#webui.feed.item.dc.creator = dc.contributor.author
```

```
## dc:date value (may be contradicted by webui.feed.item.date)
#webui.feed.item.dc.date = dc.date.issued
## dc:description (e.g. for a distinct field that is ONLY the abstract)
#webui.feed.item.dc.description = dc.description.abstract

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

- **NEW** Opensearch Feature is new to DSpace

```
#### OpenSearch Settings ####
# NB: for result data formatting, OpenSearch uses Syndication Feed Settings
# so even if Syndication Feeds are not enabled, they must be configured
# enable open search
websvc.opensearch.enable = false
# context for html request URLs - change only for non-standard servlet mapping
websvc.opensearch.uicontext = simple-search
# context for RSS/Atom request URLs - change only for non-standard servlet mapping
websvc.opensearch.svccontext = open-search/
# present autodiscovery link in every page head
websvc.opensearch.autolink = true
# number of hours to retain results before recalculating
websvc.opensearch.validity = 48
# short name used in browsers for search service
# should be 16 or fewer characters
websvc.opensearch.shortname = DSpace
# longer (up to 48 characters) name
websvc.opensearch.longname = ${dspace.name}
# brief service description
websvc.opensearch.description = ${dspace.name} DSpace repository
# location of favicon for service, if any must be 16X16 pixels
websvc.opensearch.faviconurl = http://www.dspace.org/images/favicon.ico
# sample query - should return results
websvc.opensearch.samplequery = photosynthesis
# tags used to describe search service
websvc.opensearch.tags = IR DSpace
# result formats offered - use 1 or more comma-separated from: html,atom,rss
# NB: html is required for autodiscovery in browsers to function,
# and must be the first in the list if present
websvc.opensearch.formats = html,atom,rss
```

- **NEW* *Exposure of METS metadata can be now hidden. (See "*OAI-PMH SPECIFIC CONFIGURATIONS*" in the *dspace.cfg* file)

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

- **NEW* *SWORD has added the following to accept MIME/types. (See "*SWORD Specific Configurations*" Section)

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

- **NEW* *New OAI Harvesting Configuration settings are now available. (See "*OAI Harvesting Configurations*"

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

### Harvester settings

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

# This field works in much the same way as harvester.oai.metadataformats.PluginName
# The {name} must correspond to a declared ingestion crosswalk, while the
# {namespace} must be supported by the target OAI-PMH provider when harvesting content.
# harvester.oai.oreSerializationFormat.{name} = {namespace}

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

# 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.
```

```
#harvester.timePadding = 120

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

# 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 harvestFrequency) value. The scheduler is optimized to then sleep until the
# next collection is actually ready to be harvested. The minHeartbeat and
# maxHeartbeat are the lower and upper bounds on this timeframe. Measured in seconds.
# Default minHeartbeat is 30.  Default maxHeartbeat is 3600.
#harvester.minHeartbeat = 30
#harvester.maxHeartbeat = 3600

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

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

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

# The webapp responsible for minting the URIs for ORE Resource Maps.
# If using oai, the dspace.oai.uri config value must be set.
# The URIs generated for ORE ReMs follow the following convention for both cases.
# format: [baseURI]/metadata/handle/[theHandle]/ore.xml
# Default value is oai
#ore.authoritative.source = oai

# A harvest process will attempt to scan the metadata of the incoming items
# (dc.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: hdl.handle.net
#harvester.acceptedHandleServer = hdl.handle.net, handle.myu.edu

# 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: 123456789.
#harvester.rejectedHandlePrefix = 123456789, myTestHandle
```

- **NEW** SOLR Statistics Configurations. For a little more detailed information regarding the configuration, please refer to DSpace SOLR Statistics Configuration (see page 225) ; or, for installation procedures, refer to Advanced Installation: Dspace Statistics (see page 57).

```
#---------------------------------------------------------------#
#--------------SOLR STATISTICS CONFIGURATIONS-------------------#
#---------------------------------------------------------------#
# These configs are only used by the SOLR interface/webapp to   #
# track usage statistics.                                       #
#---------------------------------------------------------------#

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

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

6. **Build DSpace.** Run the following commands to compile DSpace.:

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

You will find the result in *[dspace-source]/dspace/target/dspace-[version]-build.dir*. Inside this directory is the compiled binary distribution of DSpace .Before rebuilding DSpace, the above command will clean out any previously compiled code ('clean') and ensure that your local DSpace JAR files are updated from the remote maven repository.

7. **Update the database**. The database schema needs to be updated to accommodate changes to the database. SQL files contain the relevant updates are provided. Please note that if you have made any local customizations to the database schema, you should consult these updates and make sure they will work for you.
    - For PostgreSQL: `psql -U [dspace-user] -f [dspace-source]/dspace/etc/postgres/database_schema_15-16.sql [database name]` (Your database name is by default 'dspace'). Example: `psql -U dspace -f /dspace-1.6-1-src-release/dspace/etc/postgres/database_schema_15-16.sql dspace`
    - For Oracle: Execute the upgrade script, e.g. with sqlplus, recording the output:
        1. Start SQL*Plus with `sqlplus [connect args]`
        2. Record the output: `SQL> spool 'upgrade.lst'`
        3. Run the upgrade script `SQL> @[dspace-source]/dspace/etc/oracle/database_schema_15-16.sql`
        4. Turn off recording of output: `SQL> spool off`
        5. Please note: The final few statements WILL FAIL. That is because you have run some queries and use the results to construct the statements to remove the constraints, manually; Oracle doesn't have any easy way to automate this (unless you know PL/SQL). So, look for the comment line beginning:

            ```
            --You need to remove the already in place constraints
            ```

            and follow the instructions in the actual SQL file. Refer to the contents of the spool file "upgrade.lst" for the output of the queries you'll need.
8. **Update DSpace**. Update the DSpace installed directory with the new code and libraries. Issue the following commands:

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

9. **Update Registry for the CC License.** If you use the CC License, an incorrect mime-type type is being assigned. You will need to run the following step: _dspace]/bin/dspace registry-loader -bitstream [dspace]/etc/upgrades/15-16/new-bitstream-formats.xml _
10. **Generate Browse and Search Indexes**. It makes good policy to rebuild your search and browse indexes when upgrading to a new release. Almost every release has database changes and indexes can be affected by this. In the DSpace 1.6 release there is Authority Control features and those will need the indexes to be regenerated. To do this, run the following command from your DSpace install directory (as the dspace user):*[dspace]/bin/dspace index-init*
11. **Deploy Web Applications**. Copy the web applications files from your *[dspace]/webapps* directory to the subdirectory of your servlet container (e.g. tomcat):*cp -R [dspace]/webapps/* [tomcat]/webapps/*
12. **Restart servlet.** Now restart your Tomcat/Jetty/Resin server program and test out the upgrade.

13. **Rolling Log Appender Upgrade**. You will want to upgrade your logs to the new format to use the SOLR Statistics now included with DSpace. While the commands for this are found in Chapter 8, here are the steps needed to be performed.

```
[dspace]/bin/dspace stats-log-converter -i input file name -o output file name -m (if you
have more than one dspace.log file)
[dspace]/bin/dspace stats-log-importer -i input file name (probably the output name from
above) -m
```

The user is highly recommended to see the System Administration : DSpace Log Converter (see page) documentation.

5.7 Upgrading From 1.5 or 1.5.1 to 1.5.2



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

5.7.1 Upgrade Steps

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

1. **Backup your DSpace** First and foremost, make a complete backup of your system, including:
   - A snapshot of the database
   - The asset store (*[dspace]/assetstore* by default)
   - Your configuration files and customizations to DSpace
   - Your statistics scripts (*[dspace]/bin/stat**) which contain customizable dates
2. **Download DSpace 1.5.2** Get the new DSpace 1.5.2 source code either as a download from DSpace.org or check it out directly from the SVN code repository. If you downloaded DSpace do not unpack it on top of your existing installation.
3. **Build DSpace** Run the following commands to compile DSpace.

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

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

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

5. **Apply any customizations** If you have made any local customizations to your DSpace installation they will need to be migrated over to the new DSpace. Commonly these modifications are made to "JSP" pages located inside the *[dspace 1.4.2]/jsp/local* directory. These should be moved *[dspace-source]/dspace/modules/jspui/src/main/webapp/* in the new build structure. See Customizing the JSP Pages for more information.
6. **Update DSpace** Update the DSpace installed directory with new code and libraries. Inside the *[dspace-source]/dspace/target/dspace-1.5-build.dir/* directory run:

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

7. **Update configuration files** This ant target preserves existing files in *[dspace]/config _ and will copy any new configuration files in place. If an existing file prevents copying the new file in place, the new file will have the suffix _.new*, for example *[dspace]/local/dspace.cfg.new*. Note: there is also a configuration option *-Doverwrite=true* which will instead copy the conflicting target files to *.old suffixes and overwrite target file then with the new file (essentially the opposite) this is beneficial for developers and those who use the *[dspace-source]/dspace/config* to maintain their changes.

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

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

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

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

- Special groups for LDAP and password authentication.

```
##### Password users group #####

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

##### LDAP users group #####

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

- new option for case insensitivity in browse tables.

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

- New usage event handler for collecting statistics:

```
### Usage event settings ###
# The usage event handler to call.  The default is the "passive" handler, which ignores
events.
# plugin.single.org.dspace.app.statistics.AbstractUsageEvent = \
#   org.dspace.app.statistics.PassiveUsageEvent
```

- The location where sitemaps are stored is now configurable.

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

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

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

- Hierarchical LDAP support.

```
##### Hierarchical LDAP Settings #####
# If your users are spread out across a hierarchical tree on your
# LDAP server, you will need to use the following stackable authentication
# class:
#  plugin.sequence.org.dspace.authenticate.AuthenticationMethod = \
#        org.dspace.authenticate.LDAPHierarchicalAuthentication
#
# You can optionally specify the search scope. If anonymous access is not
# enabled on your LDAP server, you will need to specify the full DN and
# password of a user that is allowed to bind in order to search for the
# users.

# This is the search scope value for the LDAP search during
# autoregistering. This will depend on your LDAP server setup.
# This value must be one of the following integers corresponding
# to the following values:
# object scope : 0
# one level scope : 1
# subtree scope : 2
#ldap.search_scope = 2

# The full DN and password of a user allowed to connect to the LDAP server
# and search for the DN of the user trying to log in.  If these are not specified,
# the initial bind will be performed anonymously.
#ldap.search.user = cn=admin,ou=people,o=myu.edu
#ldap.search.password = password

# If your LDAP server does not hold an email address for a user, you can use
# the following field to specify your email domain. This value is appended
# to the netid in order to make an email address. E.g. a netid of 'user' and
# ldap.netid_email_domain as '@example.com' would set the email of the user
# to be 'user@example.com
#ldap.netid_email_domain = @example.com
```

- Shibboleth authentication support.

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

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

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

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

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

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

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

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

# The following mappings specify role mapping between IdP and Dspace.
# the left side of the entry is IdP's role (prefixed with
```

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

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

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

  In configuration sections such as webui.itemdisplay.default, values can be changed from (e.g.) metadata.dc.identifier.doi to metadata.doi.dc.identifier.doi
- The whole of the SWORD configuration has changed. The SWORD section must be removed and replaced with

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

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

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

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

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

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

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

# The URL which identifies the sword software which provides
# the sword interface.  This is the URL which DSpace will use
# to fill out the atom:generator element of its atom documents.
#
# The default is:
#
# http://www.dspace.org/ns/sword/1.3.1
#
# If you have modified your sword software, you should change
# this URI to identify your own version.  If you are using the
# standard dspace-sword module you will not, in general, need
# to change this setting
#
# sword.generator.url = http://www.dspace.org/ns/sword/1.3.1

# The metadata field in which to store the updated date for
# items deposited via SWORD.
#
sword.updated.field = dc.date.updated

# The metadata field in which to store the value of the slug
# header if it is supplied
#
sword.slug.field = dc.identifier.slug

# the accept packaging properties, along with their associated
# quality values where appropriate.
#
# Global settings; these will be used on all DSpace collections
#
sword.accept-packaging.METSDSpaceSIP.identifier =
http://purl.org/net/sword-types/METSDSpaceSIP
sword.accept-packaging.METSDSpaceSIP.q = 1.0

# Collection Specific settings: these will be used on the collections
# with the given handles
#
# sword.accept-packaging.[handle].METSDSpaceSIP.identifier =
http://purl.org/net/sword-types/METSDSpaceSIP
# sword.accept-packaging.[handle].METSDSpaceSIP.q = 1.0
```

```
# Should the server offer up items in collections as sword deposit
# targets.  This will be effected by placing a URI in the collection
# description which will list all the allowed items for the depositing
# user in that collection on request
#
# NOTE: this will require an implementation of deposit onto items, which
# will not be forthcoming for a short while
#
sword.expose-items = false

# Should the server offer as the default the list of all Communities
# to a Service Document request.  If false, the server will offer
# the list of all collections, which is the default and recommended
# 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
#
sword.expose-communities = false

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

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

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

# Should the server identify the sword version in deposit response?
#
```

```
# It is recommended to leave this enabled.
#
sword.identify-version = true

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

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

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

5.8 Upgrading From 1.4.2 to 1.5



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

5.8.1 Upgrade Steps

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

1. **Backup your DSpace** First and foremost, make a complete backup of your system, including:
   - A snapshot of the database
   - The asset store (*[dspace]/assetstore* by default)
   - Your configuration files and customizations to DSpace
   - Your statistics scripts (*[dspace]/bin/stat**) which contain customizable dates
2. **Download DSpace 1.5.x** Get the new DSpace 1.5 source code either as a download from SourceForge or check it out directly from the SVN code repository. If you downloaded DSpace do not unpack it on top of your existing installation.
3. **Build DSpace** The build process has radically changed for DSpace 1.5. With this new release the build system has moved to a maven-based system enabling the various projects (JSPUI, XMLUI, OAI, and Core API) into separate projects. See the Installation (see page 38) section for more information on building DSpace using the new maven-based build system. Run the following commands to compile DSpace.

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

   You will find the result in *[dspace-source]/dspace/target/dspace-1.5-build.dir/*; inside this directory is the compiled binary distribution of DSpace.
4. **Stop Tomcat** Take down your servlet container, for Tomcat use the *bin/shutdown.sh* script.
5. **Update *dspace.cfg*** Several new parameters need to be added to your *[dspace]/config/dspace.cfg*. While it is advisable to start with a fresh *DSpace 1.5 _dspace.cfg* configuration file_ here are the minimum set of parameters that need to be added to an old *DSpace 1.4.2 configuration*.

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

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

    org.dspace.app.webui.util.CollectionStyleSelection


   ###### Browse Configuration ######
   #
   ```

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

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

# Sorting options
webui.itemlist.sort-option.1 = title:dc.title:title
webui.itemlist.sort-option.2 = dateissued:dc.date.issued:date
webui.itemlist.sort-option.3 =
 dateaccessioned:dc.date.accessioned:date

# Recent submissions
recent.submissions.count = 5

# Itemmapper browse index
itemmap.author.index = author

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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


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

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

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

11. **Update the database** The database schema needs updating. SQL files contain the relevant updates are provided, note if you have made any local customizations to the database schema you should consult these updates and make sure they will work for you.
    - For PostgreSQL *psql -U [dspace-user] -f [dspace-source]/dspace/etc/database_schema_14-15.sql [database-name]*
    - For Oracle *[dspace-source]/dspace/etc/oracle/database_schema_142-15.sql* contains the commands necessary to upgrade your database schema on oracle.
12. **Apply any customizations** If you have made any local customizations to your DSpace installation they will need to be migrated over to the new DSpace. Commonly these modifications are made to "JSP" pages located inside the *[dspace 1.4.2]/jsp/local* directory. These should be moved *[dspace-source]/dspace/modules/jspui/src/main/webapp/* in the new build structure. See Customizing the JSP Pages for more information.
13. **Update DSpace** Update the DSpace installed directory with new code and libraries. Inside the *[dspace-source]/dspace/target/dspace-1.5-build.dir/* directory run:

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

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

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

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

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

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

16. **Update statistics scripts** The statistics scripts have been rewritten for DSpace 1.5. Prior to 1.5 they were written in Perl, but have been rewritten in Java to avoid having to install Perl. First, make a note of the dates you have specified in your statistics scripts for the statistics to run from. You will find these in *[dspace]/bin/stat-initial*, as *$start_year* and *$start_month*. Note down these values.Copy the new stats scripts:

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

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

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

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

5.9 Upgrading From 1.4.1 to 1.4.2

5.9.1 Upgrade Steps

See Upgrading From 1.4 to 1.4.x (see page 118); the same instructions apply.

5.10 Upgrading From 1.4 to 1.4.x



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

5.10.1 Upgrade Steps

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

1. Get the new DSpace 1.4.x source code from the DSpace page on SourceForge and unpack it somewhere. Do not unpack it on top of your existing installation!!
2. Copy the PostgreSQL driver JAR to the source tree. For example:

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

3. **Note:** Licensing conditions for the *handle.jar* file have changed. As a result, the latest version of the *handle.jar* file is not included in this distribution. It is recommended you read the [new license conditions|http://www.handle.net/upgrade_6-2_DSpace.html] and decide whether you wish to update your installation's *handle.jar*. If you decide to update, you should replace the existing *handle.jar* in *[dspace-1.4.x-source]*/lib with the new version.
4. Take down Tomcat (or whichever servlet container you're using).
5. A new configuration item *webui.html.max-depth-guess* has been added to avoid infinite URL spaces. Add the following to the dspace.cfg file:

```
#### Multi-file HTML document/site settings #####
#
# When serving up composite HTML items, how deep can the request be for us to
# serve up a file with the same name?
#
# e.g. 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. Default value is 3.
#
webui.html.max-depth-guess = 3
```

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

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

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

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

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

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

5.11 Upgrading From 1.3.2 to 1.4.x



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

5.11.1 Upgrade Steps

1. First and foremost, **make a complete backup** of your system, including:
   - A snapshot of the database
   - The asset store (*[dspace]/assetstore* by default)
   - Your configuration files and localized JSPs
2. Download the latest DSpace 1.4.x source bundle and unpack it in a suitable location (*not* over your existing DSpace installation or source tree!)
3. Copy the PostgreSQL driver JAR to the source tree. For example:

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

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

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

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

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

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

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


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

plugin.sequence.org.dspace.app.mediafilter.MediaFilter = \
    org.dspace.app.mediafilter.PDFFilter,
 org.dspace.app.mediafilter.HTMLFilter, \
    org.dspace.app.mediafilter.WordFilter,
 org.dspace.app.mediafilter.JPEGFilter
# to enable branded preview: remove last line above, and uncomment 2
 lines below
#   org.dspace.app.mediafilter.WordFilter,
 org.dspace.app.mediafilter.JPEGFilter, \
#   org.dspace.app.mediafilter.BrandedPreviewJPEGFilter

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

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


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

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

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

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

  That needs to be changed to:

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

- If you use LDAP or X509 authentication, you'll need to add *org.dspace.eperson.LDAPAuthentication* or *org.dspace.eperson.X509Authentication* respectively. See also configuring custom authentication code.

- If you have custom Media Filters, note that these are now configured through *dspace.cfg* (instead of *mediafilter.cfg* which is obsolete.)
- Also, take a look through the default *dspace.cfg* file supplied with DSpace 1.4.x, as this contains configuration options for various new features you might like to use. In general, these new features default to 'off' and you'll need to add configuration properties as described in the default 1.4.x *dspace.cfg* to activate them.

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

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

9. The database schema needs updating. SQL files containing the relevant file are provided. If you've modified the schema locally, you may need to check over this and make alterations.
   - **For PostgreSQL**: *[dspace-1.4.x-source]/etc/database_schema_13-14.sql* contains the SQL commands to achieve this for PostgreSQL. To apply the changes, go to the source directory, and run:*psql -f etc/database_schema_13-14.sql [DSpace database name] -h localhost*
   - **For Oracle**: *[dspace-1.4.x-source]/etc/oracle/database_schema_13-14.sql* should be run on the DSpace database to update the schema.
10. Rebuild the search indexes: *[dspace]/bin/index-all*
11. Copy the *.war* Web application files in *[dspace-1.4-source]/build* to the *webapps* sub-directory of your servlet container (e.g. Tomcat). e.g.:

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

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

5.12 Upgrading From 1.3.1 to 1.3.2



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

5.12.1 Upgrade Steps

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

1. Get the new DSpace 1.3.2 source code from the DSpace page on SourceForge and unpack it somewhere. Do not unpack it on top of your existing installation!!
2. Copy the PostgreSQL driver JAR to the source tree. For example:

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

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

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

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

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

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

5.13 Upgrading From 1.2.x to 1.3.x



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

5.13.1 Upgrade Steps

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

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

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

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

8. Build and install the updated DSpace 1.3.x code. Go to the *[dspace-1.3.x-source]* directory, and run:*ant -Dconfig=[dspace]/config/dspace.cfg update*
9. You'll need to make some changes to the database schema in your PostgreSQL database. *[dspace-1.3.x-source]/etc/database_schema_12-13.sql* contains the SQL commands to achieve this. If you've modified the schema locally, you may need to check over this and make alterations. To apply the changes, go to the source directory, and run: *psql -f etc/database_schema_12-13.sql [DSpace database name] -h localhost*
10. Customize the stat generating statistics as per the instructions in System Statistical Reports
11. Initialize the statistics using: *[dspace]/bin/stat-initial[dspace]/bin/stat-general[dspace]/bin/stat-report-initial [dspace]/bin/stat-report-general*
12. Rebuild the search indexes: *[dspace]/bin/index-all*
13. Copy the *.war* Web application files in *[dspace-1.3.x-source]/build* to the *webapps* sub-directory of your servlet container (e.g. Tomcat). e.g.:*cp [dspace-1.3.x-source]/build/*.war [tomcat]/webapps*
14. Restart Tomcat.

5.14 Upgrading From 1.2.1 to 1.2.2

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

5.14.1 Upgrade Steps

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

1. Get the new DSpace 1.2.2 source code from the DSpace page on SourceForge and unpack it somewhere. Do not unpack it on top of your existing installation!!
2. Copy the PostgreSQL driver JAR to the source tree. For example:

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

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

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

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

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

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

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

   If you're using Tomcat, you need to delete the directories corresponding to the old *.war* files. For example, if *dspace.war* is installed in *[tomcat]/webapps/dspace.war*, you should delete the *[tomcat]/webapps/dspace* directory. Otherwise, Tomcat will continue to use the old code in that directory.
8. To finalize the install of the new configurable submission forms you need to copy the file *[dspace-1.2.2-source]/config/input-forms.xml* into *[dspace]/config*.
9. Restart Tomcat.

5.15 Upgrading From 1.2 to 1.2.1



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

5.15.1 Upgrade Steps

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

1. Get the new DSpace 1.2.1 source code from the DSpace page on SourceForge and unpack it somewhere. Do not unpack it on top of your existing installation!!
2. Copy the PostgreSQL driver JAR to the source tree. For example:

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

3. Take down Tomcat (or whichever servlet container you're using).
4. Your 'localized' JSPs (those in jsp/local) now need to be maintained in the *source* directory. If you have locally modified JSPs in your *[dspace]/jsp/local* directory, you might like to merge the changes in the new 1.2.1 versions into your locally modified ones. You can use the *diff* command to compare the 1.2 and 1.2.1 versions to do this. Also see the version history for a list of modified JSPs.

5. You need to add a few new parameters to your *[dspace]/dspace.cfg* for browse/search and item thumbnails display, and for configurable DC metadata fields to be indexed.

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

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

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

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


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

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

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

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

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

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

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

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

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

5.16 Upgrading From 1.1.x to 1.2

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

The process for upgrading to 1.2 from either 1.1 or 1.1.1 is the same. If you are running DSpace 1.0 or 1.0.1, you need to follow the instructions for Upgrading From 1.0.1 to 1.1 (see page 133) before following these instructions.

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

5.16.1 Upgrade Steps

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

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

4. Stop Tomcat (or other servlet container.)

5. It's a good idea to upgrade all of the various third-party tools that DSpace uses to their latest versions:
    - Java (note that now version 1.4.0 or later is *required*)
    - Tomcat (Any version after 4.0 will work; symbolic links are no longer an issue)
    - PostgreSQL (don't forget to build/download an updated JDBC driver .jar file! Also, **back up the database** first.)
    - Ant
6. You need to add the following new parameters to your *[dspace]/dspace.cfg*:

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

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

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

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

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

    Be sure to remove the <Context> elements for both the Web UI and the OAI Web applications.
8. Build and install the updated DSpace 1.2 code. Go to the DSpace 1.2 source directory, and run:

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

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

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

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

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

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

```
[dspace]/bin/dsrun
 org.dspace.administer.Upgrade11To12
```

12. Then rebuild the search indexes:

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

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

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

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

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

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

16. **Note 1**: This update process has effectively 'touched' all of your items. Although the dates in the Dublin Core metadata won't have changed (accession date and so forth), the 'last modified' date in the database for each will have been changed. This means the e-mail subscription tool may be confused, thinking that all items in the archive have been deposited that day, and could thus send a rather long email to lots of subscribers. So, it is recommended that you **turn off the e-mail subscription feature for the next day**, by commenting out the relevant line in DSpace's cron job, and then re-activating it the next day. Say you performed the update on 08-June-2004 (UTC), and your e-mail subscription cron job runs at 4am (UTC). When the subscription tool runs at 4am on 09-June-2004, it will find that everything in the system has a modification date in 08-June-2004, and accordingly send out huge emails. So, immediately after the update, you would edit DSpace's 'crontab' and comment out the */dspace/bin/subs-daily* line. Then, after 4am on 09-June-2004 you'd 'un-comment' it out, so that things proceed normally. Of course this means, any *real* new deposits on 08-June-2004 won't get e-mailed, however if you're updating the system it's likely to be down for some time so this shouldn't be a big problem.
17. **Note 2:** After consultation with the OAI community, various OAI-PMH changes have occurred:
    - The OAI-PMH identifiers have changed (they're now of the form *oai:hostname:handle* as opposed to just Handles)
    - The set structure has changed, due to the new sub-communities feature.
    - The default base URL has changed
    - As noted in note 1, every item has been 'touched' and will need re-harvesting. The above means that, if already registered and harvested, you will need to re-register your repository, effectively as a 'new' OAI-PMH data provider. You should also consider posting an announcement to the OAI implementers e-mail list so that harvesters know to update their systems. Also note that your site may, over the next few days, take quite a big hit from OAI-PMH harvesters. The resumption token support should alleviate this a little, but you might want to temporarily whack up the database connection pool parameters in *[dspace]/config/dspace.cfg*. See the *dspace.cfg* distributed with the source code to see what these parameters are and how to use them. (You need to stop and restart Tomcat after changing them.)I realize this is not ideal; for discussion as to the reasons behind this please see relevant posts to the OAI community: post one, post two. If you really can't live with updating the base URL like this, you can fairly easily have thing proceed more-or-less as they are, by doing the following:
    - Change the value of *OAI_ID_PREFIX* at the top of the *org.dspace.app.oai.DSpaceOAICatalog* class to *hdl:*
    - Change the servlet mapping for the *OAIHandler* servlet back to / (from */request*)
    - Rebuild and deploy _oai.war_However, note that in this case, all the records will be re-harvested by harvesters anyway, so you still need to brace for the associated DB activity; also note that the set spec changes may not be picked up by some harvesters. It's recommended you read the above-linked mailing list posts to understand why the change was made.
      Now, you should be finished!

5.17 Upgrading From 1.1 to 1.1.1

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

5.17.1 Upgrade Steps

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

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

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

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

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

5. Restart Tomcat.

5.18 Upgrading From 1.0.1 to 1.1

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

5.18.1 Upgrade Steps

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

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

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

CREATE INDEX eperson_email_idx ON EPerson(email);
CREATE INDEX item2bundle_item_idx on Item2Bundle(item_id);
CREATE INDEX bundle2bitstream_bundle_idx ON
 Bundle2Bitstream(bundle_id);
CREATE INDEX dcvalue_item_idx on DCValue(item_id);
CREATE INDEX collection2item_collection_idx ON
 Collection2Item(collection_id);
CREATE INDEX resourcepolicy_type_id_idx ON ResourcePolicy
 (resource_type_id,resource_id);
CREATE INDEX epersongroup2eperson_group_idx on
 EPersonGroup2EPerson(eperson_group_id);
CREATE INDEX handle_handle_idx ON Handle(handle);
CREATE INDEX sort_author_idx on ItemsByAuthor(sort_author);
CREATE INDEX sort_title_idx on ItemsByTitle(sort_title);
CREATE INDEX date_issued_idx on ItemsByDate(date_issued);

DROP VIEW CollectionItemsByDateAccessioned;

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

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

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

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

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

   (If you haven't modified any JSPs, you don't have to do anything.)
5. Copy the required Java libraries that we couldn't include in the bundle to the source tree. For example:

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

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

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

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

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

8. Run the collection default authorization policy tool:

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

9. Fix the OAICat properties file. Edit *[dspace]/config/templates/oaicat.properties*. Change the line that says

```
Identify.deletedRecord=yes
```

   To:

```
Identify.deletedRecord=persistent
```

   This is needed to fix the OAI-PMH 'Identity' verb response. Then run *[dspace]/bin/install-configs*.
10. Re-run the indexing to index abstracts and fill out the renamed database views:

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

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

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

6 Configuration

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 (see page 137) - addresses general conventions used with configuring not only the dspace.cfg file, but other configuration files which use similar conventions.
- The dspace.cfg Configuration Properties File (see page) - specifies the basic `dspace.cfg` file settings
- Optional or Advanced Configuration Settings (see page 226) - contain other more advanced settings that are optional in the dspace.cfg configuration file.

The full table of contents follows:

6.1 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)

6.1.1 Input Conventions

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

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

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

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

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

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

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

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

6.1.2 Update Reminder

Things you should know about editing `dspace.cfg` files.
It is important to remember that there are * two `dspace.cfg` files after an installation of DSpace.*

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

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

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

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

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

- Run `ant -Dconfig=[dspace]/config/dspace.cfg update` if you are updating your `dspace.cfg` file and wish to see the changes appear. Follow the usual sequence with copying your webapps.

- If you edit *dspace.cfg* in *[dspace-source]/dspace/config/*, you should then run '*ant init_configs*' in the directory *[dspace-source]/dspace/target/dspace-1.5.2-build.dir* so that any changes you may have made are reflected in the configuration files of other applications, for example Apache. You may then need to restart those applications, depending on what you changed.

6.2 The dspace.cfg Configuration Properties File

The primary way of configuring DSpace is to edit the *dspace.cfg*. You will definitely have to do this before you can run DSpace properly. *dspace.cfg* contains basic information about a DSpace installation, including system path information, network host information, and other like items. To assist you in this endeavor, below is a place for you to write down some of the preliminary data so that you may facilitate faster configuration.

- Server IP: _____
- Host Name (Server name): _____
- dspace.url: _____
- Administrator's email: _____
- handle prefix: _____
- assetstore directory: _____
- SMTP server: _____

6.2.1 The dspace.cfg file

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

Property	Ref. Sect.
Basic Information	

`dspace.dir`
`dspace.hostname`
`dspace.baseUrl`
`dspace.url`
`dspace.oai.url`
`dspace.name`
	6.3.2

Database Settings	

`db.name`
`db.url`
`db.driver`
`db.username`
`db.password`
	4.2.3 or 6.3.3

Advanced Database Configuration	

`db.schema`
`db.maxconnection`
`db.maxwait`
`db.maxidle`
`db.statementpool`
`db.poolname`
	6.3.3

Email Settings	

`mail.server`
`mail.server.username`
`mail.server.password`
`mail.server.port`
`mail.from.address`
`feedback.recipient`
`mail.admin`
`alert.recipient`
`registration.notify`
`mail.charset`
`mail.allowed.referrers`
`mail.extraproperties`
`mail.server.disabled`
	6.3.4

File Storage	

`assetstore.dir`
`[assetstore.dir.1`
`assetstore.dir.2`
`assetstore.incoming]`
	6.3.5

SRB File Storage	

`srb.hosts.1`
`srb.port.1`
`srb.mcatzone.1`
`srb.mdasdomainname.1`
`srb.defaultstorageresource.1`
`srb.username.1`
`srb.password.1`
`srb.homedirectory.1`
`srb.parentdir.1`
	6.3.6

Logging Configuration	

`log.init.config`
`log.dir`
`useProxies`
	6.3.7

Search Settings	

`search.dir`
`search.max-clauses`
`search.analyzer`
`search.operator`
`search.maxfieldlengthsearch.index.n`
`search.index.1`
	6.3.8

Handle Settings	
`handle.prefix` `handle.dir`	6.3.9
Delegation Administration : Authorization System Configuration	

``` core.authorization.community-admin.create-subelement core.authorization.community-admin.delete-subelement core.authorization.community-admin.policies core.authorization.community-admin.admin-group core.authorization.community-admin.collection.policies core.authorization.community-admin.collection.template-item core.authorization.community-admin.collection.submitters core.authorization.community-admin.collection.workflows core.authorization.community-admin.collection.admin-group core.authorization.community-admin.item.delete core.authorization.community-admin.item.withdraw core.authorization.community-admin.item.reinstatiate core.authorization.community-admin.item.policies core.authorization.community-admin.item.create-bitstream core.authorization.community-admin.item.delete-bitstream core.authorization.community-admin.item-admin.cc-license core.authorization.collection-admin.policies core.authorization.collection-admin.template-item core.authorization.collection-admin.submitters core.authorization.collection-admin.workflows core.authorization.collection-admin.admin-group core.authorization.collection-admin.item.delete core.authorization.collection-admin.item.withdraw core.authorization.collection-admin.item.reinstatiate core.authorization.collection-admin.item.policies core.authorization.collection-admin.item.create-bitstream core.authorization.collection-admin.item.delete-bitstream core.authorization.collection-admin.item-admin.cc-license core.authorization.item-admin.policies core.authorization.item-admin.create-bitstream core.authorization.item-admin.delete-bitstream core.authorization.item-admin.cc-license ```	6.3.10
Restricted Item Visibility Settings	
``` harvest.includerestricted.rss harvest.includerestricted.oai harvest.includerestricted.subscription ```	6.3.12
Proxy Settings	
``` http.proxy.host http.proxy.port ```	6.3.13

Media Filter--Format Filter Plugin Settings	

`filter.plugins`
`plugin.named.org.dspace.app.mediafilter.FormatFilter`
`filter.org.dspace.app.mediafilter.PDFFilter.inputFormats`
`filter.org.dspace.app.mediafilter.HTMLFilter.inputFormats`
`filter.org.dspace.app.mediafilter.WordFilter.inputFormats`
`filter.org.dspace.app.mediafilter.JPEGFilter.inputFormats`
`filter.org.dspace.app.mediafilter.BrandedPreviewJPEGFilter.inputFormats`
	6.3.14

Custom settings for PDFFilter	
`pdffilter.largepdfsdffilter.skiponmemoryexception`	6.3.14
Crosswalk and Packager Plugin Settings (MODS, QDC, XSLT, etc.)	

`crosswalk.mods.properties.MODS`
`crosswalk.mods.properties.mods`
`crosswalk.submission.MODS.stylesheet`
	6.3.15.1

`crosswalk.qdc.namespace.QDC.dc`
`crosswalk.qdc.namespace.QDC.dcterms`
`crosswalk.qdc.schemaLocation.QDC`
`crosswalk.qdc.properties.QDC`
`mets.submission.crosswalk.DC`
`mets.submission.preserveManifest`
`mets.submission.useCollectionTemplate`
	6.3.15

`plugin.named.org.dspace.content.crosswalk.IngestionCrosswalk`
`plugin.selfnamed.org.dspace.content.crosswalk.IngestionCrosswalk`
`plugin.named.org.dspace.content.crosswalk.DisseminationCrosswalk`
`plugin.selfnamed.org.dspace.content.crosswalk.DisseminationCrosswalk`
	6.3.15.4

`plugin.named.org.dspace.content.packager.PackageDisseminator` `plugin.named.org.dspace.content.packager.PackageIngester`	6.3.15.5
Event System Configuration	

`event.dispatcher.default.class`
`event.dispatcher.default.consumers`
`event.dispatcher.noindex.class`
`event.dispatcher.noindex.consumers`
`event.consumer.search.class`
`event.consumer.search.filters`
`event.consumer.browse.class`
`event.consumer.browse.filters`
`event.consumer.eperson.class`
`event.consumer.eperson.filters`
`event.consumer.harvester.class`
`event.consumer.harvester.filters`
`event.consumer.test.class`
`event.consumer.test.filters`
`testConsumer.verbose`
	6.3.16

Embargo Settings	

`embargo.field.terms`
`embargo.field.lift`
`embargo.field.open`
`plugin.single.org.dspace.embargo.EmbargoSetter`
`plugin.single.org.dspace.embargo.EmbargoLifter`
	6.3.17

Checksum Checker	

`plugin.single.org.dspace.checker.BitsreamDispatcher`
`checker.retention.default`
`checker.retention.CHECKSUM-MATCH`
	6.3.18

Item Export and Download Settings	

`org.dspace.app.itemexport.work.dir`
`org.dspace.app.itemexport.download.dir`
`org.dspace.app.itemexport.life.span.hours`
`org.dspace.app.itemexport.max.size`
	6.3.19

Subscription Email Option	
`eperson.subscription.onlynew`	6.3.20
Bulk (Batch) Metadata Editing	

`bulkedit.valueseparator`
`bulkedit.fieldseparator`
`bulkedit.gui-item-limit`
`bulkedit.ignore-on-export`
	6.3.21

Hide Item Metadata Fields Setting	
`metadata.hide.dc.description.provenance`	6.3.22
Submission Process	
`webui.submit.blocktheses` `webui.submit.upload.required`	6.3.23
`webui.submit.enable-cc` `webui.submit.cc-jurisdiction`	6.3.24
Settings for Thumbnail Creation	

`webui.browse.thumbnail.show`
`webui.browse.thumbnail.max.height`
`webui.browse.thumbnail.max.width`
`webui.item.thumbnail.show`
`webui.browse.thumbnail.linkbehaviour`
`thumbnail.maxwidth`
`thumbnail.maxheight`
	6.3.25

Settings for Item Preview	

`webui.preview.enabled`
`webui.preview.maxwidth`
`webui.preview.maxheight`
`webui.preview.brand`
`webui.preview.brand.abbrev`
`webui.preview.brand.height`
`webui.preview.brand.font`
`webui.preview.brank.fontpoint`
`webui.preview.dc`
	6.3.25

Settings for Content Count/Strength Information	
`webui.strengths.show` `webui.strengths.cache`	6.3.25
Browse Configuration	
`webui.browse.index.n`	6.3.26
`webui.itemlist.sort-option.n`	6.3.26
`webui.browse.medata.case-insensitive`	6.3.26.3

`webui.browse.value_columns.max`
`webui.browse.sort_columns.max`
`webui.browse.value_columns.omission_mark`
`plugin.named.org.dspace.sort.OrderFormatDelegate`
	6.3.26.4

Multiple Metadata Value Display	

`webui.browse.author-field` `webui.browse.author-limit`	6.3.27
Other Browse Contexts	
`webui.browse.link.n`	6.3.28
Recent Submission	

`recent.submission.sort-option`
`recent.submissions.count`
`plugin.sequence.org.dspace.plugin.CommunityHomeProcessor`
`plugin.sequence.org.dspace.plugin.CollectionHomeProcessor`
	6.3.29

Submission License Substitution Variables	
`plugin.named.org.dspace.content.license.LicenseArgumentFormatter`	6.3.30
Syndication Feed (RSS) Settings	

`webui.feed.enable`
`webui.feed.items`
`webui.feed.cache.size`
`webui.cache.age`
`webui.feed.formats`
`webui.feed.localresolve`
`webui.feed.item.title`
`webui.feed.item.date`
`webui.feed.item.description`
`webui.feed.item.author`
`webui.feed.item.dc.creator`
`webui.feed.item.dc.date`
`webui.feed.item.dc.description`
`webui.feed.logo.url`
	6.3.31

OpenSearch Settings	

`websvc.opensearch.enable`
`websvc.opensearch.uicontext`
`websvc.opensearch.svccontext`
`websvc.opensearch.autolink`
`websvc.opensearch.validity`
`websvc.opensearch.shortname`
`websvc.opensearch.longname`
`websvc.opensearch.description`
`websvc.opensearch.faviconurl`
`websvc.opensearch.samplequery`
`websvc.opensearch.tags`
`websvc.opensearch.formats`
	6.3.32

Content Inline Disposition Threshold	
`webui.content_disposition_threshold` `xmlui.content_disposition_threshold`	6.3.33
Multifile HTML Settings	
`webui.html.max-depth-guess` `xmlui.html.max-depth-guess`	6.3.34
Sitemap Settings	
`sitemap.dir` `sitemap.engineurls`	6.3.35
Authority Control Settings	

`plugin.named.org.dspace.content.authority.ChoiceAuthority`
`plugin.selfnamed.org.dspace.content.authority.ChoiceAuthority`
`lcname.url`
`sherpa.romeo.url`
`authority.minconfidence`
`xmlui.lookup.select.size`
	6.3.36

JSPUI Upload File Settings	
``` upload.temp.dir upload.max ```	6.3.37
JSP Web Interface Settings	
``` webui.itemdisplay.default webui.resolver.1.urn webui.resolver.1.baseurl webui.resolver.2.urn webui.resolver.2.baseurl plugin.single.org.dspace.app.webui.util.StyleSelection webui.itemdisplay.thesis.collections webui.itemdisplay.metadata-style webui.itemlist.columns webui.itemlist.widths 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 webui.itemlist.dateaccessioned.columns webui.itemlist.dateaccessioned.widths webui.itemlist.tablewidth ```	6.3.38
JSPUI i18n Locales / Languages	
`default.locale`	6.3.39
JSPUI Additional Configuration for Item Mapper	
`itemmap.author.index`	6.3.40
JSPUI MyDSpace Display of Group Membership	
`webui.mydspace.showgroupmembership`	6.3.41
JSPUI SFX Server Setting	
`sfx.server.url`	6.3.42
JSPUI Item Recommendation Settings	

`webui.suggest.enable` `webui.suggest.loggedinusers.only`	6.3.43
JSPUI Controlled Vocabulary Settings	
`webui.controlledvocabulary.enable`	6.3.44
JSPUI Session Invalidation	
`webui.session.invalidate`	6.3.45
XMLUI Settings (Manakin)	

`xmlui.supported.locales`
`xmlui.force.ssl`
`xmlui.user.registration`
`xmlui.user.editmetadata`
`xmlui.user.assumelogin`
`xmlui.user.logindirect`
`xmlui.theme.allowoverrides`
`xmlui.bundle.upload`
`xmlui.community-list.render.full`
`xmlui.community-list.cache`
`xmlui.bitstream.mods`
`xmlui.bitstream.mets`
`xmlui.google.analytics.key`
`xmlui.controlpanel.activity.max`
`xmlui.controlpanel.activity.ipheader`
	6.3.46

SOLR Statistics Configurations	

`solr.log.server`
`solr.dbfilesolr.resolver.timeout`
`statistics.item.authorization.adminsolr.statistics.logBots`
`solr.statistics.query.filter.spiderIP`
`solr.statistics.query.filter.isBot`
`solr.spiderips.urls`
	6.3.49

6.2.2 Main DSpace Configurations

Property:	`dspace.dir`

Example Value:	`/dspace`
Informational Note:	Root directory of DSpace installation. Omit the trailing '/'. Note that if you change this, there are several other parameters you will probably want to change to match, e.g. `assetstore.dir` .
Property:	`dspace.hostname`
Example Value:	`dspace.hostname = dspace.mysu.edu`
Informational Note:	Fully qualified hostname; do not include port number.
Property:	`dspace.baseUrl`
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:	`dspace.url`
Example Value:	`dspace.url = ${dspace.baseUrl}/jspui`
Informational note	DSpace base URL. URL that determines whether JSPUI or XMLUI will be loaded by default. Include port number etc., but NOT trailing slash. Change to `/xmlui` if you wish to use the xmlui (Manakin) as the default, or remove "`/jspui`" and set webapp of your choice as the "ROOT" webapp in the servlet engine.
Property:	`dspace.oai.url`
Example Value:	`dspace.oai.url = ${dspace.baseUrl}/oai`
Informational note:	The base URL of the OAI webapp (do not include /request).
Property:	`dspace.name`
Example Value:	`dspace.name = DSpace at My University`
Informational Note:	Short and sweet site name, used throughout Web UI, e-mails and elsewhere (such as OAI protocol)

6.2.3 DSpace Database Configuration

Many of the database configurations are software-dependent. That is, it will be based on the choice of database software being used. Currently, DSpace properly supports PostgreSQL and Oracle.

Property:	`db.name`
Example Value:	`db.name = postgres`
Informational Note:	Both `postgres` or `oracle` are accepted parameters.
Property:	`db.url`
Example Value:	`db.url = jdbc:postgresql://localhost:5432/dspace_-services`
Informational Note:	The above value is the default value when configuring with PostgreSQL. When using Oracle, use this value: `jbdc.oracle.thin:@//host:port/dspace`
Property:	`db.username`
Example Value:	`db.username = dspace`
Informational Note:	In the installation directions, the administrator is instructed to create the user "dspace" who will own the database "dspace".
Property:	`password`
Example Value:	`password = dspace5`
Informational Note:	This is the password that was prompted during the installation process (cf. 3.2.3. Installation)
Property:	`db.schema`
Example Value:	`db.schema = vra`
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:	`db.maxconnections`

Example Value:	`db.maxconnections = 30`
Informational Note:	Maximum number of Database connections in the connection pool
Property:	`db.maxwait`
Example Value:	`db.maxwait = 5000`
Informational Note:	Maximum time to wait before giving up if all connections in pool are busy (in milliseconds).
Property:	`db.maxidle`
Example Value:	`db.maxidle = -1`
Informational Note:	Maximum number of idle connections in pool. (*-1* = unlimited)
Property:	`db.statementpool`
Example Value:	`db.statementpool = true`
Informational Note:	Determines if prepared statement should be cached. (Default is set to true)
Property:	`db.poolname`
Example Value:	`db.poolname = dspacepool`
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'

6.2.4 DSpace Email Settings

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

Property:	`mail.server`
Example Value:	`mail.server = smtp.my.edu`

Informational Note:	The address on which your outgoing SMTP email server can be reached.
Property:	`mail.server.username`
Example Value:	`mail.server.username = myusername`
Informational Note:	SMTP mail server authentication username, if required. This property is optional.
Property:	`mail.server.password`
Example Value:	`mail.server.password = mypassword`
Informational Note:	SMTP mail server authentication password, if required. This property is optional/
Property:	`mail.server.port`
Example Value:	`mail.server.port = 25`
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:	`mail.from.address`
Example Value:	`mail.from.address = dspace-noreply@myu.edu`
Informational Note:	The "From" address for email. Change the 'myu.edu' to the site's host name.
Property:	`feedback.recipient`
Example Value:	`feedback.recipient = dspace-help@myu.edu`
Informational Note:	When a user clicks on the feedback link/feature, the information will be send to the email address of choice. This configuration is currently limited to only one recipient.
Property:	`mail.admin`
Example Value:	`mail.admin = dspace-help@myu.edu`
Informational Note:	Email address of the general site administrator (Webmaster)

Property:	`alert.recipient`
Example Value:	`alert.recipient = john.doe@myu.edu`
Informational Note:	Enter the recipient for server errors and alerts. This property is optional.
Property:	`registration.notify`
Example Value:	`registration.notify = mike.smith@myu.edu`
Informational Note:	Enter the recipient that will be notified when a new user registers on DSpace. This property is optional.
Property:	`mail.charset`
Example Value:	`mail.charset = UTF-8`
Informational Note:	Set the default mail character set. This may be over-ridden by providing a line inside the email template '*charset: <encoding>*', otherwise this default is used.
Property:	`mail.allowed.referrers`
Example Value:	`mail.allowed.referrers = localhost`
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 *dspace.hostname*. This property is optional.
Property:	`mail.extraproperties`

Example Value:	`mail.extraproperties = mail.smtp.socketFactory.port=465, \`
	`mail.smtp.socketFactory.class=javax.net.ssl.SSLSocketFactory, \`
	`mail.smtp.socketFactory.fallback=false`

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:	`mail.server.disabled`
Example Value:	`mail.server.disabled = false`

Informational Note:	An option is added to disable the mailserver. By default, this property is set to '`false`'. By setting value to '`true`', DSpace will not send out emails. It will instead log the subject of the email which should have been sent. This is especially useful for development and test environments where production data is used when testing functionality. This property is optional.
Property:	`default.language`
Example Value:	`default.language = en_US`
Informational Note:	If no other language is explicitly stated in the *input-forms.xml*, 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`

6.2.5 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 (see page 157) for details regarding SRB.

Property:	`assetstore.dir`
Example Value:	`assetstore.dir = ${dspace.dir}/assetstore`

Informational Note:	This is Asset (bitstream) store number 0 (Zero). You need not place your assetstore under the */dspace* 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: `assetstore.dir = /storevgm/assestore` .
Property:	`assetstore.dir.1` `assetstore.dir.2`
Example Value:	`assetstore.dir.1 = /second/assetstore` `assetstore.dir.2 = /third/assetstore`
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:	`assetstore.incoming`
Example Value:	`assetstore.incoming = 1`
Informational Note:	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`

6.2.6 SRB (Storage Resource Brokerage) File Storage

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

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

Property:	`srb.hosts.1`
Example value:	`srb.hosts.1 = mysrbmcathost.myu.edu`
Property:	`srb.port.1`
Example value:	`srb.port.1 = 5544`
Property:	`srb.mcatzone.1`
Example value:	`srb.mcatzone.1 = mysrbzone`
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:	`srb.mdasdomainname.1`
Example Value:	`srb.mdasdomainname.1 = mysrbdomain`
Informational Note:	Your SRB domain. This domain should be created under the same zone, specified in srb.mcatzone. Information on domains is included here http://www.sdsc.edu/srb/index.php/Zones.
Property:	`srb.defaultstorageresource.1`
Example Value:	`srb.defaultstorageresource.1 = mydefaultsrbresource`

Informational Note:	Your default SRB Storage resource.
Property:	`srb.username.1`
Example Value:	`srb.username.1 = mysrbuser`
Informational Note:	Your SRB Username.
Property:	`srb.password.1`
Example Value:	`srb.password.1 = mysrbpassword`
Informational Note:	Your SRB Password.
Property:	`srb.homedirectory.1`
Example Value:	`srb.homedirectory.1 = /mysrbzone/home/ mysrbuser.mysrbdomain`
Informational Note:	Your SRB Homedirectory
Property:	`srb.parentdir.1`
Example Value:	`srb.parentdir.1 = mysrbdspaceassetstore`
Informational Note:	Several of the terms, such as mcatzone, have meaning only in the SRB context and will be familiar to SRB users. The last, `srb.paratdir.n`, 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.

6.2.7 Logging Configuration

Property:	`log.init.config`
Example Value:	`log.init.config = ${dspace.dir}/config/log4j.properties`

Informational Note:	This is where your logging configuration file is located. You may override the default log4j configuration by providing your own. Existing alternatives are:
	`log.init.config = ${dspace.dir}/config/log4j.properties`
	`log.init.config = ${dspace.dir}/config/log4j-console.properties`

Property:	`log.dir`
Example value:	`log.dir = ${dspace.dir}/log`
Informational Note:	This is where to put the logs. (This is used for initial configuration only)
Property:	`useProxies`
Example Value:	`useProxies = true`

Informational Note:	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 *true.* This also affects IPAuthentication, and should be enabled for that to work properly if your installation uses a proxy server.

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.

6.2.8 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:	`search.dir`
Example Value:	`search.dir = ${dspace.dir}/search`
Informational Note:	Where to put the search index files
Property:	`search.max-clauses`
Example Value:	`search.max-clauses = 2048`
Informational Note:	By setting higher values of search.max-clauses will enable prefix searches to work on larger repositories.
Property:	`search.index.delay`
Example Value:	`search.index.delay = 5000`
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:	`search.analyzer`
Example Value:	`search.analyzer = org.dspace.search.DSAnalyzer`

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:	`search.analyzer`
Example Value:	`search.analyzer = org.dspace.search.DSNonStemmingAnalyzer`
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:	`search.analyzer`
Example Value:	`search.analyzer = org.apache.lucene.analysis.cn.ChineseAnalyzer`
Informational Note:	Instead of the standard English analyzer, the Chinese analyzer is used.
Property:	`search.operator`
Example Value:	`search.operator = OR`
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:	`search.maxfieldlength`
Example Value:	`search.maxfieldlength = 10000`
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:	`search.index.n`
Example Value:	`search.index.1 = author:dc.contributor.*`

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.11 = language:dc.language.iso
```

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

`<id>`	is an incremental number to distinguish each search index entry
`<search label>`	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.

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-init` to regenerate the indexes.

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

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

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 does 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.

6.2.9 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 Section 3.4.4.. The Handle Server section of Installing DSpace.

Property:	`handle.canonical.prefix`
Example Value	handle.canonical.prefix = http://hdl.handle.net/ handle.canonical.prefix = ${dspace.url}/handle/
Informational Note:	Canonical Handle URL prefix. By default, DSpace is configured to use http://hdl.handle.net/ as the canonical URL prefix when generating `dc.identifier.uri` 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. `handle.canonical.prefix = ${dspace.url}/handle/`. Note that this will not alter `dc.identifer.uri` metadata for existing items (only for subsequent submissions).
Property:	`handle.prefix`
Example Value	`handle.prefix = 1234.56789`
Informational Note:	The default installed by DSpace is `123456789` but you will replace this upon receiving a handle from CNRI.
Property:	`handle.dir`
Example Value:	`handle.dir = ${dspace.dir}/handle-server`
Informational Note:	The default files, as shown in the Example Value is where DSpace will install the files used for the Handle Server.

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

6.2.10 Delegation Administration : Authorization System Configuration

(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:	`core.authorization.community-admin.create-subelement`
Example Value:	`core.authorization.community-admin.create-subelement = true`
Informational Note:	Authorization for a delegated community administrator to create subcommunities or collections.
Property:	`core.authorization.community-admin.delete-subelement`
Example Value:	`core.authorization.community-admin.delete-subelement = true`
Informational Note:	Authorization for a delegated community administrator to delete subcommunities or collections.
Community Administration: Policies and The group of administrators	
Property:	`core.authorization.community-admin.policies`
Example Value:	`core.authorization.community-admin.policies = true`

Informational Note:	Authorization for a delegated community administrator to administrate the community policies.
Property:	`core.authorization.community-admin.admin-group`
Example Value:	`core.authorization.community-admin.admin-group = true`
Informational Note:	Authorization for a delegated community administrator to edit the group of community admins.
Community Administration: Collections in the above Community	
Property:	`core.authorization.community-admin.collection.policies`
Example Value:	`core.authorization.community-admin.collection.policies = true`
Informational Note:	Authorization for a delegated community administrator to administrate the policies for underlying collections.
Property:	`core.authorization.community-admin.collection.template-item`
Example Value:	`core.authorization.community-admin.collection.template-item = true`
Informational Note:	Authorization for a delegated community administrator to administrate the item template for underlying collections.
Property:	`core.authorization.community-admin.collection.submitters`
Example Value:	`core.authorization.community-admin.collection.submitters = true`
Informational Note:	Authorization for a delegated community administrator to administrate the group of submitters for underlying collections.
Property:	`core.authorization.community-admin.collection.workflows`
Example Value:	`core.authorization.community-admin.collection.workflows = true`
Informational Note:	Authorization for a delegated community administrator to administrate the workflows for underlying collections.
Property:	`core.authorization.community-admin.collection.admin-group`

Example Value:	`core.authorization.community-admin.collection.admin-group = true`
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:	`core.authorization.community-admin.item.delete`
Example Value:	`core.authorization.community-admin.item.delete = true`
Informational Note:	Authorization for a delegated community administrator to delete items in underlying collections.
Property:	`core.authorization.community-admin.item.withdraw`
Example Value:	`core.authorization.community-admin.item.withdraw = true`
Informational Note:	Authorization for a delegated community administrator to withdraw items in underlying collections.
Property:	`core.authorization.community-admin.item.reinstate`
Example Value:	`core.authorization.community-admin.item.reinstate = true`
Informational Note:	Authorization for a delegated community administrator to reinstate items in underlying collections.
Property:	`core.authorization.community-admin.item.policies`
Example Value:	`core.authorization.community-admin.item.policies = true`
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:	`core.authorization.community-admin.item.create-bitstream`
Example Value:	`core.authorization.community-admin.item.create-bitstream = true`

Informational Note:	Authorization for a delegated community administrator to create additional bitstreams in items in underlying collections.
Property:	`core.authorization.community-admin.item.delete-bitstream`
Example Value:	`core.authorization.community-admin.item.delete-bitstream = true`
Informational Note:	Authorization for a delegated community administrator to delete bitstreams from items in underlying collections.
Property:	`core.authorization.community-admin.item.cc-license`
Example Value:	`core.authorization.community-admin.item.cc-license = true`
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.	`core.authorization.collection-admin.policies`
	`core.authorization.collection-admin.template-item`
	`core.authorization.collection-admin.submitters`
	`core.authorization.collection-admin.workflows`
	`core.authorization.collection-admin.admin-group`

Collection Administration: Item owned by the above CollectionThe properties for collection administrators work similar to those of community administrators, with respect to administration of items in underlying collections.	`core.authorization.collection-admin.item.delete`
	`core.authorization.collection-admin.item.withdraw`
	`core.authorization.collection-admin.item.reinstatiate`
	`core.authorization.collection-admin.item.policies`

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.	`core.authorization.collection-admin.item.create-bitstream`
	`core.authorization.collection-admin.item.delete-bitstream`
	`core.authorization.collection-admin.item-admin.cc-license`

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.	`core.authorization.item-admin.policies`

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.	`core.authorization.item-admin.create-bitstream`
	`core.authorization.item-admin.delete-bitstream`
	`core.authorization.item-admin.cc-license`

Oracle users should consult *Chapter 4 Updating a DSpace Installation* regarding the necessary database changes that need to take place.

6.2.11 Restricted Item Visibility Settings

By default RSS feeds, OAI-PMH and subscription emails will include ALL items regardless of permissions set on them. If you wish to only expose items through these channels where the ANONYMOUS user is granted READ permission, then set the following options to false.

In large repositories, setting harvest.includerestricted.oai to false may cause performance problems as all items will need to have their authorization permissions checked, but because DSpace has not implemented resumption tokens in ListIdentifiers, ALL items will need checking whenever a ListIdentifers request is made.

Property:	`harvest.includerestricted.rss`
Example Value:	`harvest.includerestricted.rss = true`
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:	`harvest.includerestricted.oai`
Example Value:	`harvest.includerestricted.oai = true`
Informational Note:	When set to true (default), items that haven't got the READ permission for the ANONYMOUS user, will be included in OAI sets anyway.
Property:	`harvest.includerestricted.subscription`
Example Value:	`harvest.includerestricted.subscription = true`
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.

6.2.12 Proxy Settings

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

Property:	`http.proxy.host`
Example Value	`http.proxy.host = proxy.myu.edu`
Informational Note	Enter the host name without the port number.
Property:	`http.proxy.port`

Example Value	`http.proxy.port = 2048`
Informational Note	Enter the port number for the proxy server.

6.2.13 Configuring Media Filters

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

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

Property:	`filter.plugins`
Example Value:	`filter.plugins = PDF Text Extractor, Html Text Extractor, \` `  Word Text Extractor, JPEG Thumbnail`

Informational Note:	Place the names of the enabled MediaFilter or FormatFilter plugins. To enable Branded Preview, comment out the previous one line and then uncomment the two lines in found in *dspace.cfg*:
	`Word Text Extractor, JPEG Thumbnail,\`
	`  Branded Preview JPEG`

Property:	`plugin.named.org.dspace.app.mediafilter.FormatFilter`

Example Value:	`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`

Informational Note:	Assign "human-understandable" names to each filter

Property:	`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`

Example Value:	`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`

Informational Note:	Configure each filter's input format(s)
Property:	`pdffilter.largepdfs`
Example Value:	`pdffilter.largepdfs = true`
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:	`pdffilter.skiponmemoryexception`
Example Value:	`pdffilter.skiponmemoryexception = true`
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 PDFilter 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` .

6.2.14 Crosswalk and Packager Plugin Settings

The subsections below give configuration details based on the types of crosswalks and packager plugins you need to implement.

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:	`crosswalk.mods.properties.MODS` `crosswalk.mods.properties.mods`
Example Values:	`crosswalk.mods.properties.MODS = crosswalks/mods.properties` `crosswalk.mods.properties.mods = crosswalks/mods.properties`
Informational Note:	This defines a crosswalk named MODS whose configuration comes from the file `[dspace]/config/crosswalks/mods.properties` . (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:	`crosswalk.submission.MODS.stylesheet`
Example Value:	`crosswalk.submission.MODS.stylesheet = crosswalks/mods-submission.xsl`
Informational Note:	Configuration XSLT-driven submission crosswalk for MODS

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

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

`crosswalk` first part of the property key.
`submission` second part of the property key.
`PluginName` is the name of the plugin. The *path* value is the path to the file containing the crosswalk stylesheet (relative to `/[dspace]/config`).
Here is an example that configures a crosswalk named "LOM" using a stylesheet in `[dspace]/config/crosswalks/d-lom.xsl`:
`crosswalk.submission.LOM.stylesheet = crosswalks/d-lom.xsl`
A dissemination crosswalk can be configured by starting with the property key *crosswalk.dissemination*.
Example:
`crosswalk.dissemination.PluginName.stylesheet = path`
The *PluginName* is the name of the plugin (!) . The *path* value is the path to the file containing the crosswalk stylesheet (relative to `/[dspace]/config`).

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

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

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

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

For example:

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

Testing XSLT Crosswalks

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

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

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

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

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

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:	`crosswalk.qdc.namspace.qdc.dc`
Example Value:	`crosswalk.qdc.namspace.qdc.dc = http://purl.org/dc/elements/1.1_`
Properties:	`crosswalk.qdc.namspace.qdc.dcterms`
Example Value:	`crosswalk.qdc.namspace.qdc.dc = http://purl.org/dc/terms/_`
Properties:	`crosswalk.qdc.schemaLocation.QDC`

Example Value:	`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`

Properties:	`crosswalk.qdc.properties.QDC`
Example Value:	`crosswalk.qdc.properties.QDC = crosswalks/QDC.properties`
Informational Note:	Configuration of the QDC Crosswalk dissemination plugin for Qualified DC. *(Add lower-case name for OAI-PMH. That is, change QDC to qdc.)*}}

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

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

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

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

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

For example, in this property:

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

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

```
<dcterms:temporal>Fall, 2005</dcterms:temporal>
```

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.

6.2.15 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:	`event.dispatcher.default.class`
Example Value:	`event.dispatcher.default.class = org.dspace.event.BasicDispatcher`
Informational Note:	This is the default synchronous dispatcher (Same behavior as traditional DSpace).
Property:	`event.dispatcher.default.consumers`
Example Value:	`event.dispatcher.default.consumers = search, browse, eperson`
Informational Note:	This is the default synchronous dispatcher (Same behavior as traditional DSpace).
Property:	`event.dispatcher.noindex.class`
Example Value:	`event.dispatcher.noindex.class = org.dspace.event.BasicDispatcher`
Informational Note:	The noindex dispatcher will not create search or browse indexes (useful for batch item imports).
Property:	`event.dispatcher.noindex.consumers`
Example Value:	`event.dispatcher.noindex.consumers = eperson`
Informational Note:	The noindex dispatcher will not create search or browse indexes (useful for batch item imports).
Property:	`event.consumer.search.class`
Example Value:	`event.consumer.search.class = org.dspace.search.SearchConsumer`
Informational Note:	Consumer to maintain the search index.
Property:	`event.consumer.search.filters`

Example Value:	{{event.consumer.search.filters = }} `Community | Collection | Item | Bundle+Add | Create | Modify | Modify_Metadata | Delete | Remove`
Informational Note:	Consumer to maintain the search index.
Property:	`event.consumer.browse.class`
Example Value:	`event.consumer.browse.class = org.dspace.browse.BrowseConsumer`
Informational Note:	Consumer to maintain the browse index.
Property:	`event.consumer.browse.filters`
Example Value:	`event.consumer.browse.filters =` `Community | Collection | Item | Bundle+Add | Create | Modify | Modify_Metadata | Delete | Remove`
Informational Note:	Consumer to maintain the browse index.
Property:	`event.consumer.eperson.class`
Example Value:	`event.consumer.eperson.class = org.dspace.eperson.EPersonConsumer`
Informational Note:	Consumer related to EPerson changes
Property:	`event.consumer.eperson.filters`
Example Value:	`event.consumer.eperson.filters = EPerson+Create`
Informational Note:	Consumer related to EPerson changes
Property:	`event.consumer.test.class`
Example Value:	`event.consumer.test.class = org.dspace.event.TestConsumer`
Informational Note:	Test consumer for debugging and monitoring. **Commented out by default.**
Property:	`event.consumer.test.filters`

Example Value:	`event.consumer.test.filters = All+All`
Informational Note:	Test consumer for debugging and monitoring. **Commented out by default.**
Property:	`testConsumer.verbose`
Example Value:	`testConsumer.verbose = true`
Informational Note:	Set this to true to enable testConsumer messages to standard output. **Commented out by default.**

6.2.16 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:	`embargo.field.terms`
Example Value:	`embargo.field.terms = SCHEMA.ELEMENT.QUALIFIER`
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 dc.embargo.terms
Property:	`embargo.field.lift`
Example Value:	`embargo.field.lift = SCHEMA.ELEMENT.QUALIFIER`
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 dc.embargo.liftdate
Property:	`embargo.terms.open`
Example Value:	`embargo.terms.open = forever`

Informational Note:	You can determine your own values for the embargo.field.terms 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:	`plugin.single.org.dspace.embargo.EmbargoSetter`
Example Value:	`plugin.single.org.dspace.embargo.EmbargoSetter = org.dspace.embargo.DefaultEmbargoSetter`
Informational Note:	To implement the business logic to set your embargos, you need to override the EmbargoSetter class. If you use the value DefaultEmbargoSetter, the default implementation will be used.
Property:	`plugin.single.org.dspace.embargo.EmbargoLifter`
Example Value:	`plugin.single.org.dspace.embargo.EmbargoLifter = org.dspace.embargo.DefaultEmbargoLifter`
Informational Note:	To implement the business logic to lift your embargos, you need to override the EmbargoLifter class. If you use the value DefaultEmbargoLifter, the default implementation will be used.

Key Recommendations:

1. 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.
2. 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.
3. 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 two distinct fields instead.
   . Detailed 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 Section 8.

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 aspects.

1. 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.
2. 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
   ```

Step-by-Step Setup Examples

1. Simple Dates.If you want to enter simple calendar dates for when an embargo will expire, follow these steps.
    1. Select a metadata field. Let's use *dc.description.embargo*. This field does not exist in in the default DSpace metadata directory, so login as an administrator, go the metadata registry page, select the 'dc' schema, then add the metadata field.
    2. Expose the metadata field. Edit *[dspace]/config/input-forms.xml* . If you have only one form, usually 'traditional', add it there. If you have multiple forms, add it only to the forms linked to collections for which embargo applies:

```
<form name="traditional">
    <page number="1">
      ...
       <field>
         <dc-schema>dc</dc-schema>
         <dc-element>description</dc-element>
         <dc-qualifier>embargo</dc-qualifier>
         <repeatable>false</repeatable>
         <label>Embargo Date</label>
         <input-type>onebox</input-type>
         <hint>If required, enter date 'yyyy-mm-dd' when embargo expires or
'forever'.</hint>
         <required></required>
       </field>
```

    Note: if you want to require embargo terms for every item, put a phrase in the <required> element. Example:*<required>You must enter an embargo date</required>*
    3. Configure Embargo. Edit *[dspace]/config/dspace.cfg*. Find the Embargo properties and set these two:

```
# DC metadata field to hold the user-supplied embargo terms
embargo.field.terms = dc.description.embargo

# DC metadata field to hold computed "lift date" of embargo
embargo.field.lift = dc.description.embargo
```

    4. Restart DSpace application. This will pick up these changes. Now just enter future dates (if applicable) in web submission and the items will be placed under embargo. You can enter years ('2020'), years and months ('2020-12'), or also days ('2020-12-15').
    5. Periodically run the lifter. Run the task:_[dspace]/bin/dspace embargo-lifter_You will want to run this task in a cron-scheduled or other repeating way. Item embargoes will be lifted as their dates pass.
2. Period Sets. If you wish to use a fixed set of time periods (e.g. 90 days, 6 months and 1 year) as embargo terms, follow these steps, which involve using a custom 'setter'.

1. Select two metadata fields. Let's use '*dc.embargo.terms*' and '*dc.embargo.lift*'. These fields do not exist in the default DSpace metadata registry. Login as an administrator, go the metadata registry page, select the 'dc' schema, then add the metadata fields.
2. Expose the 'term' metadata field. The lift field will be assigned by the embargo system, so it should not be exposed directly. Edit *[dspace]/config/input-forms.xml* . If you have only one form (usually 'traditional') add it there. If you have multiple forms, add it only to the form(s) linked to collection(s) for which embargo applies. First, add the new field to the 'form definition':

```
<form name="traditional">
    <page number="1">
      ...
       <field>
         <dc-schema>dc</dc-schema>
         <dc-element>embargo</dc-element>
         <dc-qualifier>terms</dc-qualifier>
         <repeatable>false</repeatable>
         <label>Embargo Terms</label>
         <input-type value-pairs-name="embargo_terms">dropdown</input-type>
         <hint>If required, select embargo terms.</hint>
         <required></required>
         </field>
```

Note: If you want to require embargo terms for every item, put a phrase in the *<required>* element, e.g._<required>You must select embargo terms</required>_Observe that we have referenced a new value-pair list: "embargo_terms'. We must now define that as well (only once even if references by multiple forms):

```
<form-value-pairs>
   ...
   <value-pairs value-pairs-name="embargo_terms" dc-term="embargo.terms">
     <pair>
       <displayed-value>90 days</displayed-value>
       <stored-value>90 days</stored-value>
     </pair>
     <pair>
       <displayed-value>6 months</displayed-value>
       <stored-value>6 months</stored-value>
     </pair>
     <pair>
       <displayed-value>1 year</displayed-value>
       <stored-value>1 year</stored-value>
     </pair>
   </value-pairs>
```

Note: if desired, you could localize the language of the displayed value.

3. Configure Embargo. Edit /dspace/config/dspace.cfg. Find the Embargo properties and set the following properties:

```
# DC metadata field to hold the user-supplied embargo terms
    embargo.field.terms = dc.embargo.terms

    # DC metadata field to hold computed "lift date" of embargo
    embargo.field.lift = dc.embargo.lift

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

Now add a new property called '*embargo.terms.days*' as follows:

```
# DC metadata field to hold computed "lift date" of embargo
    embargo.terms.days = 90 days:90, 6 months:180, 1 year:365
```

1. 1. This step is the same as Step A.4 above, except that instead of entering a date, the submitter will select a value form a drop-down list.

1. 1. Periodically run the lifter. Run the task:
   `[dspace]/bin/dspace embargo-lifter` .
   You will want to run this task in a cron-scheduled or other repeating way. Item embargoes will be lifted as their dates pass.

6.2.17 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:	`plugin.single.org.dspace.checker.BitstreamDispatcher`
Example Value:	`plugin.single.org.dspace.checker.BitstreamDispatcher = org.dspace.checker.SimpleDispatcher`

Informational Note:	The Default dispatcher is case non is specified.
Property:	`checker.retention.default`
Example Value:	`checker.retention.default = 10y`
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:	`checker.retention.CHECKSUM_MATCH`
Example Value:	`checker.retention.CHECKSUM_MATCH = 8w`
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).

6.2.18 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:	`org.dspace.app.itemexport.work.dir`
Example Value:	`org.dspace.app.itemexport.work.dir = ${dspace.dir}/exports`
Informational Note:	The directory where the exports will be done and compressed.
Property:	`org.dspace.app.itemexport.download.dir`
Example Value:	`org.dspace.app.itemexport.download.dir = ${dspace.dir}/exports/download`

Informational Note	The directory where the compressed files will reside and be read by the downloader.
Property:	`org.dspace.app.itemexport.life.span.hours`
Example Value:	`org.dspace.app.itemexport.life.span.hours = 48`
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:	`org.dspace.app.itemexport.max.size`
Example Value:	`org.dspace.app.itemexport.max.size = 200`
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.

6.2.19 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:	`eperson.subscription.onlynew`
Example Value:	`eperson.subscription.onlynew = true`
Informational Note:	For backwards compatibility, the subscription emails by default include any modified items. The property key is COMMENTED OUT by default.

6.2.20 Hiding Metadata

It is now possible to hide metadata from public consumption that is only available to the Administrator.

Property:	`metadata.hide.dc.description.provenance`
Example Value:	`metadata.hide.dc.description.provenance = true`

Informational Note:	Hides the metadata in the property key above except to the administrator. Fields named here are hidden in the following places UNLESS the logged-in user is an Administrator: 1. XMLUI metadata XML view, and Item splash pages (long and short views). 2. JSPUI Item splash pages 3. OAI-PMH server, "oai_dc" format. (Note: Other formats are ***not*** affected.)To designate a field as hidden, add a property here in the form: `metadata.hide.SCHEMA.ELEMENT.QUALIFIER = true`. This default configuration hides the `dc.description.provenance` field, since that usually contains email addresses which ought to be kept private and is mainly of interest to administrators.

6.2.21 Settings for the Submission Process

These settings control two aspects of the submission process: thesis submission permission and whether or not a bitstream file is required when submitting to a collection.

Property:	`webui.submit.blocktheses`
Example Value:	`webui.submit.blocktheses = false`
Informational Note:	Controls whether or not that the submission should be marked as a thesis.
Property:	`webui.submit.upload.required`
Example Value:	`webui.submit.upload.required = true`
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.

6.2.22 Configuring Creative Commons License

This enables the Creative Commons license step in the submission process of either the JSP or XML User Interface (JSP UI or XML UI). Submitters are given an opportunity to select a Creative Common 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 enabled as one step of the configurable submission process, and therefore 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 'Customizing and Configuring 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. For the JSP UI, Creative Commons licensing is effected by opening an Iframe to the Creative Commons site and capturing the selection result in several bitstreams, but the XML UI utilizes a more flexible web service. By default, when a license is selected in the interface, the URI for the license is stored in the 'dc.rights.uri' metadata field for the Item, and a representation of the license text is stored in a license bundle. In addition, the following properties in `[dspace]/config/dspace.cfg` may be customized for use:

Property:	`cc.api.rooturl`
Example Value:	`cc.api.rooturl = http://api.creativecommons.org/rest/1.5`
Informational Note:	`Generally will never have to assign a different value - this is the base URL of the Creative Commons service API.`
Property:	`cc.license.uri`
Example Value:	`cc.license.uri = dc.rights.uri`
Informational Note:	`The field that holds the Creative Commons license URI. If you change from the default value (dc.rights.uri), you will have to reconfigure the XMLUI for proper display of license data`
Property:	`cc.license.name`
Example Value:	`cc.license.name = dc.rights`
Informational Note:	`The field that holds the Creative Commons license Name. If you change from the default value (dc.rights), you will have to reconfigure the XMLUI for proper display of license data`
Property:	`cc.submit.setname`
Example Value:	`cc.submit.setname = true`

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:	cc.submit.addbitstream
Example Value:	cc.submit.addbitstream = true
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:	cc.license.classfilter
Example Value:	cc.license.classfilter = recombo,mark
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:	cc.license.jurisdiction
Example Value:	cc.license.jurisdiction = nz
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)

6.2.23 WEB User Interface Configurations

General Web User Interface Configurations
In this section of Configuration, we address the agnostic WEB User Interface that is used for JSP UI and XML UI. Some of the configurations will give information towards customization or refer you to the appropriate documentation.

Property:	webui.licence_bundle.show
Example Value:	webui.licence_bundle.show = false
Informational Note:	Sets whether to display the contents of the license bundle (often just the deposit license in the standard DSpace installation).
Property:	webui.browse.thubnail.show

Example Value:	`webui.browse.thubnail.show = true`
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:	`webui.browse.thumbnail.maxheight`
Example Value:	`webui.browse.thumbnail.maxheight = 80`
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:	`webui.browse.thumbnail.maxwidth`
Example Value:	`webui.browse.thumbnail.maxwidth = 80`
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:	`webui.item.thumbnail.show`
Example Value:	`webui.item.thumbnail.show = true`
Informational Note:	This determines whether or not to display the thumbnail against each bitstream. *(This configuration property key is not used by XMLUI. To show thumbnails using XMLUI, you need to create a theme which displays them).*
Property:	`webui.browse.thumbnail.linkbehavior`
Example Value:	`webui.browse.thumbnail.linkbehavior = item`
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:	`thumbnail.maxwidth`
Example Value:	`thumbnail.maxwidth = 80`

Informational Note:	This property sets the maximum width of generated thumbnails that are being displayed on item pages.
Property:	`thumbnail.maxheight`
Example Value:	`thumbnail.maxheight = 80`
Informational Note:	This property sets the maximum height of generated thumbnails that are being displayed on item pages.
Property:	`webui.preview.enabled`
Example Value:	`webui.preview.enabled = false`
Informational Note:	Whether or not the user can "preview" the image.
Property:	`webui.preview.maxwidth`
Example Value:	`webui.preview.maxwidth = 600`
Informational Note:	This property sets the maximum width for the preview image.
Property:	`webui.preview.maxheight`
Example Value:	`webui.preview.maxheight = 600`
Informational Note:	This property sets the maximum height for the preview image.
Property:	`webui.preview.brand`
Example Value:	`webui.preview.brand = My Institution Name`
Informational Note:	This is the brand text that will appear with the image.
Property:	`webui.preview.brand.abbrev`
Example Value:	`webui.preview.brand.abbrev = MyOrg`
Informational Note:	An abbreviated form of the full Branded Name. This will be used when the preview image cannot fit the normal text.

Property:	`webui.preview.brand.height`
Example Value:	`webui.preview.brand.height = 20`
Informational Note:	The height (in px) of the brand.
Property:	`webui.preview.brand.font`
Example Value:	`webui.preview.brand.font = SansSerif`
Informational Note:	This property sets the font for your Brand text that appears with the image.
Property:	`webui.preview.brand.fontpoint`
Example Value:	`webui.preview.brand.fontpoint = 12`
Informational Note:	This property sets the font point (size) for your Brand text that appears with the image.
Property:	`webui.preview.dc`
Example Value:	`webui.preview.dc = rights`
Informational Note:	The Dublin Core field that will display along with the preview. This field is optional.
Property:	`webui.strengths.show`
Example Value:	`webui.strengths.show = false`
Informational Note:	Determines if communities and collections should display item counts when listed. The default behavior if omitted, is true. *(This configuration property key is not used by XMLUI. To show thumbnails using XMLUI, you need to create a theme which displays them).*
Property:	`webui.strengths.cache`
Example Value:	`webui.strengths.cache = false`

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: `/[dspace]/bin/dspace itemcounter`. The default is to count in real time (set to "false").

6.2.24 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:	`webui.browse.index.<n>`
Example Value:	{{webui.browse.index.1 = dateissued:metadata:dc.date.issued:date:full }}
Informational Note:	This is an example of how one "Defines the Indexes". See Defining the Indexes in the next sub-section.
Property:	`webui.itemlist.sort-option.<n>`
Example Value:	`webui.itemlist.sort-option.1 = title:dc.title:title`
Informational Note:	This is an example of how one "Defines the Sort Options". See Defining Sort Options in the following sub-section.

Defining the Indexes.

DSpace arrives with four default indexes already defined: author, title, date issued, and subjects. Users may also define additional indexes or re-configure the current indexes for different levels of specificity. For example, the default entries that appear in the *dspace.cfg* as default installation:

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

The format of each entry is `webui.browse.index.<n> = <index name>:<metadata>:<schema prefix>.<element>.<qualifier>:<data-type field>:<sort option>`. Please notice that the punctuation is paramount in typing this property key in the `dspace.cfg` file. The following table explains each element:

Element	Definition and Options (if available)
`webui.browse.index.{<n>`	*n* 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).
`<index name>`	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: `browse.type.metadata.<index name>` .
`<metadata>`	Only two options are available: "`metadata`" or "`item`"
`<schema prefix>`	The schema used for the field to be index. The default is dc (for Dublin Core).
`<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 `webui.itemlist.columns` ; "single" will be a single list of only the indexed term.

If you are customizing this list beyond the default, you will need to insert the text you wish to appear in the navigation and on link and buttons. You need to edit the `Messages.properties` file. The form of the parameter(s) in the file:
`browse.type.<index name>`

Defining Sort Options

Sort options will be available when browsing a list of items (i.e. only in "full" mode, not "single" mode). You can define an arbitrary number of fields to sort on, irrespective of which fields you display using *web.itemlist.columns* . For example, the default entries that appear in the *dspace.cfg* as default installation:

```
webui.itemlist.sort-option.1 = title:dc.title:title
webui.itemlist.sort-option.2 = dateissued:dc.date.issued:date
webui.itemlist.sort-option.3 = dateaccessioned:dc.date.accessioned:date
```

The format of each entry is `web.browse.sort-option.<n> = <option name>:<schema prefix>.<element>.<qualifier>:<datatype>`. Please notice the punctuation used between the different elements. The following table explains the each element:

Element	Definition and Options (if available)
`webui.browse.index.n`	*n* is an arbitrary number you choose.
`<option name>`	The name by which the sort option will be identified. This may be used in later configuration or to locate the message key (found in *Messages.properties* file) for this index.
`<schema prefix>`	The schema used for the field to be index. The default is dc (for Dublin Core).
`<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.
`<datatype field>`	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

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:	`webui.browse.metadata.case-insensitive`
Example Value:	`webui.browse.metadata.case-insensitive = true`
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:	`webui.browse.value_columns.max`
Example Value:	`webui.browse.value_columns.max = 500`
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:	`webui.browse.sort_columns.max`
Example Value:	`webui.browse.sort_columns.max = 200`
Informational Note:	Size of field for hidden sort columns (this will affect only sorting, not display). Commented out as default.
Property:	`webui.browse.value_columns.omission_mark`

Example Value:	`webui.browse.value_columns.omission_mark = ...`
Informational Note:	Omission mark to be placed after truncated strings in display. The default is "...".
Property:	`plugin.named.org.dspace.sort.OrderFormatDelegate`
Example Value:	`plugin.named.org.dspace.sort.OrderFormatDelegate = \` `org.dspace.sort.OrderFormatTitleMarc21=title`

Informational Note:	This sets the option for how the indexes are sorted. All sort normalizations are carried out by the OrderFormatDelegate. The plugin manager can be used to specify your own delegates for each datatype. The default datatypes (and delegates) are:
	`author = org.dspace.sort.OrderFormatAuthor`
	`title = org.dspace.sort.OrderFormatTitle`
	`text = org.dspace.sort.OrderFormatText`
	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 *dspace.cfg* file.

Browse Index Authority Control Configuration

Property:	`webui.browse.index.<n>`
Example Value:	`webui.browse.index.5 =` `lcAuthor:metadataAuthority:dc.contributor.author:authority`
Informational Note:	

6.2.25 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:	`webui.browse.author-field`

Example Value:	`webui.browse.author-field = dc.contributor.*`
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:	`webui.browse.author-limit`
Example Value:	`webui.browse.author-limit = <n>`
Informational Note: | Where <n> is an integer number of values to be displayed. Use `-1` for unlimited (the default value).	

6.2.26 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:	`webui.browse.link.n`
Example Value:	`webui.browse.link.1 = author:dc.contributor.*`
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 (`webui.browse.index.n`) with a metadata field listed in `webui.itemlist.columns` above. If this condition is not fulfilled, cross-linking will not work. Note also that crosslinking only works for metadata fields not tagged as `title` in `webui.itemlist.columns`.

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)
`webui.browse.link.n`	{{*n* is an arbitrary number you choose
`<index name>`	This need to match your entry for the index name from *webui.browse.index* property key.
`<display column metadata>`	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.

6.2.27 Recent Submissions

This allows us to define which index to base Recent Submission display on, and how many we should show at any one time. This uses the PluginManager to automatically load the relevant plugin for the Community and Collection home pages. Values given in examples are the defaults supplied in *dspace.cfg*

Property:	`recent.submission.sort-option`
Example Value:	`recent.submission.sort-option = dateaccessioned`
Informational Note:	First is to define the sort name (from *webui.browse.sort-options*) to use for displaying recent submissions.
Property:	`recent.submissions.count`
Example Value:	`recent.submissions.count = 5`
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
```

6.2.28 Submission License Substitution Variables

Property:	`plugin.named.org.dspace.content.license.`
	`LicenseArgumentFormatter`
	
	(property key broken up for display purposes only)

Example Value:	`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`

Informational Note:	It is possible include contextual information in the submission license using substitution variables. The text substitution is driven by a plugin implementation.

6.2.29 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:	`webui.feed.enable`
Example Value:	`webui.feed.enable = true`
Informational Note:	By default, RSS feeds are set to true (on) . Change key to "false" to disable.
Property:	`webui.feed.items`

Example Value:	`webui.feed.items = 4`
Informational Note:	Defines the number of DSpace items per feed (the most recent submissions)
Property:	`webui.feed.cache.size`
Example Value:	`webui.feed.cache.size = 100`
Informational Note:	Defines the maximum number of feeds in memory cache. Value of "`0`" will disable caching.
Property:	`webui.feed.cache.age`
Example Value:	`webui.feed.cache.age = 48`
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:	`webui.feed.formats`
Example Value:	`webui.feed.formats = rss_1.0,rss_2.0,atom_1.0`
Informational Note:	Defines which syndication formats to offer. You can use more than one; use a comma-separated list. The following list are the available values: rss_0.90, rss_0.91, rss_0.92, rss_0.93, rss_0.94, rss_1.0, rss_2.0, atom_1.0.
Property:	`webui.feed.localresolve`
Example Value:	`webui.feed.localresolve = false`
Informational Note:	By default, (set to false), URLs returned by the feed will point at the global handle resolver (e.g. http://hdl.handle.net/123456789/1). If set to ***true*** the local server URLs are used (e.g. http://myserver.myorg/handle/123456789/1).
Property:	`webui.feed.item.title`
Example Value:	`webui.feed.item.title = dc.title`
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 <scheme prefix>.<element>.<qualifier> 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:	`webui.feed.item.date`
Example Value:	`webui.feed.item.date = dc.date.issued`
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 <scheme prefix>.<element>.<qualifier> 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:	`webui.feed.item.description`

Example Value:	`webui.feed.item.description = dc.title, dc.contributor.author, \`
	`dc.contributor.editor, dc.description.abstract, \`
	`dc.description`

Informational Note:	One can customize the metadata fields to show in the feed for each item's description. Elements are displayed in the order they are specified in *dspace.cfg*.Like other property keys, the format of this property key is: *webui.feed.item.description* = <*scheme prefix*>.<*element*>.<*qualifier*>. 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:	`webui.feed.item.author`
Example Value:	`webui.feed.item.author = dc.contributor.author`
Informational Note:	The name of field to use for authors (Atom only); repeatable.
Property:	`webui.feed.logo.url`
Example Value:	`webui.feed.logo.url =` `${dspace.url}/themes/mysite/images/mysite-logo.png`
Informational Note:	Customize the image icon included with the site-wide feeds. This must be an absolute URL.
Property:	`webui.feed.item.dc.creator`
Example Value:	`webui.feed.item.dc.creator = dc.contributor.author`

Informational Note:	This optional property adds *structured* DC elements as XML elements to the feed description. They are not the same thing as, for example, *webui.feed.item.description*. Useful when a program or stylesheet will be transforming a feed and wants separate author, description, date, etc.
Property:	`webui.feed.item.dc.date`
Example Value:	`webui.feed.item.dc.date = dc.date.issued`
Informational Note:	This optional property adds *structured* DC elements as XML elements to the feed description. They are not the same thing as, for example, *webui.feed.item.description*. Useful when a program or stylesheet will be transforming a feed and wants separate author, description, date, etc.
Property:	`webui.feed.item.dc.description`
Example Value:	`webui.feed.item.dc.description = dc.description.abstract`
Informational Note:	This optional property adds *structured* DC elements as XML elements to the feed description. They are not the same thing as, for example, *webui.feed.item.description*. Useful when a program or stylesheet will be transforming a feed and wants separate author, description, date, etc.
Property:	`webui.feed.podcast.collections`
Example Value:	`webui.feed.podcast.collections = 1811/45183,1811/47223`
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:	`webui.feed.podcast.communities`
Example Value:	`webui.feed.podcast.communities = 1811/47223`
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:	`webui.feed.podcast.mimetypes`

Example Value:	`webui.feed.podcast.mimetypes = audio/x-mpeg,application/pdf`
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:	`webui.feed.podcast.sourceuri`
Example Value:	`webui.feed.podcast.sourceuri = dc.source.uri`
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

6.2.30 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:	`websvc.opensearch.enable`
Example Value:	`websvc.opensearch.enable = false`
Informational Note:	Whether or not OpenSearch is enabled. By default, the feature is disabled. Change the property key to 'true' to enable.
Property:	`websvc.opensearch.uicontext`
Example Value:	`websvc.opensearch.uicontext = simple-search`

Informational Note:	`Context for HTML request URLs. Change only for non-standard servlet mapping.`
Property:	`websvc.opensearch.svccontext`
Example Value:	`websvc.opensearch.svccontext = open-search/`
Informational Note:	Context for RSS/Atom request URLs. Change only for non-standard servlet mapping.
Property:	`websvc.opensearch.autolink`
Example Value:	`websvc.opensearch.autolink = true`
Informational Note:	Present autodiscovery link in every page head.
Property:	`websvc.opensearch.validity`
Example Value:	`websvc.opensearch.validity = 48`
Informational Note:	Number of hours to retain results before recalculating. This applies to the Manakin interface only.
Property:	`websvc.opensearch.shortname`
Example Value:	`websvc.opensearch.shortname = DSpace`
Informational Note:	A short name used in browsers for search service. It should be sixteen (16) or fewer characters.
Property:	`websvc.opensearch.longname`
Example Value:	`websvc.opensearch.longname = ${dspace.name}`
Informational Note:	A longer name up to 48 characters.
Property:	`websvc.opensearch.description`
Example Value:	`websvc.opensearch.description = ${dspace.name} DSpace repository`
Informational Note:	`Brief service description`

Property:	`websvc.opensearch.faviconurl`
Example Value:	_websvc.opensearch.faviconurl = http://www.dspace.org/images/favicon.ico_
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:	`websvc.opensearch.samplequery`
Example Value:	`websvc.opensearch.samplequery = photosynthesis`
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:	`websvc.opensearch.tags`
Example Value:	`websc.opensearch.tags = IR DSpace`
Informational Note:	Tags used to describe search service.
Property:	`websvc.opensearch.formats`
Example Value:	`websvc.opensearch.formats = html,atom,rss`
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.

6.2.31 Content Inline Disposition Threshold

The following configuration is used to change the disposition behavior of the browser. That is, when the browser will attempt to open the file or download it to the user-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:	`webui.content_disposition_threshold`
Example value:	`webui.content_disposition_threshold = 8388608`
Informational Note:	The default value is set to 8MB. This property key applies to the JSPUI interface.

Property:	`xmlui.content_disposition_threshold`
Example Value:	`xmlui.content_disposition_threshold = 8388608`
Informational Note:	The default value is set to 8MB. This property key applies to the XMLUI (Manakin) interface.

Other values are possible:
4 MB = 41943048 MB = 838860816 MB = 16777216

6.2.32 Multi-file HTML Document/Site Settings

The setting is used to configure the "depth" of request for html documents bearing the same name.

Property:	`webui.html.max-depth-guess`
Example Value:	`webui.html.max-depth-guess = 3`
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 "*foo/bar/index.html*" and one has a bitstream called just "*index.html*", DSpace will serve up the former bitstream (*foo/bar/index.html*) for the request if *webui.html.max-depth-guess* is 2 or greater. If *webui.html.max-depth-guess* is 1 or less, then DSpace 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 is set to 3.
Property:	`xmlui.html.max-depth-guess`
Example Value:	`xmlui.html.max-depth-guess = 3`
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 "*foo/bar/index.html*" and one has a bitstream called just "*index.html*", DSpace will serve up the former bitstream (*foo/bar/index.html*) for the request if *webui.html.max-depth-guess* is 2 or greater. If *xmlui.html.max-depth-guess* is 1 or less, then DSpace 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 is set to 3.

6.2.33 Sitemap Settings

To aid web crawlers index the content within your repository, you can make use of sitemaps.

Property:	`sitemap.dir`
Example Value:	`sitemap.dir = ${dspace.dir}/sitemaps`
Informational Note:	The directory where the generate sitemaps are stored.
Property:	`sitemap.engineurls`
Example Value:	_sitemap.engineurls = http://www.google.com/webmasters/sitemaps/ping?sitemap=_
Informational Note:	Comma-separated list of search engine URLs to 'ping' when a new Sitemap has been created. In everything except the Sitemap UL itself (which will be URL-encoded and appended to form the ac 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_ME1 . (Replace the component _REPLACE_ME* with your application ID). There is no known 'ping' UR MSN/Live search.

6.2.34 Authority Control Settings

Two new features of DSpace 1.6 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:
http://wiki.dspace.org/index.php/Authority_Control_of_Metadata_Values

Property:	`plugin.named.org.dspace.content.authority.ChoiceAuthority`

Example Value:	`plugin.named.org.dspace.content.authority.ChoiceAuthority = \`
	`org.dspace.content.authority.SampleAuthority = Sample, \`
	`org.dspace.content.authority.LCNameAuthority = LCNameAuthority, \`
	`org.dspace.content.authority.SHERPARoMEOPublisher = SRPublisher, \`
	`org.dspace.content.authority.SHERPARoMEOJournalTitle = SRJournalTitle`

Informational Note:	`--`
Property:	`plugin.selfnamed.org.dspace.content.authority.ChoiceAuthority`

Example Value:	`plugin.selfnamed.org.dspace.content.authority.ChoiceAuthority = \ org.dspace.content.authority.DCInputAuthority`
Property:	`lcname.url`
Example Value:	`lcname.url = http://alcme.oclc.org/srw/search/lcnaf_`
Informational Note:	Location (URL) of the Library of Congress Name Service
Property:	`sherpa.romeo.url`
Example Value:	`sherpa.romeo.url = http://www.sherpa.ac.uk/romeo/api24.php_`
Informational Note:	Location (URL) of the SHERPA/RoMEO authority plugin
Property:	`authority.minconfidence`
Example Value:	`authority.minconfidence = ambiguous`
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 `org.dspace.content.authority.Choices` source for descriptions.
Property:	`xmlui.lookup.select.size`
Example Value:	`xmlui.lookup.select.size = 12`
Informational Note:	This property sets the number of selectable choices in the Choices lookup popup

6.2.35 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:	`upload.temp.dir`

Example Value:	`upload.temp.dir = ${dspace.dir}/upload`
Informational Note:	This property sets where DSpace temporarily stores uploaded files.
Property:	`upload.max`
Example Value:	`upload.max = 536870912`
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.

6.2.36 JSP Web Interface (JSPUI) Settings

The following section is limited to JSPUI. If the user wishes to use XMLUI settings, please refer to Chapter 7: XMLUI Configuration and Customization.

Property:	`webui.itemdisplay.default`

Example Value:	`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`

Informational Note:	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: `<schema>.<element>.<_optional_qualifier>` . 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:
	
	`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
	The `Messages.properties` file controls how the fields defined above will display to the user. If the field is missing from the _Messages.properties_ file, it will not be displayed. Look in `Messages.properties}}under {{metadata.dc.<field>`. Example:
	`metadata.dc.contributor.other = Authors`
	`metadata.dc.contributor.author = Authors`
	`metadata.dc.title.* = Title`
	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).

Property:	`webui.resolver.1.urn`
	`webui.resolver.1.baseurl`
	`webui.resolver.2.urn`
	`webui.resolver.2.baseurl`

Example Value:	`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/`

Informational Note:	When using "resolver" in *webui.itemdisplay* to render identifiers as resolvable links, the base URL is take from <code>webui.resolver.<n>.baseurl<code> where <code>webui.resolver.<n>.baseurl<code> matches the urn specified in the metadata value. The value is appended to the "baseurl" as is, so the baseurl 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.

Property:	`plugin.single.org.dspace.app.webui.util.StyleSelection`

Example Value:	`plugin.single.org.dspace.app.webui.util.StyleSelection = \`
	`  org.dspace.app.web.util.CollectionStyleSelection`
	`  #org.dspace.app.web.util.MetadataStyleSelection`

Informational Note:	Specify which strategy to use for select the style for an item.
Property:	`webui.itemdisplay.thesis.collections`
Example Value:	`webui.itemdisplay.thesis.collections = 123456789/24, 123456789/35`
Informational Note:	Specify which collections use which views by Handle.
Property:	`webui.itemdisplay.metadata-style` `webui.itemdisplay.metadata-style`
Example Value:	`webui.itemdisplay.metadata-style = schema.element[.qualifier|.*]` `webui.itemdisplay.metadata-style = dc.type`
Informational Note:	Specify which metadata to use as name of the style
Property:	`webui.itemlist.columns`
Example Value:	`webui.itemlist.columns = thumbnail, dc.date.issued(date), dc.title, \` `  dc.contributor.*`

Informational Note:	Customize the DC fields to use in the item listing page. Elements will be displayed left to right in the order they are specified here. The form is <schema prefix>.<element>[.<qualifier> | .*][(date)], ... Although not a requirement, it would make sense to include among the listed fields at least the date and title fields as specified by the `webui.browse.index` configuration options in the next section mentioned. (cf.) If you have enabled thumbnails (webui.browse.thumbnail.show), you must also include a 'thumbnail' entry in your columns, this is where the thumbnail will be displayed.
Property:	`webui.itemlist.width`
Example Value:	`webui.itemlist.width = *, 130, 60%, 40%`
Informational Note:	You can customize the width of each column with the following line--you can have numbers (pixels) or percentages. For the 'thumbnail' column, a setting of '*' will use the max width specified for browse thumbnails (cf. `webui.browse.thumbnail.maxwidth, thumbnail.maxwidth`)

Property:	`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`

Example Value:	_}}
Informational Note:	You can override the DC fields used on the listing page for a given browse index and/or sort option. As a sort option or index may be defined on a field that isn't normally included in the list, this allows you to display the fields that have been indexed/sorted on. There are a number of forms the configuration can take, and the order in which they are listed below is the priority in which they will be used (so a combination of an index name and sort name will take precedence over just the browse name).In the last case, a sort option name will always take precedence over a browse index name. Note also, that for any additional columns you list, you will need to ensure there is an *itemlist.<field name>* entry in the messages file.
Property:	`webui.itemlist.dateaccessioned.columns`
Example Value:	`webui.itemlist.dateaccessioned.columns = thumbnail, dc.date.accessioned(date), dc.title, dc.contributor.*`
Informational Note:	This would display the date of the accession in place of the issue date whenever the dateaccessioned browsed index or sort option is selected. Just like *webui.itemlist.columns*, you will need to include a 'thumbnail' entry to display the thumbnails in the item list.

Property:	`webui.itemlist.dateaccessioned.widths`
Example Value:	`webui.itemlist.dateaccessioned.widths = *, 130, 60%, 40%`
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 *webui.itemlist.widths* for more information.
Property:	`webui.itemlist.tablewidth`
Example Value:	`webui.itemlist.tablewidth = 100%`
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:	`webui.session.invalidate`
Example Value:	`webui.session.invalidate = true`
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].

6.2.37 JSPUI Configuring Multilingual Support

[i18n – Locales]

Setting the Default Language for the Application

Property:	`default.locale`
Example Value:	`default.locale = en`
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:	`webui.supported.locale`
Example Value:	`webui.supported.locale = en, de`
`or perhaps`	`webui.supported.locals = en, en_ca, de`
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.proper`
- `[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

6.2.38 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.

6.2.39 Display of Group Membership

Property:	`webui.mydspace.showgroupmembership`
Example Value:	`webui.mydspace.showgroupmembership = false`
Informational Note:	To display group membership set to "true". If omitted, the default behavior is false.

6.2.40 JSPUI / XMLUI SFX Server

SFX Server is an OpenURL Resolver.

Property:	`sfx.server.url`
Example Value:	`sfx.server.url` = `http://sfx.myu.edu:8888/sfx?`
	`sfx.server.url` = `http://worldcatlibraries.org/registry/gateway?`

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>
```

6.2.41 JSPUI Item Recommendation Setting

Property:	`webui.suggest.enable`
Example Value:	webui.suggest.enable = true
Informational Note:	Show a link to the item recommendation page from item display page.
Property:	`webui.suggest.loggedinusers.only`
Example Value:	`webui.suggest.loggedinusers.only = true`
Informational Note:	Enable only if the user is logged in. If this key commented out, the default value is false.

6.2.42 Controlled Vocabulary Settings

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

Property:	`webui.controlledvocabulary.enable`
Example Value:	`webui.controlledvocabulary.enable = true`
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 an tree (taxonomy) and then use these keywords to describe items while they are being submitted.

We have also developed a small search engine that displays the classification tree (or taxonomy) allowing the user to select the branches that best describe the information that he/she seeks.

The taxonomies are described in XML following this (very simple) structure:

```
<node id="acmccs98" label="ACMCCS98">
    <isComposedBy>
        <node id="A." label="General Literature">
            <isComposedBy>
                <node id="A.0" label="GENERAL"/>
                <node id="A.1" label="INTRODUCTORY AND SURVEY"/>
            </isComposedBy>
        </node>
    </isComposedBy>
</node>
```

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:	`webui.session.invalidate`
Example Value:	`webui.session.invalidate = true`
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'.

6.2.43 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:	`xmlui.supported.locales`
Example Value:	`xmlui.supported.locales = en, de`

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:	`xmlui.force.ssl`
Example Value:	`xmlui.force.ssl = true`
Informational Note:	Force all authenticated connections to use SSL, only non-authenticated connections are allowed over plain http. If set to true, then you need to ensure that the '*dspace.hostname*' parameter is set to the correctly.
Property:	`xmlui.user.registration`
Example Value:	`xmlui.user.registration = true`
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:	`xmlui.user.editmetadata`
Example Value:	`xmlui.user.editmetadata = true`
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:	`xmlui.user.assumelogon`
Example Value:	`xmlui.user.assumelogon = true`
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:	`xmlui.user.loginredirect`
Example Value:	`xmlui.user.loginredirect = /profile`

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 */profile* for the user's profile, or another reasonable choice is */submissions* to see if the user has any tasks awaiting their attention. The default is the repository home page.
Property:	`xmlui.theme.allowoverrides`
Example Value:	`xmlui.theme.allowoverrides = false`
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:	`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, Manakin will fully verify any cache pages before using a cache copy. This means that when the community-list page is viewed the database is queried for each community/collection to see if their metadata has been modified. This can be expensive for repositories with a large community tree. To help solve this problem you can set the cache to be assumed valued for a specific set of time. The downside of this is that new or editing communities/collections may not show up the website for a period of time.

Property:	`xmlui.bistream.mods`
Example Value:	`xmlui.bistream.mods = true`
Informational Note:	Optionally, you may configure Manakin to take advantage of metadata stored as a bitstream. The MODS metadata file must be inside the "METADATA" bundle and named MODS.xml. If this option is set to 'true' and the bitstream is present then it is made available to the theme for display.
Property:	`xmlui.bitstream.mets`
Example Value:	`xmlui.bitstream.mets = true`
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:	`xmlui.google.analytics.key`
Example Value:	`xmlui.google.analytics.key = UA-XXXXXX-X`
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: _uacct = "UA-XXXXXXX-X" Take this key (just the UA-XXXXXX-X part) and place it here in this parameter.
Property:	`xmlui.controlpanel.activity.max`
Example Value:	`xmlui.controlpanel.activity.max = 250`
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:	`xmlui.controlpanel.activity.ipheader`
Example Value:	`xmlui.controlpanel.activity.ipheader = X-Forward-For`

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.

6.2.44 DSpace SOLR Statistics Configuration

Property:	`solr.log.server`
Example Value:	`solr.log.server = ${dspace.baseUrl}/solr/statistics`
Informational Note:	Is used by the SolrLogger Client class to connect to the SOLR server over http and perform updates and queries.
Property:	`solr.spidersfile`
Example Value:	`solr.spidersfile = ${dspace.dir}/config/spiders.txt`
Informational Note:	Spiders file is utilized by the SolrLogger, this will be populated by running the following command:`dsrun org.dspace.statistics.util.SpiderDetector -i <httpd log file>`
Property:	`solr.dbfile`
Example Value:	`solr.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:	`solr.resolver.timeout`
Example Value:	`solr.resolver.timeout = 200`
Informational Note:	Timeout for the resolver in the DNS lookup time in milliseconds, defaults to 200 for backward compatibility; your system's default is usually set in `/etc/resolv.conf` and varies between 2 to 5 seconds, too high a value might result in solr exhausting your connection pool.
Property:	`statistics.item.authorization.admin`

Example Value:	`statistics.item.authorization.admin = true`
Informational Note:	Enables access control restriction on DSpace Statistics pages, Restrictions are based on access rights to Community, Collection and Item Pages. This will require the user to sign on to see that statistics. Setting the statistics to "false" will make them publicly available.
Property:	`solr.statistics.logBots`
Example Value:	{{solr.statistics.logBots = true }}
Informational Note:	Enable/disable logging of spiders in solr statistics. If false, and IP matches an address in solr.spiderips.urls, event is not logged. If true, event will be logged with the 'isBot' field set to true (see `solr.statistics.query.filter.*` for query filter options) Default value is true.
Property:	`solr.statistics.query.filter.spiderIp`
Example Value:	`solr.statistics.query.filter.spiderIp = false`
Informational Note:	Controls solr statistics querying to filter out spider IPs. False by default.
Property:	{{solr.statistics.query.filter.isBot }}
Example Value:	`solr.statistics.query.filter.isBot = true`
Informational Note:	Controls solr statistics querying to look at "isBot" field to determine if record is a bot. True by default.
Property:	`solr.spiderips.urls`

Example Value:	`solr.spiderips.urls = 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/excite.txt, \`
	`http://iplists.com/misc.txt, \`
	`http://iplists.com/non_engines.txt`

Informational Note:	URLs to download IP addresses of search engine spiders from

6.3 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"

6.3.1 The Metadata Format and Bitstream Format Registries

The *[dspace]/config/registries* directory contains three XML files. These are used to load the *initial* contents of the Dublin Core Metadata registry and Bitstream Format registry and SWORD metadata registry. After the initial loading (performed by *ant fresh_install* above), the registries reside in the database; the XML files are not updated.

In order to change the registries, you may adjust the XML files before the first installation of DSpace. On an already running instance it is recommended to change bitstream registries via DSpace admin UI, but the metadata registries can be loaded again at any time from the XML files without 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/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.

6.3.2 XPDF Filter

This is an alternative suite of MediaFilter plugins that offers faster and more reliable text extraction from PDF Bitstreams, as well as thumbnail image generation. It replaces the built-in default PDF MediaFilter.

If this filter is so much better, why isn't it the default? The answer is that it relies on external executable programs which must be obtained and installed for your server platform. This would add too much complexity to the installation process, so it left out as an optional "extra" step.

Installation Overview

Here are the steps required to install and configure the filters:

1. Install the xpdf tools for your platform, from the downloads at http://www.foolabs.com/xpdf
2. Acquire the Sun Java Advanced Imaging Tools and create a local Maven package.
3. Edit DSpace configuration properties to add location of xpdf executables, reconfigure MediaFilter plugins.
4. Build and install DSpace, adding -Pxpdf-mediafilter-support to Maven invocation.

Install XPDF Tools

First, download the XPDF suite found at: http://www.foolabs.com/xpdf and install it on your server. The executables can be located anywhere, but make a note of the full path to each command.

You may be able to download a binary distribution for your platform, which simplifies installation. Xpdf is readily available for Linux, Solaris, MacOSX, Windows, NetBSD, HP-UX, AIX, and OpenVMS, and is reported to work on AIX, OS/2, and many other systems.

The only tools you *really* need are:

- *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=true
```

Edit DSpace Configuration

First, be sure there is a value for *thumbnail.maxwidth* and that it corresponds to the size you want for preview images for the UI, e.g.: (*NOTE:* this code doesn't pay any attention to *thumbnail.maxheight* but it's best to set it too so the other thumbnail filters make square images.)

```
# maximum width and height of generated thumbnails
        thumbnail.maxwidth= 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
PDFfilter.org.dspace.app.mediafilter.XPDF2Text.inputFormats = Adobe PDF
```

Finally, if you want PDF thumbnail images, don't forget to add that filter name to the *filter.plugins* property, e.g.:

```
filter.plugins = PDF Thumbnail,  PDF Text Extractor, ...
```

Build and Install

Follow your usual DSpace installation/update procedure, only add *-Pxpdf-mediafilter-support* to the Maven invocation:

```
mvn -Pxpdf-mediafilter-support package
     ant -Dconfig=\[dspace\]/config/dspace.cfg update
```

6.3.3 Creating a new Media/Format Filter

Creating a simple Media Filter

New Media Filters **must implement** the *org.dspace.app.mediafilter.FormatFilter* interface. More information on the methods you need to implement is provided in the *FormatFilter.java* source file. For example:

```
public class MySimpleMediaFilter implements FormatFilter
```

Alternatively, you could extend the *org.dspace.app.mediafilter.MediaFilter* class, which just defaults to performing no pre/post-processing of bitstreams before or after filtering.

```
public class MySimpleMediaFilter extends MediaFilter
```

You must give your new filter a "name", by adding it and its name to the *plugin.named.org.dspace.app.mediafilter.FormatFilter* field in *dspace.cfg*. In addition to naming your filter, make sure to specify its input formats in the *filter.<class path>.inputFormats* config item. Note the input formats must match the *short description* field in the Bitstream Format Registry (i.e. *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.output Format = 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");
```

6.3.4 Configuring Usage Instrumentation Plugins

A usage instrumentation plugin is configured as a singleton plugin for the abstract class org.dspace.app.statistics.AbstractUsageEvent.

The Passive Plugin

The Passive plugin is provided as the class org.dspace.app.statistics.PassiveUsageEvent. It absorbs events without effect. Use the Passive plugin when you have no use for usage event postings. This is the default if no plugin is configured.

The Tab File Logger Plugin

The Tab File Logger plugin is provided as the class org.dspace.app.statistics.UsageEventTabFileLogger. It writes event records to a file in tab-separated column format. If left unconfigured, an error will be noted in the DSpace log and no file will be produced. To specify the file path, provide an absolute path as the value for usageEvent.tabFileLogger.file in dspace.cfg.

The XML Logger Plugin

The XML Logger plugin is provided as the class org.dspace.app.statistics.UsageEventXMLLogger. It writes event records to a file in a simple XML-like format. If left unconfigured, an error will be noted in the DSpace log and no file will be produced. To specify the file path, provide an absolute path as the value for usageEvent.xmlLogger.file in dspace.cfg.

6.4 Authentication Plugins

6.4.1 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:	**`[dspace]/config/modules/authentication.cfg`**
Property:	`plugin.sequence.org.dspace.authenticate.AuthenticationMethod`
Example Value:	`plugin.sequence.org.dspace.authenticate.AuthenticationMethod = \` `org.dspace.authenticate.PasswordAuthentication`

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 (see page 235) (class: `org.dspace.authenticate.PasswordAuthentication`) (DEFAULT)
- Shibboleth Authentication (see page 236) (class: `org.dspace.authenticate.ShibAuthentication`)

- LDAP Authentication (see page 250) (class: `org.dspace.authenticate.LDAPAuthentication`)
  - Hierarchical LDAP Authentication (see page) (class: `org.dspace.authenticate.LDAPHierarchicalAuthentication`)
- IP Address based Authentication (see page 254) (class: `org.dspace.authenticate.IPAuthentication`)
- X.509 Certificate Authentication (see page 255) (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:	**`[dspace]/config/modules/authentication.cfg`**

Property:	`plugin.sequence.org.dspace.authenticate.AuthenticationMethod`
Example Value:	`plugin.sequence.org.dspace.authenticate.AuthenticationMethod = \` `org.dspace.authenticate.PasswordAuthentication`

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:	**`[dspace]/config/modules/authentication-password.cfg`**
Property:	`domain.valid`
Example Value:	`domain.value = @mit.edu, .ac.uk`
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 "@mit.edu" email addresses and all ".ac.uk" email addresses.
Property:	`login.specialgroup`
Example Value:	`login.specialgroup = My DSpace Group`
Informational Note:	This option allows you 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.

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:	**`[dspace]/config/modules/authentication.cfg`**
Property:	`plugin.sequence.org.dspace.authenticate.AuthenticationMethod`
Example Value:	`plugin.sequence.org.dspace.authenticate.AuthenticationMethod = \` `org.dspace.authenticate.ShibAuthentication`

Configuring Shibboleth Authentication (DSpace 1.8.0)

Additional Instructions

Detailed instructions for installing Shibboleth on DSpace 1.5.x may be found at https://mams.melcoe.mq.edu.au/zope/mams/pubs/Installation/dspace15.

Once it has been enabled (see above), Shibboleth Authentication is configured via its own `[dspace]/config/modules/authentication-shibboleth.cfg` file.

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

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

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

A full list of all available Shibboleth Configurations:

Configuration File:	**`[dspace]/config/modules/authentication-shibboleth.cfg`**
Property:	`email-header`

Example Value:	`email-header = MAIL`
Informational Note:	The option specifies that the email comes from the mentioned header. This value is CASE-Sensitive.
Property:	`firstname-header`
Example Value:	`firstname-header = SHIB-EP-GIVENNAME`
Informational Note:	Optional. Specify the header that carries the user's first name. This is going to be used for the creation of new-user.
Property:	`lastname-header`
Example Value:	`lastname-header = SHIB-EP-SURNAME`
Informational Note:	Optional. Specify the header that carries user's last name. This is used for creation of new user.
Property:	`email-use-tomcat-remote-user`
Example Value:	`email-use-tomcat-remote-user = true`
Informational Note:	This option forces the software to acquire the email from Tomcat.
Property:	`autoregister`
Example Value:	`autoregister = true`
Informational Note:	Option will allow new users to be registered automatically if the IdP provides sufficient information (and the user does not exist in DSpace).
Property:	`role-header` `role-header.ignore-scope`

Example Value:	`role-header = Shib-EP-ScopedAffiliation`
	`role-header.ignore-scope = true`
	
	or
	
	`role-header = Shib-EP-UnscopedAffiliation`
	`role-header.ignore-scope = false`

Informational Note:	These two options specify which attribute that is responsible for providing user's roles to DSpace and unscope the attributes if needed. When not specified, it is defaulted to 'Shib-EP-UnscopedAffiliation', and ignore-scope is defaulted to 'false'. The value is specified in *AAP.xml* (Shib 1.3.x) or *attribute-filter.xml* (Shib 2.x). The value is CASE-Sensitive. The values provided in this header are separated by semi-colon or comma. If your service provider (SP) only provides scoped role header, you need to set `role-header.ignore-Scope` as '*true*'. For example if you only get Shib-EP-ScopedAffiliation instead of Shib-EP-ScopedAffiliation, you name to make your settings as in the example value above.
Property:	`default-roles`
Example Value:	`default-roles = Staff, Walk-ins`
Informational Note:	When user is fully authN or IdP but would not like to release his/her roles to DSpace (for privacy reasons?), what should the default roles be given to such user. The values are separated by semi-colon or comma.
Property:	`role.Senior\ Researcher` `role.Librarian`
Example Value:	`role.Senior\ Researcher = Researcher, Staff` `role.Librarian = Administrator`

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

Configuring Shibboleth Authentication (DSpace 1.8.1)

Important Note

Because of a mix-up with the 1.8.1 release the configuration parameters for the Shibboleth plugin reverted to their previous 1.7.x style. These parameters should be included in your dspace.cfg instead of the modules directory for this release. Future releases will fix this and they will be moved back to the config modules style.

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. With Shibboleth DSpace will require 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.

See for more information on installing and configuring a Shibboleth Service Provider: https://wiki.shibboleth.net/confluence/display/SHIB2/Installation

Sessions:

When configuring your Shibboleth Service Provider there are two paradigms you may use: Active or Lazy Sessions. Active sessions is where the mob_shib module is configured to product a 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 where you want public access to most of DSpace but restricted access to limited areas - such as administration.

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 all ready 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 additional 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 "authentication.shib.role.<role-name>" which is a comma separated list of DSpace groups.

Configuration File:	**`[dspace]/config/dspace.cfg`**
Property:	`authentication.shib.lazysession`
Example Value:	`authentication.shib.lazysession = true`
Informational Note:	Whether to use lazy sessions or active sessions.
Property:	`authentication.shib.lazysession.loginurl`
Example Value:	`authentication.shib.lazysession.loginurl = /Shibboleth.sso/Login`
Informational Note:	The url to start a shibboleth session (only for lazy sessions)
Property:	`authentication.shib.lazysession.secure`
Example Value:	`authentication.shib.lazysession.secure = true`
Informational Note:	Force HTTPS when authenticating (only for lazy sessions)
Property:	`authentication.shib.netid-header`

Example Value:	`authentication.shib.netid-header = SHIB-NETID`
Informational Note:	The HTTP header where shibboleth will supply a user's NetID.
Property:	`authentication.shib.email-header`
Example Value:	`authentication.shib.email-header = SHIB-MAIL`
Informational Note:	The HTTP header where the shibboleth will supply a user's email address.
Property:	`authentication.shib.email-use-tomcat-remote-user`
Example Value:	`authentication.shib.email-use-tomcat-remote-user = false`
Informational Note:	Used when a netid or email heades are not available should shibboleth authentication fall back to using Tomcat's remote user feature.
Property:	`authentication.shib.autoregister`
Example Value:	`authentication.shib.autoregister = true`
Informational Note:	Should we allow new users to be registered automatically?
Property:	`authentication.shib.sword.compatability`
Example Value:	`authentication.shib.sword.compatability = true`
Informational Note:	Sword compatability 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:	`authentication.shib.firstname-header`
Example Value:	`authentication.shib.firstname-header = SHIB_GIVENNAME`
Informational Note:	The HTTP header where the shibboleth will supply a user's given name.

Property:	`authentication.shib.lastname-header`
Example Value:	`authentication.shib.lastname-header = SHIB_SN`
Informational Note:	The HTTP header where the shibboleth will supply a user's sur name.
Property:	`authentication.shib.eperson.metadata`

Example Value:	`authentication.shib.eperson.metadata = \`
	` SHIB-telephone => phone, \`
	` SHIB-cn => cn`

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:	`authentication.shib.eperson.metadata.autocreate`
Example Value:	`authentication.shib.eperson.metadata.autocreate = true`
Informational Note:	If the eperson metadata field is not found, should it be automatically created?
Property:	`authentication.shib.role-header`
Example Value:	`authentication.shib.role-header = SHIB_SCOPED_AFFILIATION`
Informational Note:	The shibboleth header to do role-based mappings (see section on roll based mapping section above)
Property:	`authentication.shib.role-header.ignore-scope`
Example Value:	`authentication.shib.role-header.ignore-scope = true`
Informational Note:	Weather to ignore the attribute's scope (everything after the @ sign for scoped attributes)
Property:	`authentication.shib.role-header.ignore-value`
Example Value:	`authentication.shib.role-header.ignore-value = false`

Informational Note:	Weather to ignore the attribute's value (everything before the @ sign for scoped attributes)
Property:	`authentication.shib.role.[affiliation-attribute]`

Example Value:	`authentication.shib.role.faculty = Faculty, Member \`
	`authentication.shib.role.staff = Staff, Member \`
	`authentication.shib.role.student = Students, Member`

Informational Note:	Mapping of affiliation values to DSpace groups.(See the roll based mapping section above)

Configuring Shibboleth Authentication (DSpace 1.8.2+)

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. With Shibboleth DSpace will require 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.

See for more information on installing and configuring a Shibboleth Service Provider: https://wiki.shibboleth.net/confluence/display/SHIB2/Installation

Sessions:

When configuring your Shibboleth Service Provider there are two paradigms you may use: Active or Lazy Sessions. Active sessions is where the mob_shib module is configured to product a 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 where you want public access to most of DSpace but restricted access to limited areas - such as administration.

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 all ready 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 additional 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:	**`[dspace]/config/modules/authentication-shibboleth.cfg`**
Property:	`lazysession`
Example Value:	`lazysession = true`
Informational Note:	Whether to use lazy sessions or active sessions.
Property:	`lazysession.loginurl`
Example Value:	`lazysession.loginurl = /Shibboleth.sso/Login`
Informational Note:	The url to start a shibboleth session (only for lazy sessions)
Property:	`lazysession.secure`
Example Value:	`lazysession.secure = true`
Informational Note:	Force HTTPS when authenticating (only for lazy sessions)
Property:	`netid-header`
Example Value:	`netid-header = SHIB-NETID`

Informational Note:	The HTTP header where shibboleth will supply a user's NetID.
Property:	`email-header`
Example Value:	`email-header = SHIB-MAIL`
Informational Note:	The HTTP header where the shibboleth will supply a user's email address.
Property:	`email-use-tomcat-remote-user`
Example Value:	`email-use-tomcat-remote-user = false`
Informational Note:	Used when a netid or email heades are not available should shibboleth authentication fall back to using Tomcat's remote user feature.
Property:	`autoregister`
Example Value:	`autoregister = true`
Informational Note:	Should we allow new users to be registered automatically?
Property:	`sword.compatability`
Example Value:	`sword.compatability = true`
Informational Note:	Sword compatability 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:	`firstname-header`
Example Value:	`firstname-header = SHIB_GIVENNAME`
Informational Note:	The HTTP header where the shibboleth will supply a user's given name.
Property:	`lastname-header`

Example Value:	`lastname-header = SHIB_SN`
Informational Note:	The HTTP header where the shibboleth will supply a user's sur name.
Property:	`eperson.metadata`

Example Value:	`eperson.metadata = \`
	` SHIB-telephone => phone, \`
	` SHIB-cn => cn`

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:	`eperson.metadata.autocreate`
Example Value:	`eperson.metadata.autocreate = true`
Informational Note:	If the eperson metadata field is not found, should it be automatically created?
Property:	`role-header`
Example Value:	`role-header = SHIB_SCOPED_AFFILIATION`
Informational Note:	The shibboleth header to do role-based mappings (see section on roll based mapping section above)
Property:	`role-header.ignore-scope`
Example Value:	`role-header.ignore-scope = true`
Informational Note:	Weather to ignore the attribute's scope (everything after the @ sign for scoped attributes)
Property:	`role-header.ignore-value`
Example Value:	`role-header.ignore-value = false`
Informational Note:	Weather to ignore the attribute's value (everything before the @ sign for scoped attributes)

Property:	`role.[affiliation-attribute]`

Example Value:	`role.faculty = Faculty, Member \`
	`role.staff = Staff, Member \`
	`role.student = Students, Member`

Informational Note:	Mapping of affiliation values to DSpace groups.(See the roll based mapping section above)

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:	**`[dspace]/config/modules/authentication.cfg`**
Property:	`plugin.sequence.org.dspace.authenticate.AuthenticationMethod`
Example Value:	`plugin.sequence.org.dspace.authenticate.AuthenticationMethod = \` `org.dspace.authenticate.LDAPAuthentication`

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 (see page 256) below for more information about how to do this.

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

Configuration File:	**`[dspace]/config/modules/authentication-ldap.cfg`**
Property:	`enable`

Example Value:	`enable = false`
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:	`autoregister`
Example Value:	`autoregister = true`
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:	`provider_url`
Example Value:	`provider_url = ldap://ldap.myu.edu/o=myu.edu`
Informational Note:	This is the url to your institution's LDAP server. You may or may not need the /o=myu.edu part at the end. Your server may also require the ldaps:// protocol.
Property:	`id_field`
Example Value:	`id_field = uid`
Explanation:	This is the unique identifier field in the LDAP directory where the username is stored.
Property:	`object_context`
Example Value:	`object_context = ou=people, o=myu.edu`
Informational Note:	This is the object context used when authenticating the user. It is appended to the *id_field* and username. For example `uid=username,ou=people,o=myu.edu`. You will need to modify this to match your LDAP configuration.
Property:	`search_context`
Example Value:	`search_context = ou=people`

Informational Note:	This is the search context used when looking up a user's LDAP object to retrieve their data for autoregistering. With `autoregister` turned on, when a user authenticates without an EPerson object we search the LDAP directory to get their name and email address so that we can create one for them. So after we have authenticated against *uid=username,ou=people,o=byu.edu* we now search in ou=people for filtering on [uid=username]. Often the `search_context` is the same as the `object_context` parameter. But again this depends on your LDAP server configuration.
Property:	`email_field`
Example Value:	`email_field = mail`
Informational Note:	This is the LDAP object field where the user's email address is stored. "mail" is the default and the most common for LDAP servers. If the mail field is not found the username will be used as the email address when creating the eperson object.
Property:	`surname_field`
Example Value:	`surname_field = sn`
Informational Note:	This is the LDAP object field where the user's last name is stored. "sn" is the default and is the most common for LDAP servers. If the field is not found the field will be left blank in the new eperson object.
Property:	`givenname_field`
Example Value:	`givenname_field = givenName`
Informational Note:	This is the LDAP object field where the user's given names are stored. I'm not sure how common the givenName field is in different LDAP instances. If the field is not found the field will be left blank in the new eperson object.
Property:	`phone_field`
Example Value:	`phone_field = telephoneNumber`
Informational Note:	This is the field where the user's phone number is stored in the LDAP directory. If the field is not found the field will be left blank in the new eperson object.
Property:	`login.specialgroup`
Example Value:	`login.specialgroup = group-name`

Informational Note:	If required, a group name can be given here, and all users who log into LDAP will automatically become members of this group. This is useful if you want a group made up of all internal authenticated users. (Remember to log on as the administrator, add this to the "Groups" with read rights).

Enabling Hierarchical LDAP Authentication

If your users are spread out across a hierarchical tree on your LDAP server, you may wish to instead use the Hierarchical LDAP Authentication plugin.

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

Configuration File:	**`[dspace]/config/modules/authentication.cfg`**
Property:	`plugin.sequence.org.dspace.authenticate.AuthenticationMethod`
Example Value:	`plugin.sequence.org.dspace.authenticate.AuthenticationMethod = \` `org.dspace.authenticate.LDAPHierarchicalAuthentication`

Configuring Hierarchical LDAP Authentication

Hierarchical LDAP Authentication shares all the above standard LDAP configurations (see page 250), 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:	**`[dspace]/config/modules/authentication-ldap.cfg`**
Property:	`search_scope`
Example Value:	`search_scope = 2`

Informational Note:	This is the search scope value for the LDAP search during autoregistering. This will depend on your LDAP server setup. This value must be one of the following integers corresponding to the following values:
	`object scope : 0`
	`one level scope : 1`
	`subtree scope : 2`

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

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:	**`[dspace]/config/modules/authentication.cfg`**
Property:	`plugin.sequence.org.dspace.authenticate.AuthenticationMethod`
Example Value:	`plugin.sequence.org.dspace.authenticate.AuthenticationMethod = \` `org.dspace.authenticate.IPAuthentication`

Configuring IP Authentication

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

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 (see page 50) 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:	**`[dspace]/config/modules/authentication.cfg`**
Property:	`plugin.sequence.org.dspace.authenticate.AuthenticationMethod`

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

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.

6.5 Batch Metadata Editing Configuration

The Batch Metadata Editing Tool (see page 390) 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:	**`[dspace]/config/modules/bulkedit.cfg`**
Property:	`valueseparator`

Example Value:	`valueseparator = ||`
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:	`fieldseparator`
Example Value:	`fieldseparator = ,`
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 `tab`, `semicolon` or `hash`. `fieldseparator = tab`
Property:	`gui-item-limit`
Example Value:	`gui-item-limit = 20`
Informational note	When using the WEBUI, this sets the limit of the number of items allowed to be edited in one processing. There is no limit when using the CLI.
Property:	`ignore-on-export`

Example Value:	`ignore-on-export = dc.date.accessioned, \`
	`dc.date.available, \`
	`dc.date.updated, dc.description.provenance`

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.

6.6 Configurable Workflow

6.6.1 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

6.6.2 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.

The submission aspect has been split up into muliple 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/" />
```

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 configration file contains the following property:

```
# Original Workflow
#workflow.framework: originalworkflow
#XML configurable workflow
workflow.framework: xmlworkflow
```

Workflow Data Migration

If you have existing workflow data in your DSpace instance, you will also need to follow the Data Migration Procedure (see page 259) below.

6.6.3 Data Migration (Backwards compatibility)

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.

SQL based migration

SQL based 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 migration script will migrate the policies, roles, tasks and workflowitems from the original workflow to the new workflow framework. The following SQL scripts are available depending on the database that is used by the DSpace installation:

- `[dspace]`/etc/oracle/xmlworkflow/workflow_migration.sql
- `[dspace]`/etc/postgres/xmlworkflow/workflow_migration.sql

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 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 adminstrator user
- -n: if sending submissions through the workflow, send notification emails
- -p: the provenance description to be added to the item
- -h: help

6.6.4 Configuration

DSpace.cfg configuration

Currently, there are no workflow configuration options added to the DSpace.cfg configuration file.

Main workflow configuration

The workflow main configuration can be found in the workflow.xml file, located in {dspace.dir}/config. An example of this workflow configuration file can be found bellow.

```
<?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 he 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 attrbiute 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 attache 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 exectution 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="oorg.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". BEach 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>
```

6.6.5 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

6.6.6 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 changes made to the database can always be found in the `[dspace]/etc/[database-type]/xmlworkflow/` directory in the file `xml_workflow.sql`. 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 described the role. The cwf_collectionrole table has the following columns:

- collectionrol_id: The identifier of the collectionrole and the primaty 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 primaty 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 primaty 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 currenlty 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_progess_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 performe the task
- finished: Keeps track of the fact that the user has finished the step or is still in progress of the execution

6.6.7 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 scrore 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. Dependingn 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.

6.6.8 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 CurationManager 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?

6.7 Discovery

6.7.1 What is DSpace Discovery

The Discovery Module for the XML user interface enables faceted searching & browsing for your repository.

Although these techniques are new in DSpace, they might feel familiar from other platforms like Aquabroser 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

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 contain condition applied to specific facets. In the example below, a user started with the search term "approach", which yielded 15 results. By applying the filter "economics" on the facet "Subject". After applying this filter, only 6 results remain.



Another example would be the standard search operation [**wetland + "dc.author=Mitsch, William J" + dc.subject="water quality"**]. With filtered search, a user can start by searching for [**wetland**], and then filter the results by the other attributes, author and subject.

6.7.2 Discovery Features

- Configurable sidebar browse facets that can display contents from any metadata field
  - Dynamically generated timespans for dates
- Customizable recent submissions display on the repository homepage, collection and community pages
- Auto-complete on search terms

6.7.3 DSpace 1.8 Improvements

- Configuration moved from dspace.cfg into config/modules/discovery.cfg and config/spring/discovery/spring-dspace-addon-discovery-configuration-services.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

6.7.4 Enabling Discovery

As with any upgrade procedure, it is highly recommend that you backup your existing data thoroughly. Although upgrades in versions of Solr/Lucene do tend to be forwards compatible for the data stored in the Lucene index, it is always a best practice to backup your dspace.dir/solr/statistics cores to assure no data is lost.

1. Enable the Discovery Aspects in the XMLUI by changing the following settings in config/xmlui.xconf
    1. Comment out: SearchArtifacts
    2. Uncomment: Discovery

```
<xmlui>
    <aspects>
        <!--
            @deprecated: the Artifact Browser has been devided into ViewArtifacts,
            BrowseArtifacts, SearchArtifacts
            <aspect name="Artifact Browser" path="resource://aspects/ArtifactBrowser/"
/>
        -->
        <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/" />

        <!--
            To enable Discovery, uncomment this Aspect that will enable it
            within your existing XMLUI
            Also make sure to comment the SearchArtifacts aspect
            as leaving it on together with discovery will cause UI overlap issues-->
        <aspect name="Discovery" path="resource://aspects/Discovery/" />


        <!--
            This aspect tests the various possible DRI features,
            it helps a theme developer create themes
        -->
        <!-- <aspect name="XML Tests" path="resource://aspects/XMLTest/"/> -->
    </aspects>
```

2. Enable the Discovery Indexing Consumer that will update Discovery Indexes on changes to content in XMLUI, JSPUI, SWORD, and LNI in config/dspace.cfg
    1. Add discovery to the list of event.dispatcher.default.consumers

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

    2. Change recent.submissions.count to zero

```
#Put the recent submissions count to 0 so that discovery can use it's recent
submissions,
# not doing this when discovery is enabled will cause UI overlap issues
#How many recent submissions should be displayed at any one time
#recent.submissions.count = 5
recent.submissions.count = 0
```

3. Check that the port is correct for solr.search.server in config/modules/discovery.cfg
    1. If all of your traffic runs over port 80, then you need to remove the port from the URL

```
##### Search Indexing #####
solr.search.server = http://localhost/solr/search
```

4. From the command line, navigate to the dspace directory and run the command below to index the content of your DSpace instance into Discovery.

```
./bin/dspace update-discovery-index
```

> NOTE: This step may take some time if you have a large number of items in your repository.

5. Verify if you now see the Sidebar Facets on your DSpace homepage. Note that these are **only visible when you have items in your repository**.

6.7.5 Configuration files

The configuration for discovery is located in 2 separate files.

- General settings: The **discovery.cfg** file located in the dspace.dir/config/modules directory.

- User Interface Configuration: The **spring-dspace-addon-discovery-configuration-services.xml** file is located in dspace.dir/config/spring directory.

6.7.6 General Discovery settings (config/modules/discovery.cfg)

The discovery.cfg file is located in the dspace.dir/config/modules directory and contains following properties:

Property:	**search.server**
Example Value:	search.server=http://localhost:8080/solr/search
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.ignore**
Example Value:	index.ignore=dc.description.provenance,dc.language
Informational Note:	By default, Discovery will include all of the DSpace metadata in its search index. In cases where specific metadata is confidential, repository managers can include those fields by adding them to this comma separated list.

6.7.7 Modifying the Discovery User Interface (config/spring/spring-dspace-addon-discovery-configuration-serv

The spring-dspace-addon-discovery-configuration-services.xml file is located in the dspace.dir/config/spring directory.

Structure Summary

Because 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 summaries 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
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:	**DiscoverySidebarFacetConfiguration**
Purpose:	Defines which metadata fields should be offered as a contextual sidebar browse option
Default:	dc.contributor.author, dc.creator, dc.subject.* and dc.date.issued
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)

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 yet, 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
  - **sidebarFacetAuthor:** groups the metadata fields dc.contributor.author & dc.creator with a facet limit of 10, sorted by occurrence count
  - **sidebarFacetSubject:** groups all subject metadata fields (dc.subject.*) with a facet limit of 10, sorted by occurrence count
  - **sidebarFacetDateIssued:** 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 and has a tokenized autocomplete
  - **searchFilterAuthor:** contains the dc.contributor.author & dc.creator metadata fields and has a non tokenized autocomplete configured
  - **searchFilterSubject:** contains the dc.subject.* metadata fields and has a non tokenized autocomplete configured
  - **searchFilterIssued:** contains the dc.date.issued metadata field with the type "date" and has a tokenized autocomplete
- 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.

Many of the properties contain lists which 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 these.

SidebarFacet Customization

This section explains the properties of an individual SidebarFacet, like SidebarFacetAuthor, SidebarFacetSubject and SidebarFacetDateIssued from the default configuration. In order to create custom SidebarFacets, you can either modify specific properties of those that already exist or create a totally new one from scratch.

Here's what the SidebarFacetAuthor looks like:

```
<bean id="sidebarFacetAuthor" 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>
```

The id & class attributes are mandatory for this type of bean. The properties that it contains are discussed below.

- **indexFieldName** (Required)**:** A unique sidebarfacet field 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.
- **facetLimit** (optional)**:** The maximum number of values to be shown. This property is optional, if none is specified 10 will be used. When a type of **date** is given, 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. If none is given the COUNT value is used as a default.
    - **COUNT** Facets will be sorted by the amount of times they appear in the repository
    - **VALUE** Facets will be sorted alphanumeric
- **type** (optional): the type of the sidebar facet it can either be date or text, if none is defined text will be used.
    - **text:** The facets will be treated as is
    - **date**: Only the year will be identified from the values and stored in the SOLR index. These years are automatically grouped together and offered as a drill-down browse.

SearchFilter Customization

This section explains the properties of an individual SearchFilter, like searchFilterTitle, searchFilterAuthor, searchFilterSubject and searchFilterIssued from the default configuration. In order to create custom Search Filters, you can either modify specific properties of those that already exist or create a totally new one from scratch.

Here's what the searchFilterAuthor looks like:

```
<bean id="searchFilterAuthor" class="org.dspace.discovery.configuration.DiscoverySearchFilter">
    <property name="indexFieldName" value="author"/>
    <property name="metadataFields">
        <list>
            <value>dc.contributor.author</value>
            <value>dc.creator</value>
        </list>
    </property>
    <property name="fullAutoComplete" value="true"/>
    <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.

- **indexFieldName** (Required)**:** A unique search filter field name, the metadata will be indexed under this field name
- **metadataFields** (Required): A list containing the metadata fields which can be used in this filter
- **fullAutoComplete** (optional): If set to true the values indexed for autocomplete will not be tokenized, if set to false tokenization will occur. Tokenization is the process of breaking up text strings in individual words. In this case, with tokenization activated, a title like "Medical Guidelines" will respond both to the "M" and to the "G", because both words are indexed individually for auto-completion.
- **type** (optional): the type of the search filter it can either be date or text, if none is defined text will be used.
    - **text:** The metadata will be treated as is
    - **date**: With a type of date the dates will receive the following format: yyyy-MM-dd (2011-07-01)

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

Configuring lists of sidebarFacets and searchFilters

Below is an example of how one of these lists can be configured. It's important that each of the bean references corresponds with the exact name of the earlier defined Facets, filters or sort options.

```
<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 discusses 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 & sidebarfacet 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"
- ...

Customizing the Recent Submissions display

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 discusses 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.

6.7.8 Routine Discovery SOLR Index Maintenance

Command used:	`[dspace]/bin/dspace update-discovery-index -o`
Java class:	org.dspace.discovery.SolrServiceImpl (or any other custom class that inherits from org.dspace.discovery.IndexingService)
Arguments (short and long forms):	Description
`-o`	Run maintenance on the Discovery SOLR index. Recommended to run daily, to prevent your servlet container from running out of memory

Notes:

The usage of this 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.

6.7.9 Advanced SOLR Configuration

Discovery is built as an application layer on top of the Open Source Enterprise Search Server SOLR. Therefor, 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
```

6.8 DSpace Service Manager

6.8.1 Introduction

The DSpace Spring Service Manager supports overriding configuration at many levels.

6.8.2 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 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

6.8.3 Architectural Overview

Please see Architectural Overview here: DSpace Services Framework (see page 507)

Service Manager Startup in Webapplications and CLI

Please see the DSpace Services Framework (see page 507)

6.8.4 Tutorials

Several good Spring / DSpace Services Tutorials are already available:

- DSpace Spring Services Tutorial
- The TAO of DSpace Services

6.9 DSpace 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.

6.9.1 What is exactly being logged ?

Each time a page or file gets requested, this request is being logged. 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.
The fields, stored in a usage event by default are:

```
<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="bundleName" type="string" indexed="true" stored="true" required="false"
multiValued="true" />
```

The combination of type (see page 477) and id determine which resource (either community, collection, item page or file download) has been requested.

6.9.2 Web user interface for DSpace statistics

In the XMLUI, 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 "statistics.item.authorization.admin" 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

6.9.3 Usage Event Logging and Usage Statistics Gathering

The DSpace Statistics Implementation is a Client/Server architecture based on Solr for collecting usage events in the JSPUI and XMLUI user interface applications of DSpace. Solr runs as a separate webapplication and an instance of Apache Http Client is utilized to allow parallel requests to log statistics events into this Solr instance.

6.9.4 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 Value:	server = http://127.0.0.1/solr/statistics

Informational Note:	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). To determine the correct path, you can use a tool like `wget` to see where Solr is responding on your server. For example, you'd want to send a query to Solr like the following: `wget http://127.0.0.1/solr/statistics/select?q=*:*` Assuming you get an HTTP 200 OK response, then you should set `solr.log.server` to the '/statistics' URL of 'http://127.0.0.1/solr/statistics' (essentially removing the "/select?q=:" query off the end of the responding URL.)
Property:	spiderips.urls

Example Value:	spiderips.urls =
	
	`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`

Informational Note:	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. 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: `dspace stats-util -h` from your [dspace]/bin directory
Property:	dbfile
Example Value:	dbfile = ${dspace.dir}/config/GeoLiteCity.dat

Informational Note:	The following referes 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:	statistics.item.authorization.admin
Example Value:	statistics.item.authorization.admin = true
Informational Note:	When set to true, only general administrators, collection and community administrators are able to access the 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:	solr.statistics.logBots
Example Value:	solr.statistics.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)
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:	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.

Upgrade Process for Statistics.

Example of rebuild and redeploy DSpace (only if you have configured your distribution in this manner)

First approach the traditional DSpace build process for updating

```
cd [dspace-source]/dspace
 mvn package
 cd [dspace-source]/dspace/target/dspace-<version>-build.dir
 ant -Dconfig=[dspace]/config/dspace.cfg update
 cp -R [dspace]/webapps/* [TOMCAT]/webapps
```

The last step is only used if you are not mounting *[dspace]/webapps* directly into your Tomcat, Resin or Jetty host (the recommended practice)If you only need to build the statistics, and don't make any changes to other web applications, you can replace the copy step above with:

```
cp -R dspace/webapps/solr TOMCAT/webapps
```

Again, only if you are not mounting [dspace]/webapps directly into your Tomcat, Resin or Jetty host (the recommended practice)

Restart your webapps (Tomcat/Jetty/Resin)

6.9.5 Older setting that are not related to the new 1.6 Statistics

The following Dspace.cfg fields are only applicable to the older statistics solution.

```
###### 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

6.9.6 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.

6.9.7 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 where 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
```

6.9.8 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.
```

6.9.9 Web UI Statistics Modification (XMLUI Only)

Modifying the number of months, for which statistics are displayed

Modify line 178 in the StatisticsTransformer.java file

dspace-xmlui/dspace-xmlui-api/src/main/java/org/dspace/app/xmlui/aspect/statistics/StatisticsTransformer.java

-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

6.9.10 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=sta
```

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>
```

6.10 Embargo

6.10.1 What is an embargo?

An embargo is a temporary access restriction placed on content, commencing at time of accession. It's 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 functionally 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:

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 section V. 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' 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.

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 it's 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.

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.

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 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 if easy for you to restrict embargoes to specific collections, since configurable submission can be defined per collection.

Key recommendations:

1. 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.
2. 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.
3. 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 resaon, 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
```

6.11 Google Scholar Metadata Mappings

Google Scholar, in crawling sites, prefers a meta-tag schema of its own devising. 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.dir}/config/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_authors = 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.

6.12 OAI

6.12.1 OAI Interfaces

OAI-PMH Activation

In the following sections, you will learn how to configure OAI-PMH and activate additional OAI-PMH crosswalks. The user is also referred to OAI-PMH Data Provider (see page 471) for greater depth details of the program.

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

Enabling OAI-PMH Interface

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

More information on the OAI-PMH protocol and its usage is available at: http://www.openarchives.org/pmh/

OAI-PMH Configuration

Configuration File:	**`[dspace]/config/modules/oai.cfg`**
Property:	`dspace.oai.url`
Example Value:	`dspace.oai.url = ${dspace.baseUrl}/oai`
Information Note:	This is the OAI-PMH URL for DSpace. By default, it is expected to be `${dspace.baseUrl}/oai`, where `dspace.baseUrl` is defined in your `dspace.cfg` file.
Property:	`didl.maxresponse`
Example Value:	`didl.maxresponse = 0`
Informational Note:	Max response size for DIDL. This is the maximum size in bytes of the files you wish to enclose Base64 encoded in your responses, remember that the base64 encoding process uses a lot of memory. We recommend at most 200000 for answers of 30 records each on a 1 Gigabyte machine. Ultimately this will change to a streaming model and remove this restriction. Also please remember to allocate plenty of memory, at least 512 MB to your Tomcat. Optional: DSpace uses 100 records as the limit for the oai responses. You can alter this by changing the `response.max-records` configuration below.
Property:	`response.max-records`
Example Value:	`response.max-records = 100`
Informational Note:	Maximum number of records to return for OAI-PMH responses. Defaults to 100.

Activating Additional OAI-PMH Crosswalks

DSpace comes with an unqualified DC Crosswalk used in the default OAI-PMH data provider. There are also other Crosswalks bundled with the DSpace distribution which can be activated by editing one or more configuration files. How to do this for each available Crosswalk is described below. The DSpace source includes the following crosswalk plugins available for use with OAI-PMH:

- ***mets*** - The manifest document from a DSpace METS SIP.
- ***mods*** - MODS metadata, produced by the table-driven MODS dissemination crosswalk.

- ***qdc*** - Qualified Dublin Core, produced by the configurable QDC crosswalk. Note that this QDC does *not* include all of the DSpace "dublin core" metadata fields, since the XML standard for QDC is defined for a different set of elements and qualifiers.
  OAI-PMH crosswalks based on Crosswalk Plugins are activated as follows:

1. Uncomment the appropriate `[dspace]/config/oaicat.properties` of the form: `Crosswalks.plugin_name=org.dspace.app.oai.PluginCrosswalk` (where `plugin_name` is the actual plugin's name, e.g. "mets" or "qdc"). These lines are all near the bottom of the file.
   - You can also add a brand new custom crosswalk plugin. Just make sure that the crosswalk plugin has a lower-case name (possibly in addition to its upper-case name) in the plugin configuration in `dspace.cfg`. Then add a line similar to above to the `oaicat.properties` file.
2. Restart your servlet container, e.g. Tomcat, for the change to take effect.
3. Verify the Crosswalk is activated by accessing a URL such as `http://[full-URL-to-OAI-PMH]/request?verb=ListRecords&metadataPrefix=mets`
   - The response should be an XML document containing METS, similar to the response from the DSpace Demo Server: http://demo.dspace.org/oai/request?verb=ListRecords&metadataPrefix=mets

DIDL
By activating the DIDL provider, DSpace items are represented as MPEG-21 DIDL objects. These DIDL objects are XML documents that wrap both the Dublin Core metadata that describes the DSpace item and its actual bitstreams. A bitstream is provided inline in the DIDL object in a base64 encoded manner, and/or by means of a pointer to the bitstream. The data provider exposes DIDL objects via the metadataPrefix didl.

The crosswalk does not deal with special characters and purposely skips dissemination of the *license.txt* file awaiting a better understanding on how to map DSpace rights information to MPEG21-DIDL.

The DIDL Crosswalk can be activated as follows:

1. Uncomment the `didl.maxresponse` configuration in `[dspace]/config/modules/oai.cfg`
2. Uncomment the DIDL Crosswalk entry from the `[dspace]/config/oaicat.properties` file
3. Restart your servlet container, e.g. Tomcat, for the change to take effect.
4. Verify the Crosswalk is activated by accessing a URL such as `http://[full-URL-to-OAI-PMH]/request?verb=ListRecords&metadataPrefix=didl`
   - The response should be an XML document containing DIDL, similar to the response from the DSpace Demo Server: http://demo.dspace.org/oai/request?verb=ListRecords&metadataPrefix=mets

OAI-PMH / OAI-ORE Harvester

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.

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:	**`[dspace]/config/modules/oai.cfg`**
Property:	`harvester.eperson`
Example Value:	`harvester.eperson = admin@myu.edu`
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:	`dspace.oai.url`
Example Value:	`dspace.oai.url = ${dspace.baseUrl}/oai`

Informational Note:	The base url of the OAI-PMH disseminator webapp (i.e. do not include the /request on the end). This is necessary in order to mint URIs for ORE Resource Maps. The default value of `${dspace.baseUrl}/oai` will work for a typical installation, but should be changed if appropriate. Please note that `dspace.baseUrl` is defined in your `dspace.cfg` configuration file.
Property:	`ore.authoritative.source`
Example Value:	`ore.authoritative.source = oai | xmlui`
Informational Note:	The webapp responsible for minting the URIs for ORE Resource Maps. If using oai, the `dspace.oai.url` config value must be set. • When set to 'oai', all URIs in ORE Resource Maps will be relative to the OAI-PMH URL (configured by `dspace.oai.url` above) • When set to 'xmlui', all URIs in ORE Resource Maps will be relative to the DSpace Base URL (configued by `dspace.url` in the `dspace.cfg` file) The URIs generated for ORE ReMs follow the following convention for either setting: http://[base-URL]/metadata/handle/[item-handle]/ore.xml
Property:	`harvester.autoStart`
Example Value:	`harvester.autoStart = false`
Informational Note:	Determines whether the harvest scheduler process starts up automatically when the XMLUI webapp is redeployed.
Property:	`harvester.oai.metadataformats.PluginName`
Example Value:	`harvester.oai.metadataformats.PluginName = \` `http://www.openarchives.org/OAI/2.0/oai_dc/, Simple Dublin Core`

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 `harvester.oai.metadataformats.PluginName = NamespaceURI,Optional Display Name` . The pluginName designates the metadata schemas that the harvester "knows" the local DSpace repository can support. Consequently, the PluginName 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 ListMetadataFormats request, and resolving it to whatever metadataPrefix 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 PluginName:NamespaceURI combo will be displayed instead.
Property:	`harvester.oai.oreSerializationFormat.OREPrefix`
Example Value:	`harvester.oai.oreSerializationFormat.OREPrefix = \` `http://www.w3.org/2005/Atom`
Informational Note:	This field works in much the same way as `harvester.oai.metadataformats.PluginName` . The `OREPrefix` must correspond to a declared ingestion crosswalk, while the Namespace must be supported by the target OAI-PMH provider when harvesting content.
Property:	`harvester.timePadding`
Example Value:	`harvester.timePadding = 120`
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:	`harvester.harvestFrequency`
Example Value:	`harvester.harvestFrequency = 720`
Informational Note:	How frequently the harvest scheduler checks the remote provider for updates. Should always be longer than *timePadding* . Measured in minutes. Default value is 720.
Property:	`harvester.minHeartbeat`
Example Value:	`harvester.minHeartbeat = 30`

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 *harvestFrequency*) value. The scheduler is optimized to then sleep until the next collection is actually ready to be harvested. The *minHeartbeat* and *maxHeartbeat* are the lower and upper bounds on this timeframe. Measured in seconds. Default value is 30.
Property:	`harvester.maxHeartbeat`
Example Value:	`harvester.maxHeartbeat = 3600`
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 *harvestFrequency*) value. The scheduler is optimized to then sleep until the next collection is actually ready to be harvested. The *minHeartbeat* and *maxHeartbeat* are the lower and upper bounds on this timeframe. Measured in seconds. Default value is 3600 (1 hour).
Property:	`harvester.maxThreads`
Example Value:	`harvester.maxThreads = 3`
Informational Note:	How many harvest process threads the scheduler can spool up at once. Default value is 3.
Property:	`harvester.threadTimeout`
Example Value:	`harvester.threadTimeout = 24`
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:	`harvester.unknownField`
Example Value:	`harvester.unkownField = fail | add | ignore`
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. *Fail* 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:	`harvester.unknownSchema`

Example Value:	`harvester.unknownSchema = fail | add | ignore`
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:	`harvester.acceptedHandleServer`
Example Value:	`harvester.acceptedHandleServer = \` `hdl.handle.net, handle.test.edu`
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: *hdl.handle.net* .
Property:	`harvester.rejectedHandlePrefix`
Example Value:	`harvester.rejectedHandlePrefix = 123456789, myeduHandle`
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: *123456789* .

Activating / Using the OAI-PMH / OAI-ORE Harvester

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 (see page 336)

6.13 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.

6.13.1 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/" />
```

6.13.2 Configuring the SWORD Client

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

Configuration File:	**`[dspace]/config/modules/sword-client.cfg`**
Property:	`targets`

Example value:	`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`

Informational note:	List of remote Sword servers. Used to build the drop-down list of selectable SWORD targets.
Property:	`file-types`
Example value:	`file-types = application/zip`
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:	`package-formats`
Example value:	`package-formats = http://purl.org/net/sword-types/METSDSpaceSIP`

Informational note:	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.

6.14 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.

6.14.1 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).

6.14.2 Configuring SWORD Server

Configuration File:	**`[dspace]/config/modules/sword-server.cfg`**
Property:	`mets-ingester.package-ingester`
Example Value:	`mets-ingester.package-ingester = METS`
Informational Note:	The property key tell the SWORD METS implementation which package ingester to use to insta deposited content. This should refer to one of the classes configured for: `plugin.named.org.dspace.content.packager.PackageIngester` The value of sword.mets-ingester.package-ingester tells the system which named plugin for this interface should be used to ingest SWORD METS packages.

Properties:	`mets.default.ingest.crosswalk.EPDCX`
	`mets.default.ingest.crosswalk.*`
	(NOTE: These configs are in the `dspace.cfg` file as they are used by many interfaces)

Example Value:	`mets.submission.crosswalk.EPDCX = EPDCX`
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, b others are supported.
Property:	`crosswalk.submission.EPDCX.stylesheet` (NOTE: This configuration is in the `dspace.cfg` file)
Example Value:	`crosswalk.submission.EPDCX.stylesheet = crosswalks/sword-swap-ingest.x`
Informational Note:	Define the stylesheet which will be used by the self-named XSLTIngestionCrosswalk class whe 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:	`deposit.url`
Example Value:	`deposit.url = http://www.myu.ac.uk/sword/deposit`
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 `${dspace.baseUrl}/sword/deposit` (where `dspace.baseUrl` is defined in your `dspace.cfg` file). In the event that you are not deploying DSpace as the ROOT application in the servlet container, this will generate incorrect URLs, and you should override the functionality by specifying in full as shown in the example value.
Property:	`servicedocument.url`
Example Value:	`servicedocument.url = http://www.myu.ac.uk/sword/servicedocument`
Informational Note:	The base URL of the SWORD service document. This is the URL from which DSpace will construct the service document location URLs for the site, and for individual collections. The default is `${dspace.baseUrl}/sword/servicedocument` (where `dspace.baseUrl` is defined in your `dspace.cfg` 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 th functionality by specifying in full as shown in the example value.
Property:	`media-link.url`

Example Value:	`media-link.url = http://www.myu.ac.uk/sword/media-link`
Informational Note:	The base URL of the SWORD media links. This is the URL which DSpace will use to construct t media link URLs for items which are deposited via sword. The default is `${dspace.baseUrl}/sword/media-link` (where `dspace.baseUrl` is defined in your `dspace.cfg` file). In the event that you are not deploying DSpace as the ROOT application in t servlet container, this will generate incorrect URLs, and you should override the functionality by specifying in full as shown in the example value.
Property:	`generator.url`
Example Value:	`generator.url = http://www.dspace.org/ns/sword/1.3.1`
Informational Note:	The URL which identifies the SWORD software which provides the sword interface. This is the URL which DSpace will use to fill out the atom:generator element of its atom documents. The default is: {{http://www.dspace.org/ns/sword/1.3.1 }}. If you have modified your SWORD software, you should change this URI to identify your owr version. If you are using the standard 'dspace-sword' module you will not, in general, need to change this setting.
Property:	`updated.field`
Example Value:	`updated.field = dc.date.updated`
Informational Note:	The metadata field in which to store the updated date for items deposited via SWORD.
Property:	`slug.field`
Example Value:	`slug.field = dc.identifier.slug`
Informational Note:	The metadata field in which to store the value of the slug header if it is supplied.
Properties:	`accept-packaging.METSDSpaceSIP.identifier` `accept-packaging.METSDSpaceSIP.q`

Example Value:	`accept-packaging.METSDSpaceSIP.identifier = http://purl.org/net/sword-types/METSDSpaceSIP` `accept-packaging.METSDSpaceSIP.q = 1.0`
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:	`accepts`
Example Value:	`accepts = application/zip, foo/bar`
Informational Note:	A comma separated list of MIME types that SWORD will accept.
Properties:	`accept-packaging.[handle].METSDSpaceSIP.identifier` `accept-packaging.[handle].METSDSpaceSIP.q`
Example Value:	`accept-packaging.[handle].METSDSpaceSIP.identifier = http://purl.org/net/sword-types/METSDSpaceSIP` `accept-packaging.[handle].METSDSpaceSIP.q = 1.0`
Informational Note:	Collection Specific settings: these will be used on the collections with the given handles.
Property:	`expose-items`
Example Value:	`expose-items = false`
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 iten which will not be forthcoming for a short while.
Property:	`expose-communities`
Example Value:	`expose-communities = false`

Informational Note:	Should the server offer as the default the list of all Communities to a Service Document request false, the server will offer the list of all collections, which is the default and recommended behav at this stage. **NOTE:** a service document for Communities will not offer any viable deposit targe and the client will need to request the list of Collections in the target before deposit can continu
Property:	`max-upload-size`
Example Value:	`max-upload-size = 0`
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, t sword service will default to no limit.
Property:	`keep-original-package`
Example Value:	`keep-original-package = true`
Informational Note:	Whether or not DSpace should store a copy of the original sword deposit package. **NOTE:** this 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 turn on. When set to "true", this requires that the configuration option `upload.temp.dir` (in `dspace.cfg`) is set to a valid location.
Property:	`bundle.name`
Example Value:	`bundle.name = SWORD`
Informational Note:	The bundle name that SWORD should store incoming packages under if sword.keep-original-package is set to true. The default is "SWORD" if not value is set
Properties:	`keep-package-on-fail` `failed-package.dir`
Example Value:	`keep-package-on-fail=true` `failed-package.dir=${dspace.dir}/upload`
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:	`identify-version`

Example Value:	`identify-version = true`
Informational Note:	Should the server identify the sword version in a deposit response. It is recommended to leave this unchanged.
Property:	`on-behalf-of.enable`
Example Value:	`on-behalf-of.enable = true`
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:	`restore-mode.enable`
Example Value:	`restore-mode.enable = true`
Informational Note:	Should the sword server enable restore-mode when ingesting new packages. If this is enabled 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:	`plugin.named.org.dspace.sword.SWORDingester`

Example Value:	`plugin.named.org.dspace.sword.SWORDIngester = \`
	`  org.dspace.sword.SWORDMETSIngester = http://purl.org/net/sword-types/METSDSpaceSIP \`
	`  org.dspace.sword.SimpleFileIngester = SimpleFileIngester`

Informational Note:	Configure the plugins to process incoming packages. The form of this configuration is as per the Plugin Manager's Named Plugin documentation: `plugin.named.[interface] = [implementation] = [package format identifier]` (see `dspace.cfg`). Package ingesters should implement the SWORDIngester interface, and will be loaded when a package the format specified above in: `accept-packaging.[package format].identifier = [package format identifier]` is received. In the event that this is a simple file deposit, w 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 existin DSpace Item.

6.15 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.

6.15.1 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).

6.15.2 Configuring SWORD v2 Server

Configuration File:	**`[dspace]/config/modules/sword2-server.cfg`**
Property:	`url`
Example Value:	`url = http://www.myu.ac.uk/swordv2`
Informational Note:	The base url of the SWORD 2.0 system. This defaults to `${dspace.baseUrl}/swordv2` (where `dspace.baseUrl` is defined in your `dspace.cfg` file).
Property:	`collection.url`
Example Value:	`url = http://www.myu.ac.uk/swordv2/collection`
Informational Note:	The base URL of the SWORD collection. This is the URL from which DSpace will construct the deposit location URLs for collections. This defaults to `${dspace.baseUrl}/swordv2/collection` (where `dspace.baseUrl` is defined in your `dspace.cfg` file).
Property:	`servicedocument.url`
Example Value:	`url = http://www.myu.ac.uk/swordv2/servicedocument`

Informational Note:	The service document URL of the SWORD collection. 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. This defaults to `${dspace.baseUrl}/swordv2/servicedocument` (where `dspace.baseUrl` is defined ir your `dspace.cfg` file).
Property:	`accept-packaging.collection`

Example Value:	`accept-packaging.collection.METSDSpaceSIP = http://purl.org/net/sword/package/METSDSpaceSIP`
	`accept-packaging.collection.SimpleZip = http://purl.org/net/sword/package/SimpleZip`
	`accept-packaging.collection.Binary = http://purl.org/net/sword/package/Binary`

Informational Note:	The accept packaging properties, along with their associated quality values where appropriate.
Property:	`accept-packaging.item`

Example Value:	`accept-packaging.item.METSDSpaceSIP = http://purl.org/net/sword/package/METSDSpaceSIP`
	`accept-packaging.item.SimpleZip = http://purl.org/net/sword/package/SimpleZip`
	`accept-packaging.item.Binary = http://purl.org/net/sword/package/Binary`

Informational Note:	The accept packaging properties for items. It is possible to configure this for specific collection by adding the handle of the collection to the setting, for example `accept-packaging.collection.[handle].METSDSpaceSIP = http://purl.org/net/sword-types/METSDSpaceSIP`
Property:	`accepts`
Example Value:	`accepts = application/zip, image/jpeg`
Informational Note:	A comma-separated list of MIME types that SWORD will accept.
Property:	`expose-communities`
Example Value:	`expose-communities = false`

Informational Note:	Whether or not the server should expose a list of all the communities to a service document request. As deposits can only be made into a collection, it is recommended to leave this set to false.
Property:	`max-upload-size`
Example Value:	`max-upload-size = 0`
Informational Note:	The maximum upload size of a package through the SWORD interface (measured in bytes). This will be the combined size of all the files, 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:	`keep-original-package`
Example Value:	`keep-original-package = true`
Informational Note:	Should DSpace store a copy of the orignal SWORD deposit package? This will cause the deposit process to be slightly slower and for more disk to be used, however original files will be preserved. It is recommended to leave this option enabled.
Property:	`bundle-name`
Example Value:	`bundle-name = SWORD`
Informational Note:	The bundle name that SWORD should store incoming packages within if `keep-original-package` is set to true.
Property:	`keep-package-on-fail`
Example Value:	`keep-package-on-fail = false`
Informational Note:	In the event of package ingest failure, provide an option to store the package on the file system The default is false. The location can be set using the `failed-package-dir` setting.

Property:	`failed-package-dir`
Example Value:	`failed-package-dir = /dspace/upload`
Informational Note:	If `keep-package-on-fail` is set to true, this is the location where the package would be stored.
Property:	`on-behalf-of.enable`
Example Value:	`on-behalf-of.enable = true`
Informational Note:	Should DSpace accept mediated deposits? See the SWORD specification for a detailed explanation of deposit On-Behalf-Of another user.
Property:	`generator.url`
Example Value:	`generator.url = http://www.dspace.org/ns/sword/2.0/`
Informational Note:	The URL which identifies DSpace as the software that is providing the SWORD interface.
Property:	`generator.version`
Example Value:	`generator.version = 2.0`
Informational Note:	The version of the SWORD interface.
Property:	`auth-type`
Example Value:	`auth-type = Basic`
Informational Note:	Which form of authentication to use. Normally this is set to `Basic` in order to use HTTP Basic.
Property:	`upload.tempdir`

Example Value:	`upload.tempd = /dspace/upload`
Informational Note:	The location where uploaded files and packages are stored while being processed.
Property:	`updated.field`
Example Value:	`updated.field = dc.date.updated`
Informational Note:	The metadata field in which to store the updated date for items deposited via SWORD.
Property:	`slug.field`
Example Value:	`slug.field = dc.identifier.slug`
Informational Note:	The metadata field in which to store the value of the slug header if it is supplied.
Property:	`author.field`
Example Value:	`author.field = dc.contributor.author`
Informational Note:	The metadata field in which to store the value of the atom entry author if it supplied.
Property:	`title.field`
Example Value:	`dc.title`
Informational Note:	The metadata field in which to store the value of the atom entry title if it supplied.
Property:	`disseminate-packaging`

Example Value:	`disseminate-packaging.METSDSpaceSIP = http://purl.org/net/sword/package/METSDSpaceSIP` `disseminate-packaging.SimpleZip = http://purl.org/net/sword/package/SimpleZip`
Informational Note:	Supported packaging formats for the dissemination of packages.
Property:	`plugin.single.org.dspace.sword2.WorkflowManager`
Example Value:	`plugin.single.org.dspace.sword2.WorkflowManager = org.dspace.sword2.WorkflowManagerDefault`
Informational Note:	Which workflow manager to use.
Property:	`mets-ingester.package-ingester`
Example Value:	`mets-ingester.package-ingester = METS`
Informational Note:	Which package ingester to use for METS packages.
Property:	`restore-mode.enable`
Example Value:	`restore-mode.enable = false`
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 has previously been assigned a handle then that same handle will be restored to activity.
Property:	`simpledc.*`
Example Value:	`simpledc.abstract = dc.description.abstractsimpledc.date = dc.datesimpledc.rights = dc.rights`

Informational Note:	Configuration of metadata field mapping used by the SimpleDCEntryIngester.
Property:	`multipart.entry-first`
Example Value:	`multipart.entry-first = false`
Informational Note:	The order of precedence for importing multipart content. If this is set to `true` then metadata in the package will override metadata in the atom entry, otherwise the metadata in the atom entry will override that from the package.
Property:	`workflow.notify`
Example Value:	`workflow.notify = true`
Informational Note:	If the workflow gets started (the collection being deposited into has a workflow configured), should a notification get sent?
Property:	`versions.keep`
Example Value:	`versions.keep = true`
Informational Note:	When content is replaced, should the old version be kept? This creates a copy of the ORIGINAL bundle with the name V_YYYY-MM-DD.X where YYYY-MM-DD is the date the copy was created, and X is an integer from 0 upwards.
Property:	`state.*`

Example Value:	`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 archive`

Informational Note:	Pairs of states (URI and description) than items can be in. Typical states are `workspace`, `workflow`, `archive`, and `withdrawn`.

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.SwordContentIngester = \
  org.dspace.sword2.SimpleZipContentIngester = http://purl.org/net/sword/package/SimpleZip, \
  org.dspace.sword2.SwordMETSIngester = http://purl.org/net/sword/package/METSDSpaceSIP, \
  org.dspace.sword2.BinaryContentIngester = http://purl.org/net/sword/package/Binary, \
  org.dspace.swordpackagers.SwordDocXIngester =
application/vnd.openxmlformats-officedocument.wordprocessingml.document, \
  org.dspace.swordpackagers.SwordXifIngester = image/jpeg

plugin.single.org.dspace.sword2.SwordEntryIngester = \
  org.dspace.sword2.SimpleDCEntryIngester

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
```

7 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.

7.1 Configuration

The user will need to refer to the extensive WebUI/JSPUI configurations (see page 137) that are contained in JSP Web Interface Settings.

7.2 Customizing the JSP pages

The JSPUI interface is implemented using Java Servlets which handle the business logic, and JavaServer Pages (JSPs) which produce the HTML pages sent to an end-user. Since the JSPs are much closer to HTML than Java code, altering the look and feel of DSpace is relatively easy.

To make it even easier, DSpace allows you to 'override' the JSPs included in the source distribution with modified versions, that are stored in a separate place, so when it comes to updating your site with a new DSpace release, your modified versions will not be overwritten. It should be possible to dramatically change the look of DSpace to suit your organization by just changing the CSS style file and the site 'skin' or 'layout' JSPs in *jsp/layout*; if possible, it is recommended you limit local customizations to these files to make future upgrades easier.

You can also easily edit the text that appears on each JSP page by editing the *Messages.properties* file. However, note that unless you change the entry in all of the different language message files, users of other languages will still see the default text for their language. See Internationalization in Application Layer.

Note that the data (attributes) passed from an underlying Servlet to the JSP may change between versions, so you may have to modify your customized JSP to deal with the new data.

Thus, if possible, it is recommended you limit your changes to the 'layout' JSPs and the stylesheet.

The JSPs are available in one of two places:

- *[dspace-source]/dspace-jspui/dspace-jspui-webapp/src/main/webapp/* - Only exists if you downloaded the full Source Release of DSpace
- *[dspace-source]/dspace/target/dspace-[version].dir/webapps/dspace-jspui-webapp/* - The location where they are copied after first building DSpace.

If you wish to modify a particular JSP, place your edited version in the ***[dspace-source]/dspace/modules/jspui/src/main/webapp/*** directory (*this is the replacement for the pre-1.5 /jsp/local directory*), with the same path as the original. If they exist, these will be used in preference to the default JSPs. For example:

DSpace default	Locally-modified version
[jsp.dir]/community-list.jsp	*[jsp.custom-dir]/dspace/modules/jspui/src/main/webapp/community-list.jsp*
[jsp.dir]/mydspace/main.jsp	*[jsp.custom-dir]/dspace/modules/jspui/src/main/webapp/mydspace/main.jsp*

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-[version]-build.dir* directory:

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

3. Deploy the the new webapps:

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

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

8 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.

8.1 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 (see page 536) - 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 (see page 536).
- Aspects (see page 330) - 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 (see page 332) - 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 (see page 333)
    - Additional Theme Resources include:
        - Manakin theme tutorial
        - Manakin Themes and Recipes
        - Create a new theme (Manakin)

8.1.1 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**
    1. Within that sitemap, various URL path matching takes place. If the request is to download a document, that document is returned immediately.
    2. 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.xmap` file, which controls all the Themes.
        1. The `themes.xmap` file will then load all "matching" themes which are configured in your `[dspace]/config/xmlui.xconf` file (see Themes (see page 332) below).
        2. If more than one theme matches the current URL path, then the **first match wins**
        3. Once a matching theme is located, that theme's `sitemap.xmap` file (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.xmap` file to be loaded. As the name suggests, this file obviously controls all the Aspects.
    1. The `aspects.xmap` file will then load all enabled Aspects which are configured in your `[dspace]/config/xmlui.xconf` file (see Aspects (see page 330) below).
    2. 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.*
    3. 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/aspect`
    4. 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).

8.2 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:	*xmlui.supportedLocales*
Example Value:	*xmlui.supportedLocales = en, de*
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:	*xmlui.force.ssl*
Example Value:	*xmlui.force.ssl = true*
Informational Note:	Force all authenticated connections to use SSL, only non-authenticated connections are allowed over plain http. If set to true, then you need to ensure that the '*dspace.hostname*' parameter is set to the correctly.
Property:	*xmlui.user.registration*
Example Value:	*xmlui.user.registration = true*
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:	*xmlui.user.editmetadata*
Example Value:	*xmlui.user.editmetadata = true*
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:	*xmlui.user.assumelogon*
Example Value:	*xmlui.user.assumelogon = true*
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:	*xmlui.user.loginredirect*
Example Value:	*xmlui.user.loginredirect = /profile*
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 */profile* for the user's profile, or another reasonable choice is */submissions* to see if the user has any tasks awaiting their attention. The default is the repository home page.
Property:	*xmlui.theme.allowoverrides*
Example Value:	*xmlui.theme.allowoverrides = false*
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:	*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, Manakin will fully verify any cache pages before using a cache copy. This means that when the community-list page is viewed the database is queried for each community/collection to see if their metadata has been modified. This can be expensive for repositories with a large community tree. To help solve this problem you can set the cache to be assumed valued for a specific set of time. The downside of this is that new or editing communities/collections may not show up the website for a period of time.
Property:	*xmlui.bistream.mods*
Example Value:	*xmlui.bistream.mods = true*
Informational Note:	Optionally, you may configure Manakin to take advantage of metadata stored as a bitstream. The MODS metadata file must be inside the "METADATA" bundle and named MODS.xml. If this option is set to 'true' and the bitstream is present then it is made available to the theme for display.
Property:	*xmlui.bitstream.mets*
Example Value:	*xmlui.bitstream.mets = true*
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:	*xmlui.google.analytics.key*

Example Value:	*xmlui.google.analytics.key = UA-XXXXXX-X*
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 snipit of javascript code to place on your site, inside that snip it is your Google Analytics key usually found in the line: _uacct = "UA-XXXXXXX-X" Take this key (just the UA-XXXXXX-X part) and place it here in this parameter.
Property:	*xmlui.controlpanel.activity.max*
Example Value:	*xmlui.controlpanel.activity.max = 250*
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:	*xmlui.controlpanel.activity.ipheader*
Example Value:	*xmlui.controlpanel.activity.ipheader = X-Forward-For*
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.

8.3 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.

8.3.1 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 reposonsible for displaying individual item metadata.
- **BrowseArtifacts** The BrowseArtifacts Aspect is reponsible 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

- **XMLWorkfflow** 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 togather 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

8.3.2 Themes

The <*themes*> section defines a set of "rules" that determine where themes are installed in the repository. Each rule is processed in the order that it appears, and the first rule that matches determines the theme that is applied (so order is important). Each rule consists of a <*theme*> element with several possible attributes:

- **name** (*always required*)The name attribute is used to document the theme's name.
- **path** (*always required*)The path attribute determines where the theme is located relative to the *themes/* directory and must either contain a trailing slash or point directly to the theme's *sitemap.xmap* file.
- **regex** (*either regex and/or handle is required*)The regex attribute determines which URLs the theme should apply to.
- **handle** (*either regex and/or handle is required*)The handle attribute determines which community, collection, or item the theme should apply to.
  If you use the "handle" attribute, the effect is cascading, meaning if a rule is established for a community then all collections and items within that community will also have this theme apply to them as well. Here is an example configuration:

```
<themes>
        <theme name="Theme 1" handle="123456789/23" path="theme1/"/>
        <theme name="Theme 2" regex="community-list" path="theme2/"/>
        <theme name="Reference Theme" regex=".*" path="Reference/"/>
    </themes>
```

  In the example above three themes are configured: "Theme 1", "Theme 2", and the "Reference Theme". The first rule specifies that "Theme 1" will apply to all communities, collections, or items that are contained under the parent community "123456789/23". The next rule specifies any URL containing the string "community-list" will get "Theme 2". The final rule, using the regular expression ".**", will match *anything**, so all pages which have not matched one of the preceding rules will be matched to the Reference Theme.

8.4 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*.

Manakin supplies an English only translation of the interface. In order to add other translations to the system, locate the *[dspace-source]/dspace/modules/xmlui/src/main/webapp/i18n/* directory. By default this directory will be empty; to add additional translations add alternative versions of the *messages.xml* file in specific language and country variants as needed for your installation.

To set a language other than English as the default language for the repository's interface, simply name the translation catalogue for the new default language "*messages.xml*"

8.5 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".

8.6 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>
```

8.7 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"/>
  <img src="./static/images/static-image.gif" alt="Static image in /static/images/ directory"/>
  <img src="./static/static-image.jpg" alt="Static image in /static/ directory"/>
```

8.8 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)
    - "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 (see page 338) 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.

8.8.1 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.

8.9 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)

8.10 Mirage Configuration and Customization

8.10.1 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

8.10.2 Configuration Parameters

Property:	xmlui.theme.mirage.item-list.emphasis
Example Value:	xmlui.theme.mirage.item-list.emphasis = metadata
Informational Note:	Determines which style should be used to display item lists. Allowed values: • **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:	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.
Property:	xmlui.theme.enableMinification
Example Value:	xmlui.theme.enableMinification = false

Informational Note:	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.

8.10.3 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.

8.10.4 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"> </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

8.11 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 (see page 342) - used in the generation of default Reference, Classic and Kubrick themes
- dri2xhtml-alt (see page 343) - used in the generation of default Mirage theme

You only should use **one** of these two templates, based on which seems easier to you.

8.11.1 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.

8.11.2 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 (see page 338)

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.xsl
       community-list.xsl
       community-view.xsl
       item-list.xsl
       item-view.xsl
    general
        choice-authority-control.xsl
 core
    attribute-handlers.xsl
    elements.xsl
    forms.xsl
    global-variables.xsl
    navigation.xsl
    page-structure.xsl
    utils.xsl
 dri2xhtml.xsl
```

9 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.

9.1 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 overlayed 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.)

9.2 DSpace Source Release

If you really want to get your hands dirty you may have downloaded the 'dspace-src-release' (or checked out the latest DSpace Code via Subversion) and wish to make changes to dspace-api source code. In that case it should be noted that DSpace should be rebuilt by running '`mvn package`' from the root `[dspace-source]` directory rather than from `[dspace-source]/dspace`. This will ensure that your modifications are included in the final WAR files.

In other words, with the 'dspace-src-release' there are two build options:

1. 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.
2. Running `mvn package` from the `[dspace-source]/dspace/` directory
    - This option performs a "quick build". It does **not** rebuild all DSpace modules. All it does is re-apply any Maven WAR Overlays to the previously compiled source code.

10 System Administration

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

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

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

DSpace Command Launcher

With DSpace Release 1.6, the many commands and scripts have been replaced with a simple `[dspace]/bin/dspace <command>` command. See Application Layer chapter for the details of the DSpace Command Launcher (see page 475).

10.1 AIP Backup and Restore

10.1.1 Background & Overview

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 (see page 372). 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 (see page 372)). 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 (see page 513) 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
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

Supports backup/restore of Item Mappings between Collections	Yes	Yes (During restore, the AIP Ingester may throw a false "Could not find a parent DSpaceObject" error (see Common Issues or Error Messages (see page 371)), 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 (see page 363) 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 (see page 371) 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 your 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 work help DSpace interact with DuraCloud?

This work is entirely about **exporting** DSpace content objects to a location on a local filesystem. So, this work doesn't interact solely with DuraCloud, and could be used by any backup storage system to backup your DSpace contents.

In the initial DuraCloud work, the DuraCloud team is working on a way to "synchronize" DuraCloud with a local file folder. So, DuraCloud can be configured to "watch" a given folder and automatically replicate its contents into the cloud.

Therefore, moving content from DSpace to DuraCloud would currently be a two-step process:

1. First, export AIPs describing that content from DSpace to a filesystem folder
2. Second, enable DuraCloud to watch that same filesystem folder and replicate it into the cloud.

Similarly, moving content from DuraCloud back into DSpace would also be a two-step process:

1. First, you'd tell DuraCloud to replicate the AIPs from the cloud to a folder on your file system
2. Second, you'd ingest those AIPs back into DSpace

(These backup/restore processes may change as we go forward and investigate more use cases. This is just the initial plan.)

10.1.2 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 (see page 372)

10.1.3 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 (see page 353) (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 (see page) (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 (see page 356) (`-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 (see page 359) (`-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 (see page 361) (`-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 `-a` option 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 (see page 356) (`-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)

- Restore / Replace Mode (see page 359) (`-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 (see page 363) 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 (see page 355) 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 (see page 359).

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 (see page 362)).

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 Colleciton 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 (see page 357). 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 (see page 355) above.

There are currently three restore modes:

1. Default Restore Mode (see page 359) (`-r`) = Attempt to restore object (and optionally children). Rollback all changes if any object is found to already exist.
2. Restore, Keep Existing Mode (see page 361) (`-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 (see page 361) (`-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 can be run while the system is online (i.e. no need to stop Tomcat or PostgreSQL). 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 (see page 361) *or* the Force Replace Mode (see page 361) (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 (see page 38)

- 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
/full/path/to/your/site-aip.zip
```

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 (see page 354). 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 can be run while the system is online (i.e. no need to stop Tomcat or PostgreSQL). 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`

10.1.4 Additional Packager Options

In additional to the various "modes" settings described under "Running the Code (see page 352)" 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
`createMetadataFields`	ingest-only	true	Tells the AIP ingester 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** create missing DSpace Metadata *Schemas*. If a schema is found to be missing, the ingest will always fail.)
`filterBundles`	export-only	defaults to exporting all Bundles	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. *WARNING any bundles* **not** *included in AIPs will obviously be unable to be restored.* This option can be run in two ways: • **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. (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 `aip.disseminate.rightsMD` configuration for the changes to take affect)

`ignoreHandle`	ingest-only	Restore/Replace Mode defaults to 'false', Submit Mode defaults to 'true'	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 `-s` 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 `-r` flag).
`ignoreParent`	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 `-s` flag). The new Parent object must be specified via the `-p` flag (run `dspace packager -h` 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 `-r` flag).
`includeBundles`	export-only	defaults to "all"	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. *WARNING any bundles **not** included in AIPs will obviously be unable to be restored.* 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. (See "filterBundles" option above if you wish to exclude particular Bundles. However, this "includeBundles" option **cannot** be used at the same time as "filterBundles".) (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 `aip.disseminate.rightsMD` configuration for the changes to take affect)
`manifestOnly`	both	false	If 'true', the AIP Disseminator will export an AIP which only consists of the METS Manifest file (i.e.

			result will be a single 'mets.xml' file). This METS Manifest contains URI references to all content files, but does *not* contain any content files. **This option is experimental, and should never be set to 'true' if you want to be able to restore content files.**
`passwords`	export-only	false	If 'true' (and the 'DSPACE-ROLES' crosswalk is enabled, see AIP Metadata Dissemination Configurations (see page 368)), 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`	import-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 large 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 (see page 371).
`unauthorized`	export-only	*unspecified*	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`	export-only	*unspecified*	This option works as a basic form of "incremental backup". This option requires that an ISO-8601 date is 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`	both	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 (see page 370)" section below). *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.*

How to use these 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");
```

10.1.5 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 (see page 368) 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
```

10.1.6 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 (see page 362), but are not running the command in Force Replace Mode (`-r -f`). Please see the section on Restoring an Entire Site (see page 362) 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 'skipIfParentMissing=true' option (see Additional Packager Options (see page 363)). 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 'skipIfParentMissing=true' option. All your Item Mappings should still be restored correctly.
Submit Error: PSQLException: 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 the may occur in DSpace if your Handle sequence has somehow become out-of-date. However, it's easy to fix. Just run the `[dspace]/etc/postgres/update-sequences.sql` script (or if you are using Oracle, run: `[dspace]/etc/oracle/update-sequences.sql`).

10.1.7 DSpace AIP Format

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).

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 (see page 363).)
- 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 Commmunity-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 (see page 368). 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 (see page 363).

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 (see page 380) 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 (see page 379) 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 @OTHERTYPE 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 (see page 383) 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 (see page 384) 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 (see page 387) 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) (see page 381) section below for more information.
    - `digiprovMD` 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/file` elements
            - 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 `mptr` elements:
        1. One linking to the Handle of that Item. Its `@LOCTYPE="HANDLE"`, and `@xlink:href` value is the raw Handle.
        2. (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 `mptr` elements:
            1. One linking to the Handle of that Community. Its `@LOCTYPE="HANDLE"`, and `@xlink:href` value is the raw Handle.
            2. (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 `mptr` elements:
            1. One linking to the Handle of that Collection. Its `@LOCTYPE="HANDLE"`, and `@xlink:href` value is the raw Handle.
            2. (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 `mptr` elements:
            1. One linking to the Handle of that Community. Its `@LOCTYPE="HANDLE"`, and `@xlink:href` value is the raw Handle.
            2. (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

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 (*as a URN*)
dc.relation.isReferencedBy	All other Collection's this item is linked to (*Handle URN of each non-owner*)
dc.rights.accessRights	*"WITHDRAWN"* 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
dc.relation.isPartOf	Owning Community's Handle (*as a URN*)
dc.relation.isReferencedBy	

	All other Communities this Collection is linked to (*Handle URN of each non-owner*)

AIP Technical Metadata for Community

Metadata Field	Value
dc.identifier.uri	Handle of Community
dc.relation.isPartOf	Handle of Parent Community (*as a URN*)

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 withn 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>
      <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>
```

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>
```

10.2 Batch Metadata Editing

10.2.1 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 CVS format. The batch editing tool facilitates the user to perform the following:

- Batch editing of metadata (e.g. perform an external spell check)
- Batch additions of metadata (e.g. add an abstract to a set of items, add controlled vocabulary such as LCSH)
- Batch find and replace of metadata values (e.g. correct misspelled surname across several records)
- Mass move items between collections
- 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 (see page 256)

Export Function

The following table summarizes the basics.

	`[dspace]/bin/dspace metadata-export`

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

Exporting Process

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

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

Example:

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

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

Import Function

The following table summarizes the basics.

Command used:	`[dspace]/bin/dspace metadata-import`
Java class:	org.dspace.app.bulkedit.MetadataImport
Arguments short and (long) forms:	Description
`-f` or `--file`	Required. The filename of the CSV file to load.
`-s` or `--silent`	

	Silent mode. The import function does not prompt you to make sure you wish to make the changes.
`-e` or `--email`	The email address of the user. This is only required when adding new items.
`-w` or `--workflow`	When adding new items, the program will queue the items up to use the Collection Workflow processes.
`-n` or `--notify`	when adding new items using a workflow, send notification emails.
`-t` or `--template`	When adding new items, use the Collection template, if it exists.
`-h` or `--help`	Display the brief help page.

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

Importing Process

To run the batch importer, at the command line:

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

Example

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

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

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

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



Importing large CSV files

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

The CSV Files

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

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

A typical heading row looks like:

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

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

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

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

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

Editing Collection Membership

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

Adding 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.

10.3 Curation System

As of release 1.7, DSpace supports running curation tasks, which are described in this section. DSpace 1.7 and subsequent distributions will bundle (include) several useful tasks, 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.

10.3.1 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

10.3.2 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.

10.3.3 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`

10.3.4 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.

10.3.5 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 vscan -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

> **Not available for JSPUI**
>
> At this point in time, Curation Tasks cannot be run from the JSPUI Admin interface. However, users of the JSPUI can still run Curation Tasks from the Command Line or from Workflow.

In the XMLUI, there are several ways to execute configured Curation Tasks:

1. **From the 'Curate' tab 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).

10.3.6 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.

10.3.7 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");
```

10.3.8 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.

10.3.9 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 @Supendable 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.

10.3.10 Scripted Tasks

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.

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;
```

10.3.11 Starter Tasks

DSpace 1.7 bundles a few tasks and activates two (2) by default to demonstrate the use of the curation system. These may be removed (deactivated by means of configuration) if desired without affecting system integrity. Each task is briefly described here.

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 be performed on each item, and will display the *last* 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)

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 1st Infected Bitstream

Container	F	Stop on 1st Infected Item
Item	T	Stop on 1st Infected Bitstream
Item	F	Scan all bitstreams
Command Line		
Container	T	Report on 1st infected bitstream within an item/Scan all contained Items
Container	F	Report on all infected bitstreams/Scan all contained Items
Item		Report on 1st infected bitstream
Item		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 cab 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
```

10.4 Importing and Exporting Content via Packages

10.4.1 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 (see page 372). This is used as part of the DSpace AIP Backup and Restore (see page 347) process
- DSPACE-ROLES - Ingests DSpace users/groups in the DSPACE-ROLES XML Schema (see page 384). This is primarily used by the DSpace AIP Backup and Restore (see page 347) 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 (see page 372). This is used as part of the DSpace AIP Backup and Restore (see page 347) process
- DSPACE-ROLES - Exports DSpace users/groups in the DSPACE-ROLES XML Schema (see page 384). This is primarily used by the DSpace AIP Backup and Restore (see page 347) 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 Packagers Support Bulk Ingest

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

- METS Packager Plugin
- AIP Packager Plugin (see page 347)

Restoring/Replacing using Packages

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

There are currently three restore modes:

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

Default Restore Mode

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

Use this 'packager' command template:

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

For example:

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

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

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

Restore, Keep Existing Mode

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

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

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

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

For example:

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

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

Force Replace Mode

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

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

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

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

For example:

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

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

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

Disseminating

Disseminating a Single Object

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

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

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

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

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

Disseminating Multiple Objects at Once

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

For example, use this 'packager' command template:

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

for example:

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

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

Archival Information Packages (AIPs)

As of DSpace 1.7, DSpace now can backup and restore all of its contents as a set of AIP Files (see page 372). 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 (see page 372)). 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 (see page 347).

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

10.5 Importing and Exporting Items via Simple Archive Format

10.5.1 Item Importer and Exporter

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

DSpace Simple Archive Format

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

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

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

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

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

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

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

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

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

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

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

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

Where '\t' is the tab character.

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

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

'DESCRIPTION' is text of the files description.

Primary is used to specify the primary bitstream.

Configuring metadata_[prefix].xml for Different Schema

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

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

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

Importing Items

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

Command used:	`[dspace]/bin/dspace import`

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

‡ These are mutually exclusive.

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

Adding Items to a Collection

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

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

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

or by using the short form:

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

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

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

Replacing Items in Collection

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

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

Long form:

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

Deleting or Unimporting Items in a Collection

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

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

In long form:

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

Other Options

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

Exporting Items

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

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

Exporting a Collection

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

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

Short form:

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

Exporting a Single Item

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

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

Short form:

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

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

The `-m` Argument

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

10.6 Importing Community and Collection Hierarchy

10.6.1 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
-f	Source xml file.
-o	Output xml file.
-e	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>
                   <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

10.7 Managing Community Hierarchy

10.7.1 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:	`[dspace]/bin/dspace community-filiator`
Java class:	*org.dspace.administer.CommunityFiliator*
Arguments short and (long) forms:	Description

`-s` or `--set`	Set a parent/child relationship
`-r` or `--remove`	Remove a parent/child relationship
`-c` or `--child`	Child community (Handle or database ID)
`-p` or `--parent`	Parent community (Handle or database ID
`-h` or `--help`	Online help.

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

```
[dspace]/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.

10.8 Managing Embargoed Content

10.8.1 Embargo Lifter

If you have implemented the Embargo (see page 294) feature, you will need to run it periodically to check for Items with expired embargoes and lift them.

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

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

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

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

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

10.9 Managing Usage Statistics

10.9.1 DSpace Log Converter

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

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

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

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

Command used:	`[dspace]/bin/dspace stats-log-importer`
Java class:	org.dspace.statistics.util.StatisticsImporter
Arguments (short and	Description

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

Although the DSpace Log Convertor applies basic spider filtering (googlebot, yahoo slurp, msnbot), it is far from complete. Please refer to Filtering and Pruning Spiders (see page 429) for spider removal operations, after converting your old logs.

10.9.2 Filtering and Pruning Spiders

Command used:	`[dspace]/bin/dspace stats-util`
Java class:	org.dspace.statistics.util.StatisticsClient
Arguments (short and long forms):	Description
`-u` or `-update-spider-files`	Update Spider IP Files from internet into `/dspace/config/spiders`. Downloads Spider files identified in `dspace.cfg` under property `solr.spiderips.urls`. See DSpace SOLR Statistics Configuration (see page 225)
`-f` or `-delete-spiders-by-flag`	Delete Spiders in Solr By isBot Flag. Will prune out all records that have `isBot:true`
`-i` or `-delete-spiders-by-ip`	Delete Spiders in Solr By IP Address. Will prune out all records that have IP's that match spider IPs.
`-m` or `-mark-spiders`	Update isBog Flag in Solr. Marks any records currently stored in statistics that have IP addresses matched in spiders files
`-h` or `-help`	Calls up this brief help table at command line.

Notes:

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

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

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

10.9.3 Routine SOLR Index Maintenance

Command used:	`[dspace]/bin/dspace stats-util`
Java class:	org.dspace.statistics.util.StatisticsClient
Arguments (short and long forms):	Description
`-o` or `-optimize`	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 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.

10.10 Moving Items

10.10.1 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.

10.10.2 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 (see page 393) section under Batch Metadata Editing (see page 390)).

10.11 Performance Tuning DSpace

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.

10.11.1 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 >= 512MB of PermGen Space 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 `/bin/etc/default/tomcat6` appears to be the best place to put these environmental variables.

10.11.2 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`

10.11.3 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

10.11.4 SOLR Statistics Performance Tuning

This 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.

10.12 Registering (not Importing) Bitstreams via Simple Archive Format

10.12.1 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 (see page 417) 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 (see page 417) 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 (see page).

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

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

Internal Identification and Retrieval of Registered Items

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

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

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

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

Fourth, an MD5 checksum is calculated by reading the registered file if it is in local storage. If the 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 (see page)) using the exporter and importer, the registered files in the export directory will again 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 (see page 417)) 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.

10.13 ReIndexing Content (for Browse or Search)

10.13.1 Overview

DSpace offers two options to index content for Browsing & Searching:

1. Traditional Browse & Search (via Lucene & Database indexes) - this is enabled by default
2. Faceted/Filtered Browse & Search (via Solr & DSpace Discovery (see page 270)) - available for XMLUI only and disabled by default

This particular page only describes the "Traditional Browse & Search" indexing processes. For more information on Faceted/Filtered Browse & Search, please see DSpace Discovery (see page 270).

10.13.2 Creating the Browse & Search Indexes

To create (or recreate) all the various browse/search indexes that you define in the Configuration Section (see page 137) there are a variety of options available to you. You can see these options below in the command table.

Command used:	`[dspace]/bin/dspace index-init`
Java class:	org.dspace.browse.IndexBrowse
Arguments short and long forms):	Description
`-r` or `-rebuild`	Should we rebuild all the indexes, which removes old tables and creates new ones. For use with `-f`. Mutually exclusive with `-d`
`-s` or `-start`	`-s <int>` start from this index number and work upwards (mostly only useful for debugging). For use with `-t` and `-f`
`-x` or `-execute`	Execute all the remove and create SQL against the database. For use with `-t` and `-f`
`-i` or `-index`	Actually do the indexing. Mutually exclusive with `-t` and `-f`.
`-o` or `-out`	`-o <filename>` write the remove and create SQL to the given file. For use with `-t` and `-f`
`-p` or `-print`	Write the remove and create SQL to the stdout. For use with `-t` and `-f`.
`-t` or `-tables`	Create the tables only, do no attempt to index. Mutually exclusive with `-f` and `-i`
`-f` or `-full`	Make the tables, and do the indexing. This forces `-x`. Mutually exclusive with `-f` and `-i`.
`-v` or `-verbose`	Print extra information to the stdout. If used in conjunction with `-p`, you cannot use the stdout to generate your database structure.
`-d` or `-delete`	Delete all the indexes, but do not create new ones. For use with `-f`. This is mutually exclusive with `-r`.
`-h` or `-help`	Show this help documentation. Overrides all other arguments.

10.13.3 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-init`. After the indexing command completes, you can restart Tomcat.

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

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

Updating the Indexes

By running `[dspace]/bin/dspace index-update` you will reindex your full browse & 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-update
```

Destroy and Rebuild Browse Tables

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. **WARNING: This is not really recommended unless you know what you are doing.**

At the CLI screen:

```
[dspace]/bin/dspace index \-r \-t \-p \-v \-x \-o myfile.sql
```

10.13.4 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 (see page 194) to become familiar with the property keys and the definitions used therein before attempting heavy customizations.

Through customization is is possible to:

- Add new browse indexes besides the four that are delivered upon installation. Examples:
  - Series
  - Specific subject fields (Library of Congress Subject Headings). *(It is possible to create a browse index based on a controlled vocabulary or thesaurus.)*
  - Other metadata schema fields
- Combine metadata fields into one browse
- Combine different metadata schemas in one browse

Examples of new browse indexes that are possible. *(The system administrator is reminded to read the section on Browse Index Configuration (see page 194))*

- **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`
- **Separate subject browse.** You may want to have a separate subject browse limited to only one type of subject.
  - `webui.browse.index.7 = lcsubject.metdata:dc.subject.lcsh.text:single`

As one can see, the choices are limited only by your metadata schema, the metadata, and your imagination.

Because Browse Indexes are stored in database tables, remember to run `index-init` after adding any new definitions in the `dspace.cfg` to have the indexes created and the data indexed.

Search Index Customization

For information about configuring new Search Indexes, please refer to Configuring Lucene Search Indexes (see page 161).

10.14 Testing Database Connection

10.14.1 Test Database

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

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

10.15 Transferring or Copying Content Between Repositories

10.15.1 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

You can transfer any DSpace content (Communities, Collections or Items) from one DSpace to another by utilizing the AIP Backup and Restore (see page 347) 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 (see page 347).

10.15.2 Transferring Items using Simple Archive Format

Migration of Data

Where items are to be moved between DSpace instances (for example from a test DSpace into a production DSpace) the Item Exporter and Item Importer (see page 417) can be used in conjunction with a script to assist in this process.

First, you should export the DSpace Item(s) into the Simple Archive Format, as detailed at: Importing and Exporting Items via Simple Archive Format (see page 417)

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

- date.accessioned
- date.available
- date.issued

- description.provenance
- format.extent
- format.mimetype
- identifier.uri

In order to avoid duplication of this metadata, run

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

prior to running the item importer. This will remove the above metadata items, except for `date.issued` (if the item has been published or publicly distributed before) and `identifier.uri` (if it is not the handle), from the `dublin_core.xml` file and remove all `handle` files. It will then be safe to run the item importer (see page 417)
.

10.15.3 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 Harverter (see page 336). 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 (see page 336)

10.15.4 Copying Items using the SWORD Client

10.16 Transforming DSpace Content (MediaFilters)

10.16.1 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 (see page 171) 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	`org.dspace.app.mediafilter.HTMLFilter`	extracts the full text of HTML documents for full text indexing	true
JPEG Thumbnail	`org.dspace.app.mediafilter.JPEGFilter`	creates thumbnail images of GIF, JPEG and PNG files	true
Branded Preview JPEG	`org.dspace.app.mediafilter.BrandedPreviewJPEGFilter`	creates a branded preview image for GIF, JPEG and PNG files (disabled by default)	false
PDF Text Extractor	`org.dspace.app.mediafilter.PDFFilter`	extracts the full text of Adobe PDF documents (only if text-based or OCRed) for full text indexing	true
Word Text Extractor	`org.dspace.app.mediafilter.WordFilter`	extracts the full text of Microsoft Word or Plain Text documents for full text indexing	true
	`org.dspace.app.mediafilter.PowerPointFilter`		true

PowerPoint Text Extractor		extracts the full text of slides and notes in Microsoft PowerPoint and PowerPoint XML documents for full text indexing	

Please note that the `filter-media` script will automatically update the DSpace search index by default (see ReIndexing Content (for Browse or Search) (see page 437)) 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) (see page) 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 (see page 171) 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 (see page 231) 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.

10.17 Updating Items via Simple Archive Format

10.17.1 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 (see page 417), 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 (see page 417).

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

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

ItemUpdate Commands

Command used:	`[dspace]/bin/dspace itemupdate`
Java class:	*org.dspace.app.itemupdate.ItemUpdate*
Arguments short and (long) forms:	Description
`-a` or `--addmetadata [metadata element]`	Repeatable for multiple elements. The metadata element should be in the form dc.x or dc.x.y. The mandatory argument indicates the metadata fields in the dublin_core.xml file to be added unless already present (multiple fields should be separated by a semicolon ';'). However, duplicate fields will not be added to the item metadata without warning or error.

`-d` or `--deletemetadata [metadata element]`	Repeatable for multiple elements. All metadata fields matching the element will be deleted.
`-A` or `--addbitstreams`	Adds bitstreams listed in the contents file with the bitstream metadata cited there.
`-D` or `--deletebitstreams [filter plug classname or alias]`	Not repeatable. With no argument, this operation deletes bitstreams listed in the `deletes_contents` file. Only bitstream IDs are recognized identifiers for this operation. The optional filter argument is the classname of an implementation of `org.dspace.app.itemdupate.BitstreamFilter` class to identify files for deletion or one of the aliases (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 `delete_contents` file is not required for any item. The filter properties file will contains properties pertinent to the particular filer used. Multiple filters are not allowed.
`-h` or `--help`	Displays brief command line help.
`-e` or `--eperson`	Email address of the person or the user's database ID **(Required)**
`-s` or `--source`	Directory archive to process **(Required)**
`-i` or `--itemfield`	Specifies the metadata field that contains the item's identifier; Default value is "dc.identifier.uri" (Optional)
`-t` or `--test`	Runs the process in test mode with logging. But no changes applied to the DSpace instance. (Optional)
`-P` or `--provenance`	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).*

10.18 Validating CheckSums of Bitstreams

10.18.1 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 beys in the *dspace.cfg* file.

Checker Execution Mode

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

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

Available command line options

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

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

  The checker will keep starting new bitstream checks for the specific durations, so actual execution duration will be slightly longer than the specified duration. Bear this in mind when scheduling checks.
- **Specific Bitstream mode:** `[dspace]/bin/dspace checker -b` Checker will only look at the internal bitstream IDs. Example: `[dspace]/bin/dspace checker -b 112 113 4567` Checker will only check bitstream IDs 112, 113 and 4567.
- **Specific Handle mode:** `[dspace]/bin/dspace checker -a` Checker will only check bitstreams within the Community, Community or the item itself. Example: `[dspace]/bin/dspace checker -a 123456/999` Checker will only check this handle. If it is a Collection or Community, it will run through the entire Collection or Community.
- **Looping mode:** `[dspace]/bin/dspace checker -l` or `[dspace]/bin/dspace checker -L` There are two modes. The lowercase 'el' (-l) specifies to check every bitstream in the repository once. This is recommended for smaller repositories who are able to loop through all their content in just a few

hours maximum. An uppercase 'L' (-L) specifies to continuously loops through the repository. This is not recommended for most repository systems. **Cron Jobs**. For large repositories that cannot be completely checked in a couple of hours, we recommend the -d option in cron.

- **Pruning mode:** `[dspace]/bin/dspace checker -p` The Checksum Checker will store the result of every check in the checksum_history table. By default, successful checksum matches that are eight weeks old or older will be deleted when the -p option is used. (Unsuccessful ones will be retained indefinitely). Without this option, the retention settings are ignored and the database table may grow rather large!

Checker Results Pruning

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

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

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

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

Checker Reporting

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

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

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

Cron or Automatic Execution of Checksum Checker

You should schedule the Checksum Checker to run automatically, based on how frequently you backup your DSpace instance (and how long you keep those backups). The size of your repository is also a factor. For very

large repositories, you may need to schedule it to run for an hour (e.g. `-d 1h` option) each evening to ensure it makes it through your entire repository within a week or so. Smaller repositories can likely get by with just running it weekly.

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

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

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

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

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

(This command should appear on a single line).

Automated Checksum Checkers' Results

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

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

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

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

11 Directories and Files

11.1 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*.

11.2 Source Directory Layout

- *[dspace-source]*
  - *dspace/* - Directory which contains all build and configuration information for DSpace
    - *CHANGES* - Detailed list of code changes between versions.
    - *KNOWN_BUGS* - Known bugs in the current version.
    - *LICENSE* - DSpace source code license.
    - *README* - Obligatory basic information file.
    - *bin/* - Some shell and Perl scripts for running DSpace command-line tasks.
    - *config/* - Configuration files:
      - *controlled-vocabularies/* - Fixed, limited vocabularies used in metadata entry
      - *crosswalks/* - Metadata crosswalks - property files or XSL stylesheets
      - *dspace.cfg* - The Main DSpace configuration file (You will need to edit this).
      - *dc2mods.cfg* - Mappings from Dublin Core metadata to MODS for the METS export.
      - *default.license* - The default license that users must grant when submitting items.
      - *dstat.cfg* , *dstat.map* - Configuration for statistical reports.
      - *input-forms.xml* - Submission UI metadata field configuration.
      - *news-side.html* - Text of the front-page news in the sidebar, only used in JSPUI.

        - *news-top.html* - Text of the front-page news in the top box, only used in teh JSPUI.
        - *emails/* - Text and layout templates for emails sent out by the system.
        - *registries/* - **Initial** contents of the bitstream format registry and Dublin Core element/qualifier registry. These are only used on initial system setup, after which they are maintained in the database.
    - *docs/* - DSpace system documentation. The technical documentation for functionality, installation, configuration, etc.
    - *etc/* -
      This directory contains administrative files needed for the install process and by developers, mostly database initialization and upgrade scripts. Any *.xml* files in *etc/* are common to all supported database systems.
        - *postgres/* - Versions of the database schema and updater SQL scripts for PostgreSQL.
        - *oracle/* - Versions of the database schema and updater SQL scripts for Oracle.
    - *modules/* - The Web UI modules "overlay" directory. DSpace uses Maven to automatically look here for any customizations you wish to make to DSpace Web interfaces.
        - *jspui* - Contains all customizations for the JSP User Interface.
            - *src/main/resources/* - The overlay for JSPUI *Resources.* This is the location to place any custom Messages.properties files. *(Previously this file had been stored at: _[dspace-source]/config/language-packs/Messages.properties_*
            - *src/main/webapp/* - The overlay for JSPUI Web Application. This is the location to place any custom JSPs to be used by DSpace.
        - *lni* - Contains all customizations for the Lightweight Network Interface.
        - *oai* - Contains all customizations for the OAI-PMH Interface.
        - *sword* - Contains all customizations for the SWORD (Simple Web-service Offering Repository Deposit) Interface.
        - *xmlui* - Contains all customizations for the XML User Interface (aka Manakin).
            - *src/main/webapp/* - The overlay for XMLUI Web Application. This is the location to place custom Themes or Configurations.
                - *i18n/* - The location to place a custom version of the XMLUI's messages.xml (You have to manually create this folder)
                - *themes/* - The location to place custom Themes for the XMLUI (You have to manually create this folder).
    - *src/* - Maven configurations for DSpace System. This directory contains the Maven and Ant build files for DSpace.
    - *target/* - (Only exists after building DSpace) This is the location Maven uses to build your DSpace installation package.
        - *dspace-[version].dir* - The location of the DSpace Installation Package (which can then be installed by running *ant update*)

- The Source Release contains the following additional directories :-
    - *dspace-api/* - Java API source module
    - *dspace-discovery* - Discovery source module

- *dspace-jspui/* - JSP-UI source module
- *dspace-oai* - OAI-PMH source module
- *dspace-xmlui* - XML-UI (Manakin) source module
- *dspace-lni* - Lightweight Network Interface source module
- *dspace-stats* - Statistics source module
- *dspace-sword* - SWORD (Simple Web-serve Offering Repository Deposit) deposit service source module
- *dspace-swordv2* - SWORDv2 source module
- *dspace-sword-client* - XMLUI client for SWORD source module
- *pom.xml* - DSpace Parent Project definition

11.3 Installed Directory Layout

Below is the basic layout of a DSpace installation using the default configuration. These paths can be configured if necessary.

- *[dspace]*
  - *assetstore/* - asset store files
  - *bin/* - shell and Perl scripts
  - *config/* - configuration, with sub-directories as above
  - *handle-server/* - Handles server files
  - *history/* - stored history files (generally RDF/XML)
  - *lib/* - JARs, including dspace.jar, containing the DSpace classes
  - *log/* - Log files
  - *reports/* - Reports generated by statistical report generator
  - *search/* - Lucene search index files
  - *upload/* - temporary directory used during file uploads etc.
  - *webapps/* - location where DSpace installs all Web Applications

11.4 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

11.5 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 (see page 338))
    - *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) (see page 342) 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) (see page 342) 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

11.6 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

[dspace]/log/dspace.log.yyyy-mm-dd	Main DSpace log file. This is where the DSpace code writes a simple log of events and errors that occur within the DSpace code. You can control the verbosity of this by editing the *[dspace-source]/config/templates/log4j.properties* file and then running "*ant init_configs*".
[dspace]/log/cocoon.log.yyyy-mm-dd	Apache Cocoon log file for the XMLUI. This is where the DSpace XMLUI logs all of its events and errors.
[tomcat]/logs/catalina.out	This is where Tomcat's standard output is written. Many errors that occur within the Tomcat code are logged here. For example, if Tomcat can't find the DSpace code (*dspace.jar*), it would be logged in *catalina.out*.
[tomcat]/logs/hostname_log.yyyy-mm-dd.txt	If you're running Tomcat stand-alone (without Apache), it logs some information and errors for specific Web applications to this log file. *hostname* will be your host name (e.g. *dspace.myu.edu*) and *yyyy-mm-dd* will be the date.
[tomcat]/logs/apache_log.yyyy-mm-dd.txt	If you're using Apache, Tomcat logs information about Web applications running through Apache (*mod_webapp*) in this log file (*yyyy-mm-dd* being the date.)
[apache]/error_log	Apache logs to this file. If there is a problem with getting *mod_webapp* working, this is a good place to look for clues. Apache also writes to several other log files, though *error_log* tends to contain the most useful information for tracking down problems.
[dspace]/log/handle-plug.log	The Handle server runs as a separate process from the DSpace Web UI (which runs under Tomcat's JVM). Due to a limitation of log4j's 'rolling file appenders', the DSpace code running in the Handle server's JVM must use a separate log file. The DSpace code that is run as part of a Handle resolution request writes log information to this file. You can control the verbosity of this by editing *[dspace-source]/config/templates/log4j-handle-plugin.properties*.
[dspace]/log/handle-server.log	This is the log file for CNRI's Handle server code. If a problem occurs within the Handle server code, before DSpace's plug-in is invoked, this is where it may be logged.
[dspace]/handle-server/error.log	On the other hand, a problem with CNRI's Handle server code might be logged here.
PostgreSQL log	PostgreSQL also writes a log file. This one doesn't seem to have a default location, you probably had to specify it yourself at some

	point during installation. In general, this log file rarely contains pertinent information--PostgreSQL is pretty stable, you're more likely to encounter problems with connecting via JDBC, and these problems will be logged in *dspace.log*.

11.6.1 log4j.properties File.

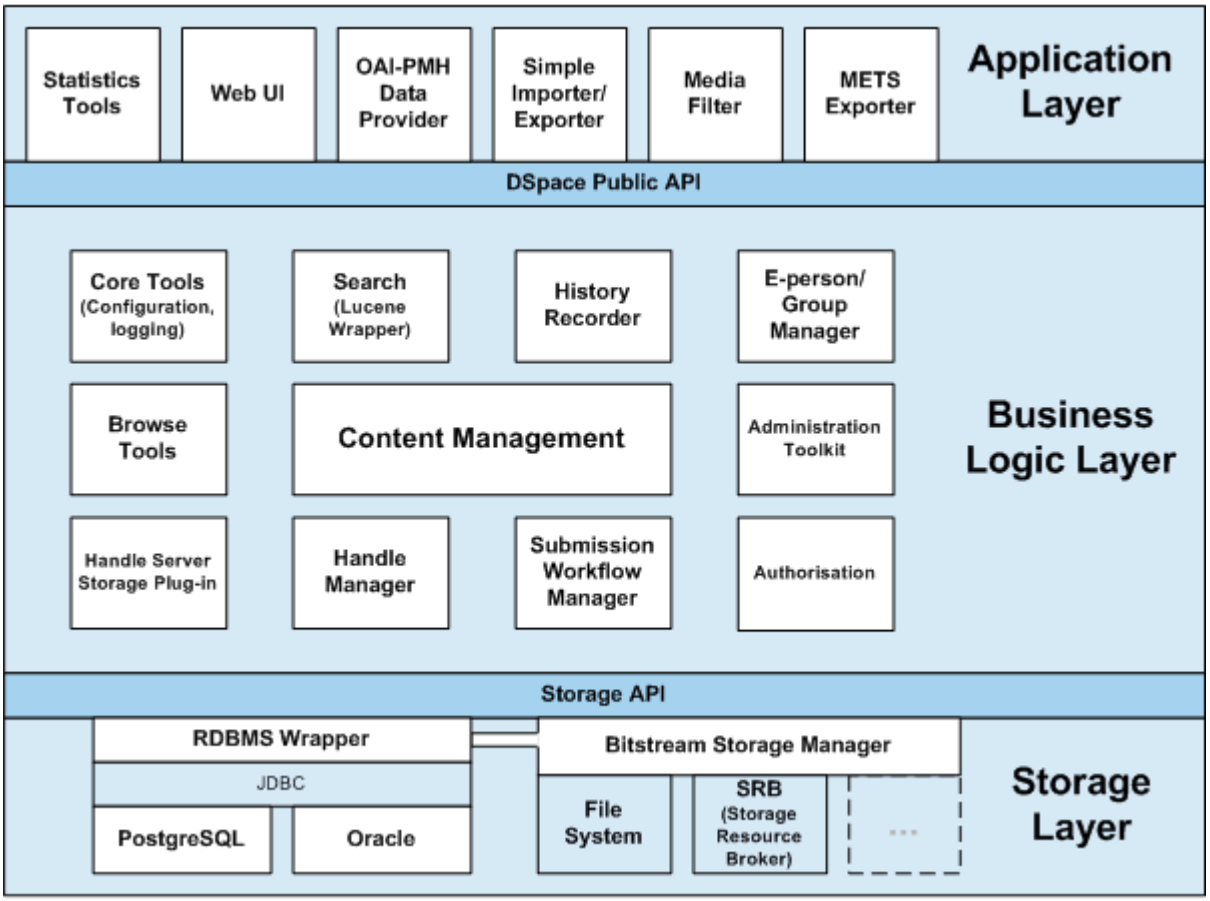
the file *[dspace]/config/log4j.properties* controls how and where log files are created. There are three sets of configurations in that file, called A1, A2, and A3. These are used to control the logs for DSpace, the checksum checker, and the XMLUI respectively. The important settings in this file are:

`log4j.rootCategory=INFO,A` `log4j.logger.org.dspace=INFO,A1`	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
log4j.appender.A1=org.dspace.app.util.DailyFileAppender	This is the name of the log file creation method used. The DailyFileAppender creates a new date-stamped file every day or month.
log4j.appender.A1.File=${log.dir}/dspace.log	This sets the filename and location of where the log file will be stored. It iwll have a date stamp appended to the file name.
log4j.appender.A1.DatePattern=yyy-MM-DD	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 *yyyy-MM*
log4j.appender.A1.MaxLogs=0	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.

12 Architecture

12.1 Overview

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



12.1.1 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**
org.dspace.app	Application layer
org.dspace	Business logic layer (except *storage* and *app*)
org.dspace.storage	Storage layer

The storage and business logic layer APIs are extensively documented with Javadoc-style comments. Generate the HTML version of these by entering the [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 (see page 513)
  - RDBMS
  - Bitstream Store
- Business Logic Layer (see page 476)
  - 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 (see page 461)
    - Web User Interface
    - OAI-PMH Data Provider
    - Item Importer and Exporter
    - Transferring Items Between DSpace Instances
    - Registration
    - METS Tools
    - Media Filters
    - Sub-Community Management

12.2 Application Layer

The following explains how the application layer is built and used.

12.2.1 Web User Interface

The DSpace Web UI is the largest and most-used component in the application layer. Built on Java Servlet and JavaServer Page technology, it allows end-users to access DSpace over the Web via their Web browsers. As of Dspace 1.3.2 the UI meets both XHTML 1.0 standards and Web Accessibility Initiative (WAI) level-2 standard.

It also features an administration section, consisting of pages intended for use by central administrators. Presently, this part of the Web UI is not particularly sophisticated; users of the administration section need to know what they are doing! Selected parts of this may also be used by collection administrators.

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 familar 'Ant' to deploy the webapps in later stages.

Location	Description
[dspace-source]/dspace-jspui/dspace-jspui-api/src/main/java/org/dspace/app/webui	Web UI source
[dspace-source]/dspace-jspui/dspace-jspui-api/src/main/java/org/dspace/app/filters	Servlet Filters (Servlet 2.3 sp

[dspace-source]/dspace-jspui/dspace-jspui-api/src/main/java/org/dspace/app/jsptag	Custom JSP ta class files
[dspace-source]/dspace-jspui/dspace-jspui-api/src/main/java/org/dspace/app/servlet	Servlets for ma Web UI (contro
[dspace-source]/dspace-jspui/dspace-jspui-api/src/main/java/org/dspace/app/servlet/admin	Servlets that comprise the administration of the Web UI
[dspace-source]/dspace-jspui/dspace-jspui-api/src/main/java/org/dspace/app/webui/util/	Miscellaneous classes used b servlets and fil
[dspace-source]/dspace-jspui	The JSP files
[dspace-source]/dspace/modules/jspui/src/main/webapp	This is where y place customiz versions of JSI see JSPUI Configuration a Customization page 322)
[dspace-source]/dspace/modules/xmlui/src/main/webapp	This is where y place customiz for the Manaki interface, see XMLUI Configu and Customiza (see page 324
[dspace-source/dspace/modules/jspui/src/main/resources	This is where y can place you customize vers the *Messages.prop* file.
[dspace-source]/dspace-jspui/dspace-jspui-webapp/src/main/webapp/WEB-INF/dspace-tags.tld	Custom DSpac JSP tag descri

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-[version]-build.dir/*. 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-[version]-build.dir/`
    - This DSpace "installation template" folder has a structure identical to the Installed Directory Layout (see page 455)

In order to then install & deploy DSpace from this "installation template" folder, you must run the following from `[dspace-source]/dspace/target/dspace-[version]-build.dir/` :

```
ant -D [dspace]/config/dspace.cfg update
```

Please see the Installation (see page 38) 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 *HttpRequest* object that represents the HTTP request being processed. This is done by invoking the *setAttribute* method of the *javax.servlet.http.HttpServletRequest* object that is passed into the servlet from Tomcat. The servlet then forwards control of the request to the appropriate JSP using the *JSPManager.showJSP* method.

The *JSPManager.showJSP* method uses the standard Java servlet forwarding mechanism is then used to forward the HTTP request to the JSP. The JSP is processed by Tomcat and the results sent back to the user's browser.

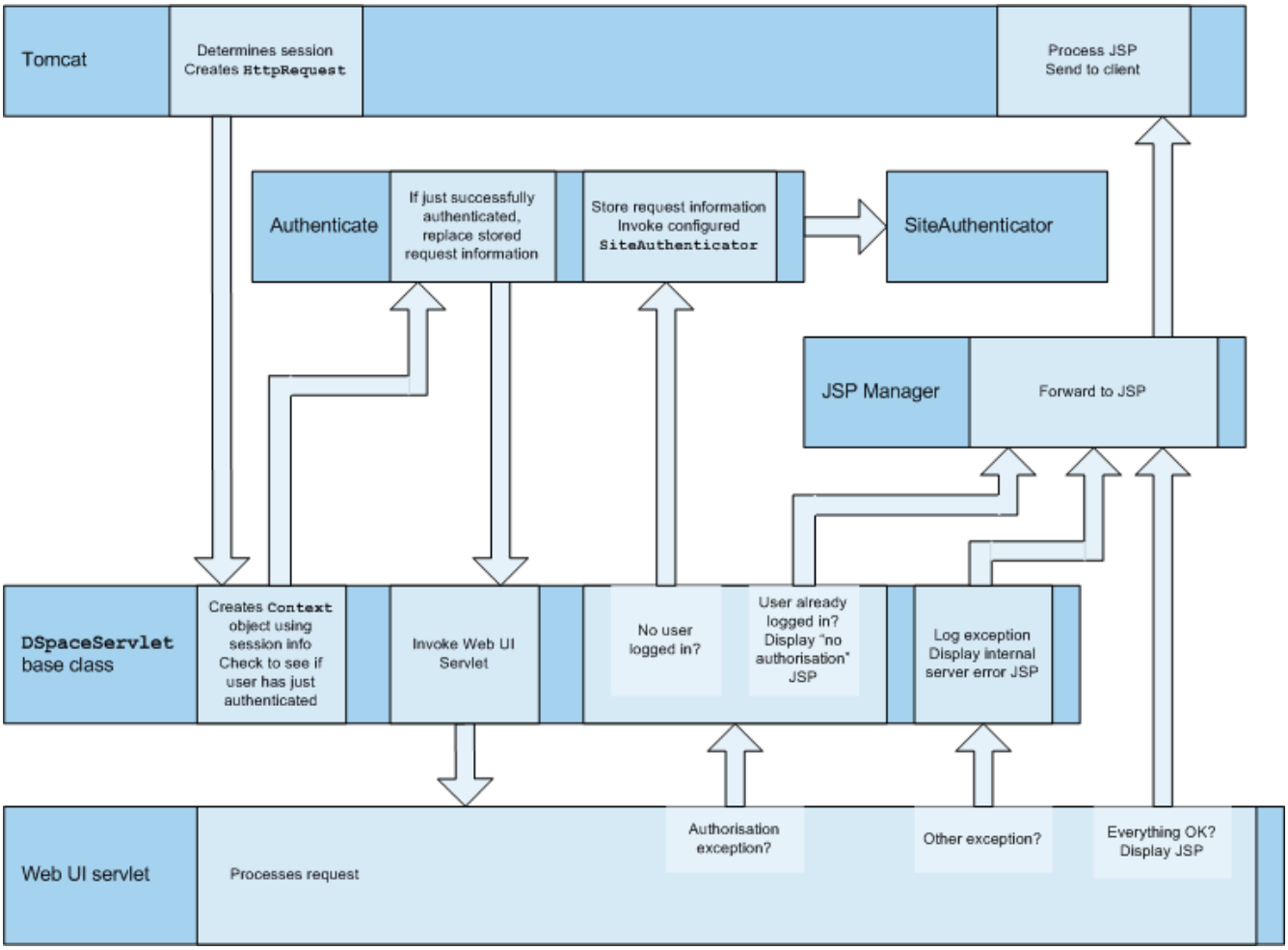
There is an exception to this servlet/JSP style: *index.jsp*, the 'home page', receives the HTTP request directly from Tomcat without a servlet being invoked first. This is because in the servlet 2.3 specification, there is no way to map a servlet to handle only requests made to '/'; such a mapping results in every request being directed to that servlet. By default, Tomcat forwards requests to '/' to *index.jsp*. To try and make things as clean as possible, *index.jsp* contains some simple code that would normally go in a servlet, and then forwards to *home.jsp* using the *JSPManager.showJSP* method. This means localized versions of the 'home page' can be created by placing a customized *home.jsp* in *[dspace-source]/jsp/local*, in the same manner as other JSPs.

[dspace-source]/jsp/dspace-admin/index.jsp, the administration UI index page, is invoked directly by Tomcat and not through a servlet for similar reasons.

At the top of each JSP file, right after the license and copyright header, is documented the appropriate attributes that a servlet must fill out prior to forwarding to that JSP. No validation is performed; if the servlet does not fill out the necessary attributes, it is likely that an internal server error will occur.

Many JSPs containing forms will include hidden parameters that tell the servlets which form has been filled out. The submission UI servlet (*SubmissionController* is a prime example of a servlet that deals with the input from many different JSPs. The *step* and *page* hidden parameters (written out by the *SubmissionController.getSubmissionParameters()* method) are used to inform the servlet which page of which step has just been filled out (i.e. which page of the submission the user has just completed).

Below is a detailed, scary diagram depicting the flow of control during the whole process of processing and responding to an HTTP request. More information about the authentication mechanism is mostly described in the configuration section.



Flow of Control During HTTP Request Processing

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.jsptag*. The tags are listed below. The *dspace-tags.tld* file contains detailed comments about how to use the tags, so that information is not repeated here.

- ***layout***: Just about every JSP uses this tag. It produces the standard HTML header and *<BODY>_tag. Thus the content of each JSP is nested inside a _<dspace:layout>* tag. The (XML-style)attributes of this tag are slightly complicated--see *dspace-tags.tld*. The JSPs in the source code bundle also provide plenty of examples.
- ***sidebar***: Can only be used inside a *layout* tag, and can only be used once per JSP. The content between the start and end *sidebar* tags is rendered in a column on the right-hand side of the HTML page. The contents can contain further JSP tags and Java 'scriptlets'.

- ***date***: Displays the date represented by an *org.dspace.content.DCDate* object. Just the one representation of date is rendered currently, but this could use the user's browser preferences to display a localized date in the future.
- ***include***: Obsolete, simple tag, similar to *jsp:include*. In versions prior to DSpace 1.2, this tag would use the locally modified version of a JSP if one was installed in jsp/local. As of 1.2, the build process now performs this function, however this tag is left in for backwards compatibility.
- ***item***: Displays an item record, including Dublin Core metadata and links to the bitstreams within it. Note that the displaying of the bitstream links is simplistic, and does not take into account any of the bundling structure. This is because DSpace does not have a fully-fledged dissemination architectural piece yet. Displaying an item record is done by a tag rather than a JSP for two reasons: Firstly, it happens in several places (when verifying an item record during submission or workflow review, as well as during standard item accesses), and secondly, displaying the item turns out to be mostly code-work rather than HTML anyway. Of course, the disadvantage of doing it this way is that it is slightly harder to customize exactly what is displayed from an item record; it is necessary to edit the tag code (*org.dspace.app.webui.jsptag.ItemTag*). Hopefully a better solution can be found in the future.
- ***itemlist***, ***collectionlist***, ***communitylist***: These tags display ordered sequences of items, collections and communities, showing minimal information but including a link to the page containing full details. These need to be used in HTML tables.
- ***popup***: This tag is used to render a link to a pop-up page (typically a help page.) If Javascript is available, the link will either open or pop to the front any existing DSpace pop-up window. If Javascript is not available, a standard HTML link is displayed that renders the link destination in a window named ' *dspace.popup*'. In graphical browsers, this usually opens a new window or re-uses an existing window of that name, but if a window is re-used it is not 'raised' which might confuse the user. In text browsers, following this link will simply replace the current page with the destination of the link. This obviously means that Javascript offers the best functionality, but other browsers are still supported.
- ***selecteperson***: A tag which produces a widget analogous to HTML <*SELECT*>, that allows a user to select one or multiple e-people from a pop-up list.
- ***sfxlink***: Using an item's Dublin Core metadata DSpace can display an SFX link, if an SFX server is available. This tag does so for a particular item if the *sfx.server.url* property is defined in *dspace.cfg*.

Internationalization (JSPUI Only)

> **XMLUI Internationalization**
>
> For information about XMLUI Internationalization please see: XMLUI Multilingual Support (see page 332).

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 belov:

```

```

```
<fmt:message key="jsp.submit.show-uploaded-file.size-in-bytes">
  <fmt:param><fmt:formatNumber><%= bitstream.getSize()%></fmt:formatNumber></fmt:param>
</fmt:message>
```

(Note: JSTL offers a way to include numbers in the message keys as *jsp.foo.key = {0,number} bytes*. Setting the parameter as *<fmt:param value="${variable}" />* workes when *variable* is a single variable name and doesn't work when trying to use a method's return value instead: *bitstream.getSize()*. Passing the number as string (or using the <%= %> expression) also does not work.)

Multiple *Messages.properties* can be created for different languages. See ResourceBundle.getBundle. e.g. you can add German and Canadian French translations:

```
Messages_de.properties
Messages_fr_CA.properties
```

The end user's browser settings determine which language is used. The English language file *Messages.properties* (or the default server locale) will be used as a default if there's no language bundle for the end user's preferred language. (Note that the English file is not called *Messages_en.properties* – this is so it is always available as a default, regardless of server configuration.)

The *dspace:layout* tag has been updated to allow dictionary keys to be passed in for the titles. It now has two new parameters: *titlekey* and *parenttitlekey*. So where before you'd do:

```
<dspace:layout title="Here"
               parentlink="/mydspace"
               parenttitle="My DSpace">
```

You now do:

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

And so the layout tag itself gets the relevant stuff out of the dictionary. *title* and *parenttitle* still work as before for backwards compatibility, and the odd spot where that's preferable.

Message Key Convention

When translating further pages, please follow the convention for naming message keys to avoid clashes.

For text in JSPs use the complete path + filename of the JSP, then a one-word name for the message. e.g. for the title of *jsp/mydspace/main.jsp* use:

```
jsp.mydspace.main.title
```

Some common words (e.g. "Help") can be brought out into keys starting *jsp.* for ease of translation, e.g.:

```
jsp.admin = Administer
```

Other common words/phrases are brought out into 'general' parameters if they relate to a set (directory) of JSPs, e.g.

```
jsp.tools.general.delete = Delete
```

Phrases that relate **strongly** to a topic (eg. MyDSpace) but used in many JSPs outside the particular directory are more convenient to be cross-referenced. For example one could use the key below in *jsp/submit/saved.jsp* to provide a link back to the user's *MyDSpace*:

*(Cross-referencing of keys **in general** is not a good idea as it may make maintenance more difficult. But in some cases it has more advantages as the meaning is obvious.)*

```
jsp.mydspace.general.goto-mydspace = Go to My DSpace
```

For text in servlet code, in custom JSP tags or wherever applicable use the fully qualified classname + a one-word name for the message. e.g.

```
org.dspace.app.webui.jsptag.ItemListTag.title = Title
```

Which Languages are currently supported?

To view translations currently being developed, please refer to the i18n page 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.

12.2.2 OAI-PMH Data Provider

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 OAICat framework from OCLC.

The DSpace build process builds a Web application archive, *[dspace-source]/build/oai.war*), in much the same way as the Web UI build process described above. The only differences are that the JSPs are not included, and *[dspace-source]/etc/oai-web.xml* is used as the deployment descriptor. This 'webapp' is deployed to receive and respond to OAI-PMH requests via HTTP. Note that typically it should *not* be deployed on SSL (*https:* protocol). In a typical configuration, this is deployed at *oai*, for example:

```
http://dspace.myu.edu/oai/request?verb=Identify
```

The 'base URL' of this DSpace deployment would be:

```
http://dspace.myu.edu/oai/request
```

It is this URL that should be registered with www.openarchives.org. Note that you can easily change the '*request* ' portion of the URL by editing *[dspace-source]/etc/oai-web.xml* and rebuilding and deploying *oai.war*.

DSpace provides implementations of the OAICat interfaces *AbstractCatalog*, *RecordFactory* and *Crosswalk* that interface with the DSpace content management API and harvesting API (in the search subsystem).

Only the basic *oai_dc* unqualified Dublin Core metadata set export is enabled by default; this is particularly easy since all items have qualified Dublin Core metadata. When this metadata is harvested, the qualifiers are simply stripped; for example, *description.abstract* is exposed as unqualified *description*. The *description.provenance* field is hidden, as this contains private information about the submitter and workflow reviewers of the item, including their e-mail addresses. Additionally, to keep in line with OAI community practices, values of *contributor.author* are exposed as *creator* values.

Other metadata formats are supported as well, using other *Crosswalk* implementations; consult the *oaicat.properties* file described below. To enable a format, simply uncomment the lines beginning with *Crosswalks.**. Multiple formats are allowed, and the current list includes, in addition to unqualified DC: MPEG DIDL, METS, MODS. There is also an incomplete, experimental qualified DC.

Note that the current simple DC implementation (*org.dspace.app.oai.OAIDCCrosswalk*) does not currently strip out any invalid XML characters that may be lying around in the data. If your database contains a DC value with, for example, some ASCII control codes (form feed etc.) this may cause OAI harvesters problems. This should rarely occur, however. XML entities (such as >) are encoded (e.g. to >)

In addition to the implementations of the OAICat interfaces, there is one main configuration file relevant to OAI-PMH support:

- **oaicat.properties**: This file resides in `[dspace]/config`. You probably won't need to edit this, as it is pre-configured to meet most needs. You might want to change the `Identify.earliestDatestamp` field to more accurately reflect the oldest datestamp in your local DSpace system. (Note that this is the value of the `last_modified` column in the `Item` database table.)

Sets

OAI-PMH allows repositories to expose an hierarchy of sets in which records may be placed. A record can be in zero or more sets.

DSpace exposes collections as sets. The organization of communities is likely to change over time, and is therefore a less stable basis for selective harvesting.

Each collection has a corresponding OAI set, discoverable by harvesters via the ListSets verb. The setSpec is the Handle of the collection, with the ':' and '/' converted to underscores so that the Handle is a legal setSpec, for example:

```
hdl_1721.1_1234
```

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

Unique Identifier

Every item in OAI-PMH data repository must have an unique identifier, which must conform to the URI syntax. As of DSpace 1.2, Handles are not used; this is because in OAI-PMH, the OAI identifier identifies the *metadata record* associated with the *resource*. The *resource* is the DSpace item, whose *resource identifier* is the Handle. In practical terms, using the Handle for the OAI identifier may cause problems in the future if DSpace instances share items with the same Handles; the OAI metadata record identifiers should be different as the different DSpace instances would need to be harvested separately and may have different metadata for the item.

The OAI identifiers that DSpace uses are of the form:

```
oai:host name:handle
```

For example:

```
oai:dspace.myu.edu:123456789/345
```

If you wish to use a different scheme, this can easily be changed by editing the value of *OAI_ID_PREFIX* at the top of the `org.dspace.app.oai.DSpaceOAICatalog` class. (You do not need to change the code if the above scheme works for you; the code picks up the host name and Handles automatically from the DSpace configuration.)

Access control

OAI provides no authentication/authorisation details, although these could be implemented using standard HTTP methods. It is assumed that all access will be anonymous for the time being.

A question is, "is all metadata public?" Presently the answer to this is yes; all metadata is exposed via OAI-PMH, even if the item has restricted access policies. The reasoning behind this is that people who do actually have permission to read a restricted item should still be able to use OAI-based services to discover the content.

If in the future, this 'expose all metadata' approach proves unsatisfactory for any reason, it should be possible to expose only publicly readable metadata. The authorisation system has separate permissions for READing and item and READing the content (bitstreams) within it. This means the system can differentiate between an item with public metadata and hidden content, and an item with hidden metadata as well as hidden content. In this case the OAI data repository should only expose items those with anonymous READ access, so it can hide the existence of records to the outside world completely. In this scenario, one should be wary of protected items that are made public after a time. When this happens, the items are "new" from the OAI-PMH perspective.

Modification Date (OAI Date Stamp)

OAI-PMH harvesters need to know when a record has been created, changed or deleted. DSpace keeps track of a 'last modified' date for each item in the system, and this date is used for the OAI-PMH date stamp. This means that any changes to the metadata (e.g. admins correcting a field, or a withdrawal) will be exposed to harvesters.

'About' Information

As part of each record given out to a harvester, there is an optional, repeatable "about" section which can be filled out in any (XML-schema conformant) way. Common uses are for provenance and rights information, and there are schemas in use by OAI communities for this. Presently DSpace does not provide any of this information.

Deletions

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

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

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

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

Flow Control (Resumption Tokens)

An OAI data provider can prevent any performance impact caused by harvesting by forcing a harvester to receive data in time-separated chunks. If the data provider receives a request for a lot of data, it can send part of the data with a resumption token. The harvester can then return later with the resumption token and continue.

DSpace supports resumption tokens for 'ListRecords' OAI-PMH requests. ListIdentifiers and ListSets requests do not produce a particularly high load on the system, so resumption tokens are not used for those requests.

Each OAI-PMH ListRecords request will return at most 100 records. This limit is set at the top of *org.dspace.app.oai.DSpaceOAICatalog.java* (*MAX_RECORDS*). A potential issue here is that if a harvest yields an exact multiple of *MAX_RECORDS*, the last operation will result in a harvest with no records in it. It is unclear from the OAI-PMH specification if this is acceptable.

When a resumption token is issued, the optional *completeListSize* and *cursor* attributes are not included. OAICat sets the *expirationDate* of the resumption token to one hour after it was issued, though in fact since DSpace resumption tokens contain all the information required to continue a request they do not actually expire.

Resumption tokens contain all the state information required to continue a request. The format is:

```
from/until/setSpec/offset
```

from and *until* are the ISO 8601 dates passed in as part of the original request, and *setSpec* is also taken from the original request. *offset* is the number of records that have already been sent to the harvester. For example:

```
2003-01-01//hdl_1721_1_1234/300
```

This means the harvest is 'from'
2003-01-01, has no 'until' date, is for collection hdl:1721.1/1234, and 300 records have already been sent to the harvester. (Actually, if the original OAI-PMH request doesn't specify a 'from' or 'until, OAICat fills them out automatically to '0000-00-00T00:00:00Z' and '9999-12-31T23:59:59Z' respectively. This means DSpace resumption tokens will always have from and until dates in them.)

12.2.3 DSpace Command Launcher

Introduced in Release 1.6, the DSpace Command Launcher brings together the various command and scripts into a standard-practice for running CLI runtime programs.

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>
```

  .

12.3 Business Logic Layer

12.3.1 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*.

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	*2002-11-11 08:11:32,903*

Level (*FATAL*, *WARN*, *INFO* or *DEBUG*)	*INFO*
Java class	*org.dspace.app.webui.servlet.DSpaceServlet*
	@
User email or *anonymous*	*anonymous*
	:
Extra log info from context	*session_id=BD84E7C194C2CF4BD0EC3A6CAD0142BB*
	:
Action	*view_item*
	:
Extra info	*handle=1721.1/1686*

The above format allows the logs to be easily parsed and 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 method that are required in a variety of places throughout the code, and thus have no particular 'home' in a subsystem.

12.3.2 Content Management API

The content management API package *org.dspace.content* contains Java classes for reading and manipulating content stored in the DSpace system. This is the API that components in the application layer will probably use most.

Classes corresponding to the main elements in the DSpace data model (*Community*, *Collection*, *Item*, *Bundle* and *Bitstream*) are sub-classes of the abstract class *DSpaceObject*. The *Item* object handles the Dublin Core metadata record.

Each class generally has one or more static *find* methods, which are used to instantiate content objects. Constructors do not have public access and are just used internally. The reasons for this are:

- "Constructing" an object may be misconstrued as the action of creating an object in the DSpace system, for example one might expect something like:

```
Context dsContent = new Context();
Item myItem = new Item(context, id)
```

to construct a brand new item in the system, rather than simply instantiating an in-memory instance of an object in the system.

- *find* methods may often be called with invalid IDs, and return *null* in such a case. A constructor would have to throw an exception in this case. A *null* return value from a static method can in general be dealt with more simply in code.
- If an instantiation representing the same underlying archival entity already exists, the *find* method can simply return that same instantiation to avoid multiple copies and any inconsistencies which might result.

Collection, *Bundle* and *Bitstream* do not have *create* methods; rather, one has to create an object using the relevant method on the container. For example, to create a collection, one must invoke *createCollection* on the community that the collection is to appear in:

```
Context context = new Context();
Community existingCommunity = Community.find(context, 123);
Collection myNewCollection = existingCommunity.createCollection();
```

The primary reason for this is for determining authorization. In order to know whether an e-person may create an object, the system must know which container the object is to be added to. It makes no sense to create a collection outside of a community, and the authorization system does not have a policy for that.

Item_s are first created in the form of an implementation of _InProgressSubmission. An *InProgressSubmission* represents an item under construction; once it is complete, it is installed into the main archive and added to the relevant collection by the *InstallItem* class. The *org.dspace.content* package provides an implementation of *InProgressSubmission* called *WorkspaceItem*; this is a simple implementation that contains some fields used by the Web submission UI. The *org.dspace.workflow* also contains an implementation called *WorkflowItem* which represents a submission undergoing a workflow process.

In the previous chapter there is an overview of the item ingest process which should clarify the previous paragraph. Also see the section on the workflow system.

Community and *BitstreamFormat* do have static *create* methods; one must be a site administrator to have authorization to invoke these.

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:

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

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

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

`Context context = new Context();`
`Bitstream bs = Bitstream.find(context, 1234);`
`Bundle bnd = Bundle.find(context, 5678);`
`bnd.add(bs);`
`context.complete();`
	The bitstream **will** be included in the bundle, since *update* doesn't need to be called

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 *DCValue* class is a simple container that represents a single Dublin Core element, optional qualifier, value and language. Note that since DSpace 1.4 the *MetadataValue* and associated classes are preferred (see

Support for Other Metadata Schemas). The other classes starting with *DC* are utility classes for handling types of data in Dublin Core, such as people's names and dates. As supplied, the DSpace registry of elements and qualifiers corresponds to the Library Application Profile for Dublin Core. It should be noted that these utility classes assume that the values will be in a certain syntax, which will be true for all data generated within the DSpace system, but since Dublin Core does not always define strict syntax, this may not be true for Dublin Core originating outside DSpace.

Below is the specific syntax that DSpace expects various fields to adhere to:

Element	Qualifier	Syntax	Helper Class
date	Any or unqualified	ISO 8601 in the UTC time zone, with either year, month, day, or second precision. Examples:_2000 2002-10 2002-08-14 1999-01-01T14:35:23Z _	*DCDate*
contributor	Any or unqualified	In general last name, then a comma, then first names, then any additional information like "Jr.". If the contributor is an organization, then simply the name. Examples:_Doe, John Smith, John Jr. van Dyke, Dick Massachusetts Institute of Technology _	*DCPersonName*
language	*iso*	A two letter code taken ISO 639, followed optionally by a two letter country code taken from ISO 3166. Examples:_en fr en_US _	*DCLanguage*
relation	*ispartofseries*	The series name, following by a semicolon followed by the number in that series. Alternatively, just free text._MIT-TR; 1234 My Report Series; ABC-1234 NS1234 _	*DCSeriesNumber*

Support for Other Metadata Schemas

To support additional metadata schemas a new set of metadata classes have been added. These are backwards compatible with the DC classes and should be used rather than the DC specific classes wherever possible. Note that hierarchical metadata schemas are not currently supported, only flat schemas (such as DC) are able to be defined.

The *MetadataField* class describes a metadata field by schema, element and optional qualifier. The value of a *MetadataField* is described by a *MetadataValue* which is roughly equivalent to the older *DCValue* class. Finally the *MetadataSchema* class is used to describe supported schemas. The DC schema is supported by default. Refer to the javadoc for method details.

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);
```

12.3.3 Plugin Manager

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

An example helps clarify the point: There is a named plugin that does crosswalks, call it *CrosswalkPlugin*. It has several implementations that crosswalk some kind of metadata. Now we add a new plugin which uses XSL stylesheet transformation (XSLT) to crosswalk many types of metadata – so the single plugin can act like many different plugins, depending on which stylesheet it employs.

This XSLT-crosswalk plugin has its own configuration that maps a Plugin Name to a stylesheet – it has to, since of course the Plugin Manager doesn't know anything about stylesheets. It becomes a self-named plugin, so that it reads its configuration data, gets the list of names to which it can respond, and passes those on to the Plugin Manager.

When the Plugin Manager creates an instance of the XSLT-crosswalk, it records the Plugin Name that was responsible for that instance. The plugin can look at that Name later in order to configure itself correctly for the Name that created it. This mechanism is all part of the SelfNamedPlugin class which is part of any self-named plugin.

Obtaining a Plugin Instance

The most common thing you will do with the Plugin Manager is obtain an instance of a plugin. To request a plugin, you must always specify the plugin interface you want. You will also supply a name when asking for a named plugin.

A sequence plugin is returned as an array of _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 intface)
       throws PluginConfigurationError;
  ```

  Returns an instance of the singleton (single) plugin implementing the given interface. There must be exactly one single plugin configured for this interface, otherwise the *PluginConfigurationError* is thrown. Note that this is the only "get plugin" method which throws an exception. It is typically used at initialization time to set up a permanent part of the system so any failure is fatal. See the *plugin.single* configuration key for configuration details.

- ```
  static Object[] getPluginSequence(Class intface);
  ```

  Returns instances of all plugins that implement the interface *intface*, in an *Array*. Returns an empty array if no there are no matching plugins. The order of the plugins in the array is the same as their class names in the configuration's value field. See the *plugin.sequence* configuration key for configuration details.

- ```
  static Object getNamedPlugin(Class intface, String name);
  ```
  ```

Returns an instance of a plugin that implements the interface *intface* and is bound to a name matching name. If there is no matching plugin, it returns null. The names are matched by *String.equals()*. See the *plugin.named* and *plugin.selfnamed* configuration keys for configuration details.

```
static void releasePlugin(Object plugin);
```

Tells the Plugin Manager to let go of any references to a reusable plugin, to prevent it from being given out again and to allow the object to be garbage-collected. Call this when a plugin instance must be taken out of circulation.

```
static String[] getAllPluginNames(Class intface);
```

Returns all of the names under which a named plugin implementing the interface *intface* can be requested (with *getNamedPlugin()*). The array is empty if there are no matches. Use this to populate a menu of plugins for interactive selection, or to document what the possible choices are. The names are NOT returned in any predictable order, so you may wish to sort them first. Note: Since a plugin may be bound to more than one name, the list of names this returns does not represent the list of plugins. To get the list of unique implementation classes corresponding to the names, you might have to eliminate duplicates (i.e. create a Set of classes).

```
static void checkConfiguration();
```

Validates the keys in the DSpace *ConfigurationManager* pertaining to the Plugin Manager and reports any errors by logging them. This is intended to be used interactively by a DSpace administrator, to check the configuration file after modifying it. See the section about validating configuration for details.

SelfNamedPlugin Class

A named plugin implementation must extend this class if it wants to supply its own Plugin Name(s). See Self-Named Plugins for why this is sometimes necessary.

```
abstract class SelfNamedPlugin
{
    // Your class must override this:
    // Return all names by which this plugin should be known.
    public static String[] getPluginNames();

    // Returns the name under which this instance was created.
    // This is implemented by SelfNamedPlugin and should NOT be
 overridden.
    public String getPluginInstanceName();
}
```

Errors and Exceptions

```
public class PluginConfigurationError extends Error
{
    public PluginConfigurationError(String message);
}
```

An error of this type means the caller asked for a single plugin, but either there was no single plugin configured matching that interface, or there was more than one. Either case causes a fatal configuration error.

```
public class PluginInstantiationException extends RuntimeException
{
    public PluginInstantiationException(String msg, Throwable cause)
}
```

This exception indicates a fatal error when instantiating a plugin class. It should only be thrown when something unexpected happens in the course of instantiating a plugin, e.g. an access error, class not found, etc. Simply not finding a class in the configuration is not an exception.

This is a *RuntimeException* so it doesn't have to be declared, and can be passed all the way up to a generalized fatal exception handler.

Configuring Plugins

All of the Plugin Manager's configuration comes from the DSpace Configuration Manager, which is a Java Properties map. You can configure these characteristics of each plugin:

1. **Interface**: Classname of the Java interface which defines the plugin, including package name. e.g. *org.dspace.app.mediafilter.FormatFilter*
2. **Implementation Class**: Classname of the implementation class, including package. e.g. *org.dspace.app.mediafilter.PDFFilter*
3. **Names**: (Named plugins only) There are two ways to bind names to plugins: listing them in the value of a plugin.named.interface key, or configuring a class in *plugin.selfnamed.interface* which extends the *SelfNamedPlugin* class.
4. **Reusable option**: (Optional) This is declared in a *plugin.reusable* configuration line. Plugins are reusable by default, so you only need to configure the non-reusable ones.

Configuring Singleton (Single) Plugins

This entry configures a Single Plugin for use with getSinglePlugin():

```
plugin.single.interface = classname
```

For example, this configures the class *org.dspace.checker.SimpleDispatcher* as the plugin for interface *org.dspace.checker.BitstreamDispatcher*:

```
plugin.single.org.dspace.checker.BitstreamDispatcher=org.dspace.checker.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.xsl
crosswalk.dissemination.stylesheet.MODS = xwalk/mods.xsl

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.

12.3.4 Workflow System

The primary classes are:

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

The workflow system models the states of an Item in a state machine with 5 states (SUBMIT, STEP_1, STEP_2, STEP_3, ARCHIVE.) These are the three optional steps where the item can be viewed and corrected by different groups of people. Actually, it's more like 8 states, with STEP_1_POOL, STEP_2_POOL, and STEP_3_POOL. These pooled states are when items are waiting to enter the primary states.

The WorkflowManager is invoked by events. While an Item is being submitted, it is held by a WorkspaceItem. Calling the start() method in the WorkflowManager converts a WorkspaceItem to a WorkflowItem, and begins processing the WorkflowItem's state. Since all three steps of the workflow are optional, if no steps are defined, then the Item is simply archived.

Workflows are set per Collection, and steps are defined by creating corresponding entries in the List named workflowGroup. If you wish the workflow to have a step 1, use the administration tools for Collections to create a workflow Group with members who you want to be able to view and approve the Item, and the workflowGroup[0] becomes set with the ID of that Group.

If a step is defined in a Collection's workflow, then the WorkflowItem's state is set to that step_POOL. This pooled state is the WorkflowItem waiting for an EPerson in that group to claim the step's task for that WorkflowItem. The WorkflowManager emails the members of that Group notifying them that there is a task to be performed (the text is defined in config/emails,) and when an EPerson goes to their 'My DSpace' page to claim

the task, the WorkflowManager is invoked with a claim event, and the WorkflowItem's state advances from STEP_x_POOL to STEP_x (where x is the corresponding step.) The EPerson can also generate an 'unclaim' event, returning the WorkflowItem to the STEP_x_POOL.

Other events the WorkflowManager handles are advance(), which advances the WorkflowItem to the next state. If there are no further states, then the WorkflowItem is removed, and the Item is then archived. An EPerson performing one of the tasks can reject the Item, which stops the workflow, rebuilds the WorkspaceItem for it and sends a rejection note to the submitter. More drastically, an abort() event is generated by the admin tools to cancel a workflow outright.

12.3.5 Administration Toolkit

The *org.dspace.administer* package contains some classes for administering a DSpace system that are not generally needed by most applications.

The *CreateAdministrator* class is a simple command-line tool, executed via `[dspace]/bin/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.DCValue* 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.

12.3.6 E-person/Group Manager

DSpace keeps track of registered users with the *org.dspace.eperson.EPerson* class. The class has methods to create and manipulate an *EPerson* such as get and set methods for first and last names, email, and password. (Actually, there is no *getPassword()* method, an MD5 hash of the password is stored, and can only be verified with the *checkPassword()* method.) There are find methods to find an EPerson by email (which is assumed to be unique,) or to find all EPeople in the system.

The *EPerson* object should probably be reworked to allow for easy expansion; the current EPerson object tracks pretty much only what MIT was interested in tracking - first and last names, email, phone. The access methods are hardcoded and should probably be replaced with methods to access arbitrary name/value pairs for institutions that wish to customize what EPerson information is stored.

Groups are simply lists of *EPerson* objects. Other than membership, *Group* objects have only one other attribute: a name. Group names must be unique, so we have adopted naming conventions where the role of the group is its name, such as *COLLECTION_100_ADD*. Groups add and remove EPerson objects with *addMember()* and *removeMember()* methods. One important thing to know about groups is that they store their membership in memory until the *update()* method is called - so when modifying a group's membership don't forget to invoke *update()* or your changes will be lost! Since group membership is used heavily by the authorization system a fast *isMember()* method is also provided.

Another kind of Group is also implemented in DSpace, special Groups. The *Context* object for each session carries around a List of Group IDs that the user is also a member of, currently the MITUser Group ID is added to the list of a user's special groups if certain IP address or certificate criteria are met.

12.3.7 Authorization

The primary classes are:

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

The authorization system is based on the classic 'police state' model of security; no action is allowed unless it is expressed in a policy. The policies are attached to resources (hence the name *ResourcePolicy*,) and detail who can perform that action. The resource can be any of the DSpace object types, listed in *org.dspace.core.Constants* (*BITSTREAM*, *ITEM*, *COLLECTION*, etc.) The 'who' is made up of EPerson groups. The actions are also in *Constants.java* (*READ*, *WRITE*, *ADD*, etc.) The only non-obvious actions are *ADD* and *REMOVE*, which are authorizations for container objects. To be able to create an Item, you must have *ADD* permission in a Collection, which contains Items. (Communities, Collections, Items, and Bundles are all container objects.)

Currently most of the read policy checking is done with items, communities and collections are assumed to be openly readable, but items and their bitstreams are checked. Separate policy checks for items and their bitstreams enables policies that allow publicly readable items, but parts of their content may be restricted to certain groups.

The *AuthorizeManager* class'
authorizeAction(Context, object, action) is the primary source of all authorization in the system. It gets a list of all of the ResourcePolicies in the system that match the object and action. It then iterates through the policies,

extracting the EPerson Group from each policy, and checks to see if the EPersonID from the Context is a member of any of those groups. If all of the policies are queried and no permission is found, then an *AuthorizeException* is thrown. An *authorizeAction()* method is also supplied that returns a boolean for applications that require higher performance.

ResourcePolicies are very simple, and there are quite a lot of them. Each can only list a single group, a single action, and a single object. So each object will likely have several policies, and if multiple groups share permissions for actions on an object, each group will get its own policy. (It's a good thing they're small.)

Special Groups

All users are assumed to be part of the public group (ID=0.) DSpace admins (ID=1) are automatically part of all groups, much like super-users in the Unix OS. The Context object also carries around a List of special groups, which are also first checked for membership. These special groups are used at MIT to indicate membership in the MIT community, something that is very difficult to enumerate in the database! When a user logs in with an MIT certificate or with an MIT IP address, the login code adds this MIT user group to the user's Context.

Miscellaneous Authorization Notes

Where do items get their read policies? From the 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.

12.3.8 Handle Manager/Handle Plugin

The *org.dspace.handle* package contains two classes; *HandleManager* is used to create and look up Handles, and *HandlePlugin* is used to expose and resolve DSpace Handles for the outside world via the CNRI Handle Server code.

Handles are stored internally in the *handle* database table in the form:

1721.123/4567

Typically when they are used outside of the system they are displayed in either URI or "URL proxy" forms:

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

It is the responsibility of the caller to extract the basic form from whichever displayed form is used.

The *handle* table maps these Handles to resource type/resource ID pairs, where resource type is a value from *org.dspace.core.Constants* and resource ID is the internal identifier (database primary key) of the object. This allows Handles to be assigned to any type of object in the system, though as explained in the functional overview, only communities, collections and items are presently assigned Handles.

HandleManager contains static methods for:

- Creating a Handle
- Finding the Handle for a *DSpaceObject*, though this is usually only invoked by the object itself, since *DSpaceObject* has a *getHandle* method
- Retrieving the *DSpaceObject* identified by a particular Handle
- Obtaining displayable forms of the Handle (URI or "proxy URL").
  HandlePlugin is a simple implementation of the Handle Server's *net.handle.hdllib.HandleStorage* interface. It only implements the basic Handle retrieval methods, which get information from the *handle* database table. The CNRI Handle Server is configured to use this plug-in via its *config.dct* file.

Note that since the Handle server runs as a separate JVM to the DSpace Web applications, it uses a separate 'Log4J' configuration, since Log4J does not support multiple JVMs using the same daily rolling logs. This alternative configuration is 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.

12.3.9 Search

DSpace's search code is a simple API which currently wraps the Lucene search engine. The first half of the search task is indexing, and *org.dspace.search.DSIndexer* is the indexing class, which contains *indexContent()* which if passed an *Item*, *Community*, or *Collection*, will add that content's fields to the index. The methods *unIndexContent()* and *reIndexContent()* remove and update content's index information. The *DSIndexer* class also has a *main()* method which will rebuild the index completely. This can be invoked by the *dspace/bin/index-init* (complete rebuild) or *dspace/bin/index-update* (update) script. The intent was for the *main()* method to be invoked on a regular basis to avoid index corruption, but we have had no problem with that so far.

Which fields are indexed by *DSIndexer*? These fields are defined in dspace.cfg in the section "Fields to index for search" as name-value-pairs. The name must be unique in the form search.index.i (i is an arbitrary positive number). The value on the right side has a unique value again, which can be referenced in search-form (e.g. title, author). Then comes the metadata element which is indexed. '*' is a wildcard which includes all 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	*contributor.**creator.**description.statementofresponsibility*
Titles	*title.**
Keywords	*subject.**
Abstracts	*description.abstractdescription.tableofcontents*
Series	*relation.ispartofseries*
MIME types	*format.mimetype*
Sponsors	*description.sponsorship*
Identifiers	*identifier.**

Harvesting API

The *org.dspace.search* package also provides a 'harvesting' API. This allows callers to extract information about items modified within a particular timeframe, and within a particular scope (all of DSpace, or a community or collection.) Currently this is used by the Open Archives Initiative metadata harvesting protocol application, and the e-mail subscription code.

The *Harvest.harvest* is invoked with the required scope and start and end dates. Either date can be omitted. The dates should be in the ISO8601, UTC time zone format used elsewhere in the DSpace system.

HarvestedItemInfo objects are returned. These objects are simple containers with basic information about the items falling within the given scope and date range. Depending on parameters passed to the *harvest* method, the *containers* and *item* fields may have been filled out with the IDs of communities and collections containing an item, and the corresponding *Item* object respectively. Electing not to have these fields filled out means the harvest operation executes considerable faster.

In case it is required, *Harvest* also offers a method for creating a single *HarvestedItemInfo* object, which might make things easier for the caller.

12.3.10 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 save issue date. Say 30 items in a collection have the issue date 2002. To be able to page through the index 20 items at a time, you need to be able to specify exactly which item's 2002 is the focus of the browse, otherwise each time you invoked the browse code, the results would start at the first item with the issue date 2002.

Author browses return *String* objects with the actual author names. You can only specify the focus as a full or partial literal *String*.

Another important point to note is that presently, the browse 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.

12.3.11 Checksum checker

Checksum checker is used to verify every item within DSpace. While DSpace calculates and records the checksum of every file submitted to it, the checker can determine whether the file has been changed. The idea being that the earlier you can identify a file has changed, the more likely you would be able to record it (assuming it was not a wanted change).

`org.dspace.checker.CheckerCommand` class, is the class for the checksum checker tool, which calculates checksums for each bitstream whose ID is in the *most_recent_checksum* table, and compares it against the last calculated checksum for that bitstream.

12.3.12 OpenSearch Support

DSpace is able to support OpenSearch. For those not acquainted with the standard, a very brief introduction, with emphasis on what possibilities it holds for current use and future development.

OpenSearch is a small set of conventions and documents for describing and using 'search engines', meaning any service that returns a set of results for a query. It is nearly ubiquitous, but also nearly invisible, in modern web sites with search capability. If you look at the page source of Wikipedia, Facebook, CNN, etc you will find buried a link element declaring OpenSearch support. It is very much a lowest-common-denominator abstraction (think Google box), but does provide a means to extend its expressive power. This first implementation for DSpace supports *none* of these extensions, many of which are of potential value, so it should be regarded as a foundation, not a finished solution. So the short answer is that DSpace appears as a 'search-engine' to OpenSearch-aware software.

Another way to look at OpenSearch is as a RESTful web service for search, very much like SRW/U, but considerably simpler. This comparative loss of power is offset by the fact that it is widely supported by web tools and players: browsers understand it, as do large metasearch tools.

How Can It Be Used

- Browser IntegrationMany 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 FeedsBecause 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 metasearchSearch aggregators like A9 (Amazon) recognize OpenSearch-compliant providers, and so can be added to metasearch sets using their UIs. Then you site can be used to aggregate search results with others.

Configuration is through the `dspace.cfg` file. See OpenSearch Support (see page 205) for more details.

12.3.13 Embargo Support

What is an Embargo?

An embargo is a temporary access restriction placed on content, commencing at time of accession. It's 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 functionally 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 it's 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.

12.4 DSpace Services Framework

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).

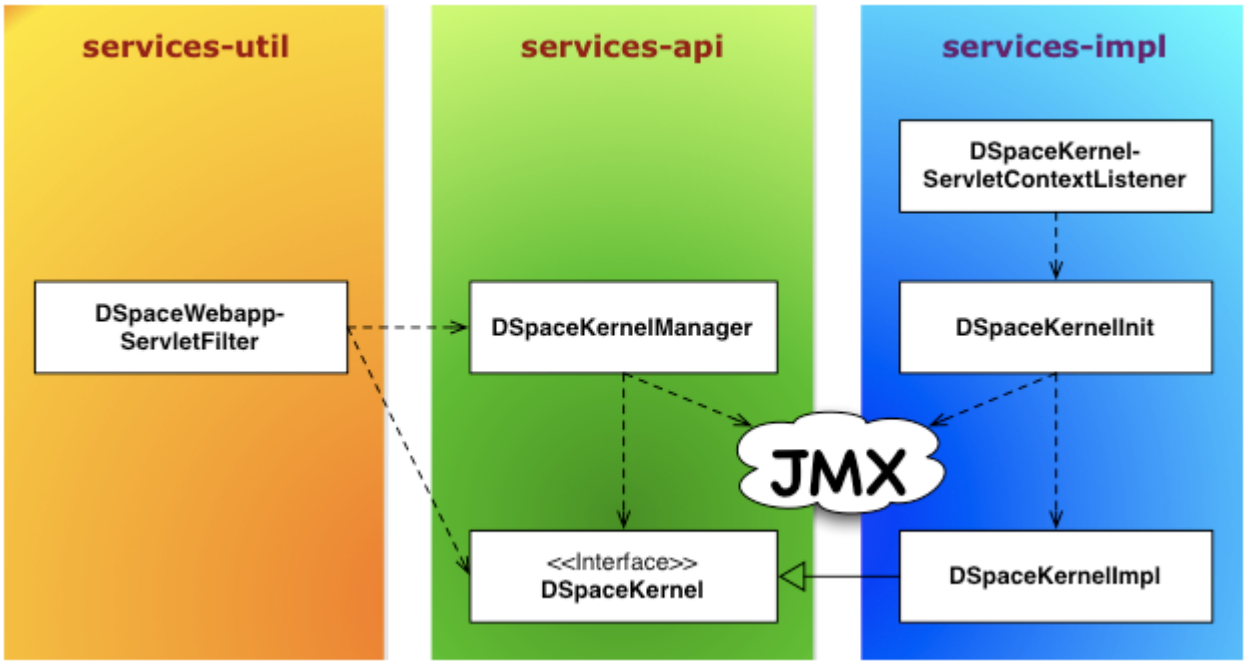
12.4.1 Architectural Overview

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.

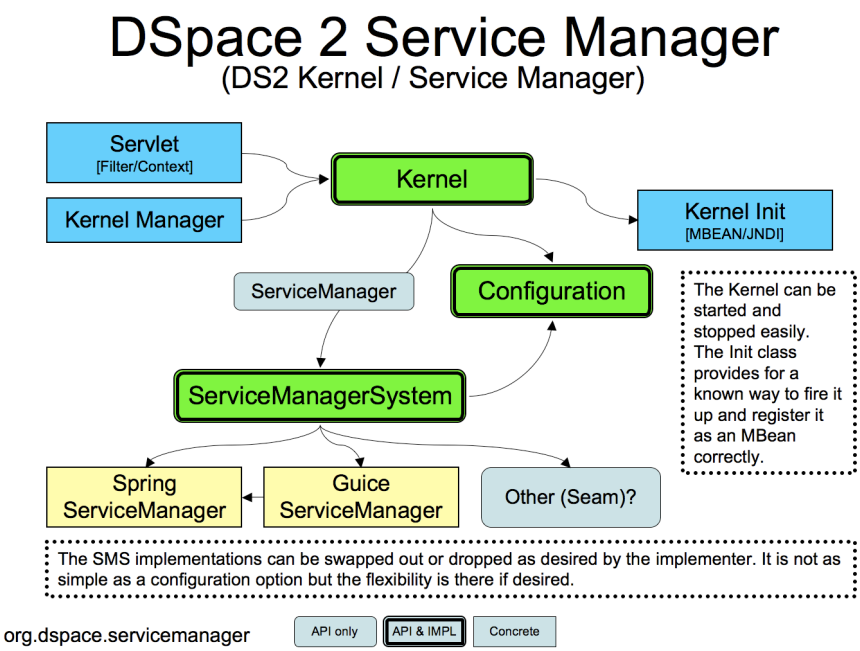
Kernel registration

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.



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.

12.4.2 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 (see page), 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.

12.4.3 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.

12.4.4 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.

12.4.5 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.)

12.4.6 Tutorials

Several tutorials on Spring / DSpace Services are available:

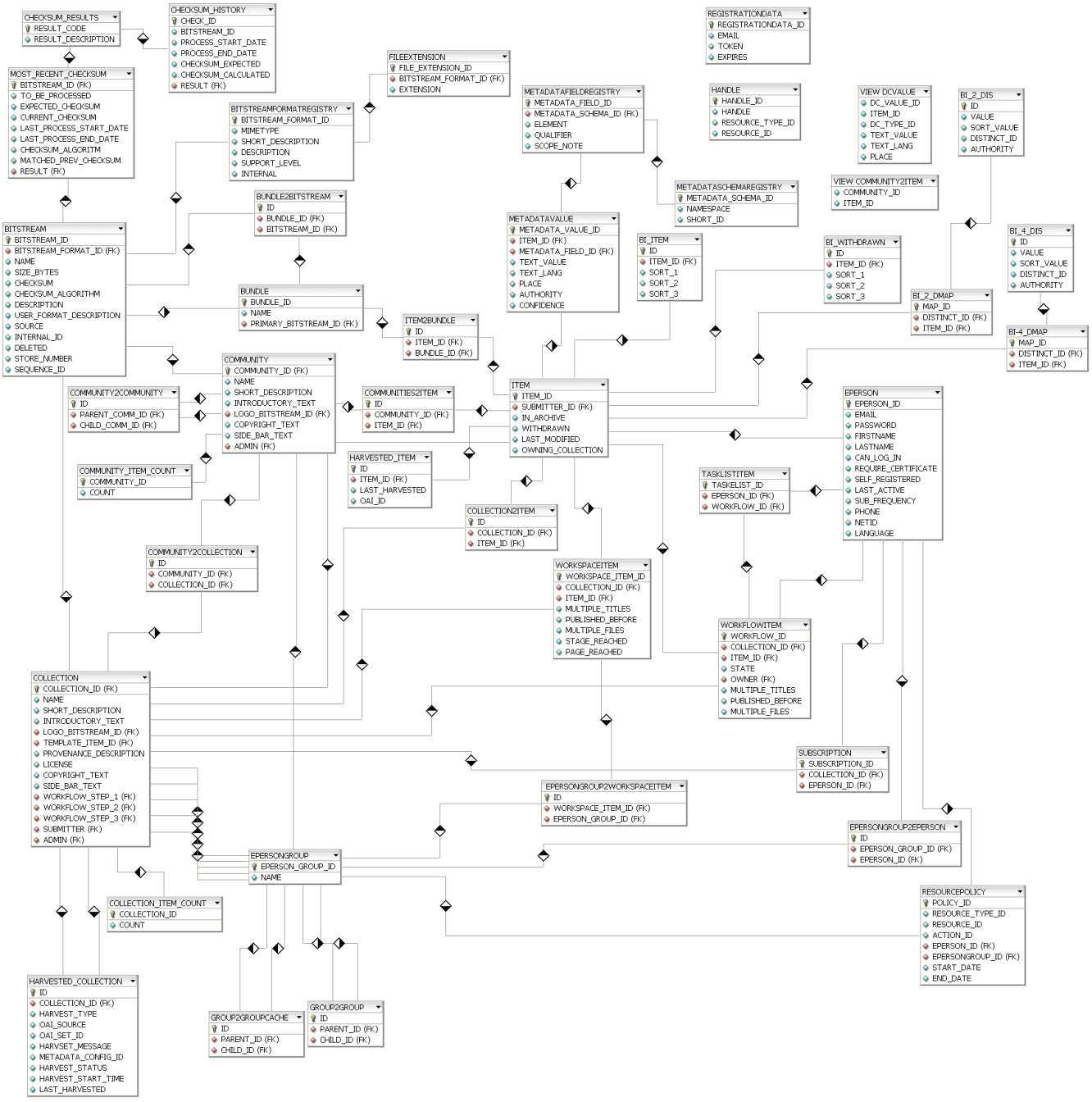
- DSpace Spring Services Tutorial
- The TAO of DSpace Services

12.5 Storage Layer

In this section, we explain the storage layer: the database structure, maintenance, and the bistream store and configurations.

12.5.1 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.



Most of the functionality that DSpace uses can be offered by any standard SQL database that supports transactions. Presently, the browse indices use some features specific to PostgreSQL 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 main class is *DatabaseManager*, which executes SQL queries and returns *TableRow* or *TableRowIterator* objects. The *InitializeDatabase* class is used to load SQL into the database via JDBC, for example to set up the schema.

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

The database schema used by DSpace is created by SQL statements stored in a directory specific to each supported RDBMS platform:

- PostgreSQL schemas are in *[dspace-source]/dspace/etc/postgres/*
- Oracle schemas are in *[dspace-source]/dspace/etc/oracle/*
  The SQL (DDL) statements to create the tables for the current release, starting with an empty database, aer in *database_schema.sql*. The schema SQL file also creates the two required e-person groups (*Anonymous* and *Administrator*) that are required for the system to function properly.

Also in *[dspace-source]/dspace/etc/[database]* are various SQL files called *database_schema_1x_1y*. These contain the necessary SQL commands to update a live DSpace database from version 1.*x* to 1.*y*. Note that this might not be the only part of an upgrade process: see Updating a DSpace Installation for details.

The DSpace database code uses an SQL function *getnextid* to assign primary keys to newly created rows. This SQL function must be safe to use if several JVMs are accessing the database at once; for example, the Web UI might be creating new rows in the database at the same time as the batch item importer. The PostgreSQL-specific implementation of the method uses *SEQUENCES* for each table in order to create new IDs. If an alternative database backend were to be used, the implementation of *getnextid* could be updated to operate with that specific DBMS.

The *etc* directory in the source distribution contains two further SQL files. *clean-database.sql* contains the SQL necessary to completely clean out the database, so use with caution! The Ant target *clean_database* can be used to execute this. *update-sequences.sql* contains SQL to reset the primary key generation sequences to appropriate values. You'd need to do this if, for example, you're restoring a backup database dump which creates rows with specific primary keys already defined. In such a case, the sequences would allocate primary keys that were already used.

Versions of the ***.sql*** files for Oracle are stored in *[dspace-source]/dspace/etc/oracle*. These need to be copied over their PostgreSQL counterparts in *[dspace-source]/dspace/etc* prior to installation.

Maintenance and Backup

When using PostgreSQL, it's a good idea to perform regular 'vacuuming' of the database to optimize performance. This is performed by the *vacuumdb* command which can be executed via a 'cron' job, for example by putting this in the system *crontab*:

```
# clean up the database nightly
40 2 * * * /usr/local/pgsql/bin/vacuumdb --analyze dspace > /dev/null 2>&1
```

The DSpace database can be backed up and restored using usual methods, for example with *pg_dump* and *psql*. However when restoring a database, you will need to perform these additional steps:

- The *fresh_install* target loads up the initial contents of the Dublin Core type and bitstream format registries, as well as two entries in the *epersongroup* table for the system anonymous and administrator groups. Before you restore a raw backup of your database you will need to remove these, since they will already exist in your backup, possibly having been modified. For example, use:

```
DELETE FROM dctyperegistry;
DELETE FROM bitstreamformatregistry;
DELETE FROM epersongroup;
```

- After restoring a backup, you will need to reset the primary key generation sequences so that they do not produce already-used primary keys. Do this by executing the SQL in *[dspace-source]/dspace/etc/update-sequences.sql*, for example with:

```
psql -U dspace -f  [dspace-source]/dspace/etc/update-sequences.sql
```

  Future updates of DSpace may involve minor changes to the database schema. Specific instructions on how to update the schema whilst keeping live data will be included. The current schema also contains a few currently unused database columns, to be used for extra functionality in future releases. These unused columns have been added in advance to minimize the effort required to upgrade.

Configuring the RDBMS Component

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

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

12.5.2 Bitstream Store

DSpace offers two means for storing content. The first is in the file system on the server. The second is using SRB (Storage Resource Broker). Both are achieved using a simple, lightweight API.

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

The terms "store", "retrieve", "in the system", "storage", and so forth, used below can refer to storage in the file system on the server ("traditional") or in SRB.

The *BitstreamStorageManager* provides low-level access to bitstreams stored in the system. In general, it should not be used directly; instead, use the *Bitstream* object in the content management API since that encapsulated authorization and other metadata to do with a bitstream that are not maintained by the *BitstreamStorageManager*.

The bitstream storage manager provides three methods that store, retrieve and delete bitstreams. Bitstreams are referred to by their 'ID'; that is the primary key *bitstream_id* column of the corresponding row in the database.

As of DSpace version 1.1, there can be multiple bitstream stores. Each of these bitstream stores can be traditional storage or SRB storage. This means that the potential storage of a DSpace system is not bound by the maximum size of a single disk or file system and also that traditional and SRB storage can be combined in one DSpace installation. Both traditional and SRB storage are specified by configuration parameters. Also see Configuring the Bitstream Store below.

Stores are numbered, starting with zero, then counting upwards. Each bitstream entry in the database has a store number, used to retrieve the bitstream when required.

At the moment, the store in which new bitstreams are placed is decided using a configuration parameter, and there is no provision for moving bitstreams between stores. Administrative tools for manipulating bitstreams and stores will be provided in future releases. Right now you can move a whole store (e.g. you could move store number 1 from */localdisk/store* to */fs/anotherdisk/store* but it would still have to be store number 1 and have the exact same contents.

Bitstreams also have an 38-digit internal ID, different from the primary key ID of the bitstream table row. This is not visible or used outside of the bitstream storage manager. It is used to determine the exact location (relative to the relevant store directory) that the bitstream is stored in traditional or SRB storage. The first three pairs of digits are the directory path that the bitstream is stored under. The bitstream is stored in a file with the internal ID as the filename.

For example, a bitstream with the internal ID *12345678901234567890123456789012345678* is stored in the directory:

```
(assetstore dir)/12/34/56/12345678901234567890123456789012345678
```

The reasons for storing files this way are:

- Using a randomly-generated 38-digit number means that the 'number space' is less cluttered than simply using the primary keys, which are allocated sequentially and are thus close together. This means that the bitstreams in the store are distributed around the directory structure, improving access efficiency.
- The internal ID is used as the filename partly to avoid requiring an extra lookup of the filename of the bitstream, and partly because bitstreams may be received from a variety of operating systems. The original name of a bitstream may be an illegal UNIX filename.
  When storing a bitstream, the *BitstreamStorageManager* DOES set the following fields in the corresponding database table row:

- *bitstream_id*
- *size*
- *checksum*
- *checksum_algorithm*
- *internal_id*
- *deleted*
- *store_number*
  The remaining fields are the responsibility of the *Bitstream* content management API class.

The bitstream storage manager is fully transaction-safe. In order to implement transaction-safety, the following algorithm is used to store bitstreams:

1. A database connection is created, separately from the currently active connection in the current DSpace context.
2. An unique internal identifier (separate from the database primary key) is generated.
3. The bitstream DB table row is created using this new connection, with the *deleted* column set to *true*.
4. The new connection is _commit_ted, so the 'deleted' bitstream row is written to the database
5. The bitstream itself is stored in a file in the configured 'asset store directory', with a directory path and filename derived from the internal ID
6. The *deleted* flag in the bitstream row is set to *false*. This will occur (or not) as part of the current DSpace *Context*.
   This means that should anything go wrong before, during or after the bitstream storage, only one of the following can be true:

- No bitstream table row was created, and no file was stored
- A bitstream table row with *deleted=true* was created, no file was stored
- A bitstream table row with *deleted=true* was created, and a file was stored
  None of these affect the integrity of the data in the database or bitstream store.

Similarly, when a bitstream is deleted for some reason, its *deleted* flag is set to true as part of the overall transaction, and the corresponding file in storage is *not* deleted.

The above techniques mean that the bitstream storage manager is transaction-safe. Over time, the bitstream database table and file store may contain a number of 'deleted' bitstreams. The *cleanup* method of *BitstreamStorageManager* goes through these deleted rows, and actually deletes them along with any

corresponding files left in the storage. It only removes 'deleted' bitstreams that are more than one hour old, just in case cleanup is happening in the middle of a storage operation.

This cleanup can be invoked from the command line via the *Cleanup* class, which can in turn be easily executed from a shell on the server machine using */dspace/bin/cleanup*. You might like to have this run regularly by *cron*, though since DSpace is read-lots, write-not-so-much it doesn't need to be run very often.

Backup

The bitstreams (files) in traditional storage may be backed up very easily by simply 'tarring' or 'zipping' the *assetstore* directory (or whichever directory is configured in *dspace.cfg*). Restoring is as simple as extracting the backed-up compressed file in the appropriate location.

Similar means could be used for SRB, but SRB offers many more options for managing backup.

It is important to note that since the bitstream storage manager holds the bitstreams in storage, and information about them in the database, that a database backup and a backup of the files in the bitstream store must be made at the same time; the bitstream data in the database must correspond to the stored files.

Of course, it isn't really ideal to 'freeze' the system while backing up to ensure that the database and files match up. Since DSpace uses the bitstream data in the database as the authoritative record, it's best to back up the database before the files. This is because it's better to have a bitstream in storage but not the database (effectively non-existent to DSpace) than a bitstream record in the database but not storage, since people would be able to find the bitstream but not actually get the contents.

With DSpace 1.7 and above, there is also the option to backup both files and metadata via the AIP Backup and Restore (see page 347) 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. The there will be a set of SRB account parameters like this:

```
srb.host.1 = mysrbmcathost.myu.edu
srb.port.1 = 5544
srb.mcatzone.1 = mysrbzone
srb.mdasdomainname.1 = mysrbdomain
srb.defaultstorageresource.1 = mydefaultsrbresource
srb.username.1 = mysrbuser
srb.password.1 = mysrbpassword
srb.homedirectory.1 = /mysrbzone/home/mysrbuser.mysrbdomain
srb.parentdir.1 = mysrbdspaceassetstore
```

Several of the terms, such as *mcatzone*, have meaning only in the SRB context and will be familiar to SRB users. The last, *srb.parentdir.n*, can be used to used for addition (SRB) upper directory structure within an SRB account. This property value could be blank as well.

(If asset store 0 would refer to SRB it would be *srb.host* = ..., *srb.port* = ..., and so on (*.0* omitted) to be consistent with the traditional storage configuration above.)

The similar use of *assetstore.incoming* to reference asset store 0 (default) or 1..n (explicit property) means that new bitstreams will be written to traditional or SRB storage determined by whether a file system directory on the server is referenced or a set of SRB account parameters are referenced.

There are comments in dspace.cfg that further elaborate the configuration of traditional and SRB storage.

13 Submission User Interface

This page explains various customization and configuration options that are available within DSpace for the Item Submission user interface.

13.1 Understanding the Submission Configuration File

The *[dspace]/config/item-submission.xml* contains the submission configurations for *both* the DSpace JSP user interface (JSPUI) or the DSpace XML user interface (XMLUI or Manakin). This configuration file contains detailed documentation within the file itself, which should help you better understand how to best utilize it.

13.1.1 The Structure of item-submission.xml

```
<item-submission>
 <!-- Where submission processes are mapped to specific Collections -->
 <submission-map>
 <name-map collection-handle="default" submission-name="traditional" /> ...
 </submission-map>
 <!-- Where "steps" which are used across many submission processes can be defined in a
    single place. They can then be referred to by ID later. -->
 <step-definitions>
 <step id="collection">
 <processing-class>org.dspace.submit.step.SelectCollectionStep</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 -> Initial Questions -> Describe -> Upload -> Verify -> License -> Complete

If you would like to customize the steps used or the ordering of the steps, you can do so within the <*submission-definition*> section of the *item-submission.xml* .

In addition, you may also specify different Submission Processes for different DSpace Collections. This can be done in the <*submission-map*> section. The *item-submission.xml* file itself documents the syntax required to perform these configuration changes.

13.1.2 Defining Steps (<step>) within the item-submission.xml

This section describes how Steps of the Submission Process are defined within the *item-submission.xml*.

Where to place your <step> definitions

<*step*> definitions can appear in one of two places within the *item-submission.xml* configuration file.

1. Within the <*step-definitions*> section
    - This is for globally defined <*step*> definitions (i.e. steps which are used in multiple <*submission-process*> definitions). Steps defined in this section **must** define a unique *id* which can be used to reference this step.
    - For example:

```
<step-definitions>
 <step id="custom-step">
    ...
 </step>
   ...
</step-definitions>
```

    - The above step definition could then be referenced from within a <*submission-process*> as simply <*step id="custom-step"/*>
2. Within a specific <*submission-process*> definition
    - This is for steps which are specific to a single <*submission-process*> definition.
    - For example:

```
<submission-process>
 <step>
    ...
 </step>
</submission-process>
```

The ordering of <step> definitions matters!

The ordering of the <*step*> tags within a <*submission-process*> definition directly corresponds to the order in which those steps will appear!

For example, the following defines a Submission Process where the *License* step directly precedes the *Initial Questions* step (more information about the structure of the information under each <step> tag can be found in the section on Structure of the <step> Definition below):

```
<submission-process>
 <!--Step 1 will be to Sign off on the License-->
    <step>
      <heading>submit.progressbar.license</heading>
      <processing-class>org.dspace.submit.step.LicenseStep</processing-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).

13.2 Reordering/Removing Submission Steps

The removal of existing steps and reordering of existing steps is a relatively easy process!

Reordering steps

1. Locate the <*submission-process*> tag which defines the Submission Process that you are using. If you are unsure which Submission Process you are using, it's likely the one with *name="traditional"*, since this is the traditional DSpace submission process.
2. Reorder the <*step*> tags within that <*submission-process*> tag. Be sure to move the *entire*<*step*> tag (i.e. everything between and including the opening <*step*> and closing </*step*> tags).

- *Hint #1:* The *<step>* defining the *Review/Verify* step only allows the user to review information from steps which appear **before** it. So, it's likely you'd want this to appear as one of your last few steps
- *Hint #2:* If you are using it, the *<step>* defining the *Initial Questions* step should always appear **before** the *Upload* or *Describe* steps since it asks questions which help to set up those later steps.

Removing one or more steps

1. Locate the *<submission-process>* tag which defines the Submission Process that you are using. If you are unsure which Submission Process you are using, it's likely the one with *name="traditional"*, since this is the traditional DSpace submission process.
2. Comment out (i.e. surround with `<!--` and `-->`) the `<step>` tags which you want to remove from that *<submission-process>* tag. Be sure to comment out the *entire {{<step>}}tag (i.e. everything between and including the opening _<step>* and closing *</step>* tags).
    - *Hint #1:* You cannot remove the *Select a Collection* step, as an DSpace Item cannot exist without belonging to a Collection.
    - *Hint #2:* If you decide to remove the *<step>* defining the *Initial Questions* step, you should be aware that this may affect your *Describe* and *Upload* steps! The *Initial Questions* step asks questions which help to initialize these later steps. If you decide to remove the *Initial Questions* step you may wish to create a custom, automated step which will provide default answers for the questions asked!

13.3 Assigning a custom Submission Process to a Collection

Assigning a custom submission process to a Collection in DSpace involves working with the *submission-map* section of the *item-submission.xml*. For a review of the structure of the *item-submission.xml* see the section above on Understanding the Submission Configuration File.

Each *name-map* element within *submission-map* associates a collection with the name of a submission definition. Its *collection-handle* attribute is the Handle of the collection. Its *submission-name* attribute is the submission definition name, which must match the *name* attribute of a *submission-process* element (in the *submission-definitions* section of *item-submission.xml*.

For example, the following fragment shows how the collection with handle "12345.6789/42" is assigned the "custom" submission process:

```
<submission-map>
    <name-map collection-handle=" 12345.6789/42" submission-name="
 custom" />
    ...
  </submission-map>

  <submission-definitions>
    <submission-process name="
```

```
custom">
   ...
 </submission-definitions>
```

It's a good idea to keep the definition of the *default* name-map from the example *input-forms.xml* so there is always a default for collections which do not have a custom form set.

13.3.1 Getting A Collection's Handle

You will need the *handle* of a collection in order to assign it a custom form set. To discover the handle, go to the "Communities & Collections" page under "Browse" in the left-hand menu on your DSpace home page. Then, find the link to your collection. It should look something like:

```
http://myhost.my.edu/dspace/handle/12345.6789/42
```

The underlined part of the URL is the handle. It should look familiar to any DSpace administrator. That is what goes in the *collection-handle* attribute of your *name-map* element.

13.4 Custom Metadata-entry Pages for Submission

13.4.1 Introduction

This section explains how to customize the Web forms used by submitters and editors to enter and modify the metadata for a new item. These metadata web forms are controlled by the *Describe* step within the Submission Process. However, they are also configurable via their own XML configuration file (*input-forms.xml*).

You can customize the "default" metadata forms used by all collections, and also create alternate sets of metadata forms and assign them to specific collections. In creating custom metadata forms, you can choose:

- The number of metadata-entry pages.
- Which fields appear on each page, and their sequence.
- Labels, prompts, and other text associated with each field.
- List of available choices for each menu-driven field.

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.

13.4.2 Describing Custom Metadata Forms

The description of a set of pages through which submitters enter their metadata is called a *form* (although it is actually a set of forms, in the HTML sense of the term). A form is identified by a unique symbolic *name*. In the XML structure, the *form* is broken down into a series of *pages*: each of these represents a separate Web page for collecting metadata elements.

To set up one of your DSpace collections with customized submission forms, first you make an entry in the *form-map*. This is effectively a table that relates a collection to a form set, by connecting the collection's *Handle* to the form name. Collections are identified by handle because their names are mutable and not necessarily unique, while handles are unique and persistent.

A special map entry, for the collection handle "default", defines the *default* form set. It applies to all collections which are not explicitly mentioned in the map. In the example XML this form set is named *traditional* (for the "traditional" DSpace user interface) but it could be named anything.

13.4.3 The Structure of input-forms.xml

The XML configuration file has a single top-level element, *input-forms*, which contains three elements in a specific order. The outline is as follows:

```
<input-forms>

  <--  Map of Collections to Form Sets -->
  <form-map>
    <name-map collection-handle="default" form-name="traditional"
 />
    ...
  </form-map>

  <--  Form Set Definitions -->
  <form-definitions>
    <form name="traditional">
    ...
  </form-definitions>

  <--  Name/Value Pairs used within Multiple Choice Widgets
 -->
  <form-value-pairs>
    <value-pairs value-pairs-name="common_iso_languages"
```

```
 dc-term="language_iso">
    ...
  </form-value-pairs>
</input-forms>
```

Adding a Collection Map

Each *name-map* element within *form-map* associates a collection with the name of a form set. Its *collection-handle* attribute is the Handle of the collection, and its *form-name* attribute is the form set name, which must match the *name* attribute of a *form* element.

For example, the following fragment shows how the collection with handle "12345.6789/42" is attached to the "TechRpt" form set:

```
<form-map>
    <name-map collection-handle=" 12345.6789/42" form-name=" TechRpt"/>
    ...
  </form-map>

  <form-definitions>
    <form name="TechRept">
    ...
  </form-definitions>
```

It's a good idea to keep the definition of the *default* name-map from the example *input-forms.xml* so there is always a default for collections which do not have a custom form set.

Getting A Collection's Handle

You will need the *handle* of a collection in order to assign it a custom form set. To discover the handle, go to the "Communities & Collections" page under "**Browse**" in the left-hand menu on your DSpace home page. Then, find the link to your collection. It should look something like:

```
http://myhost.my.edu/dspace/handle/12345.6789/42
```

The underlined part of the URL is the handle. It should look familiar to any DSpace administrator. That is what goes in the *collection-handle* attribute of your *name-map* element.

Adding a Form Set

You can add a new form set by creating a new *form* element within the *form-definitions* element. It has one attribute, *name*, which as seen above must match the value of the *name-map* for the collections it is to be used for.

Forms and Pages

The content of the *form* is a sequence of *page* elements. Each of these corresponds to a Web page of forms for entering metadata elements, presented in sequence between the initial "Describe" page and the final "Verify" page (which presents a summary of all the metadata collected).

A *form* must contain at least one and at most six pages. They are presented in the order they appear in the XML. Each *page* element must include a *number* attribute, that should be its sequence number, e.g.

```
<page number="1">
```

The *page* element, in turn, contains a sequence of *field* elements. Each field defines an interactive dialog where the submitter enters one of the Dublin Core metadata items.

Composition of a Field

Each *field* contains the following elements, in the order indicated. The required sub-elements are so marked:

- **dc-schema** (Required) : Name of metadata schema employed, e.g. *dc* for Dublin Core. This value must match the value of the *schema* element defined in *dublin-core-types.xml*
- **dc-element** (Required) : Name of the Dublin Core element entered in this field, e.g. *contributor*.
- **dc-qualifier**: Qualifier of the Dublin Core element entered in this field, e.g. when the field is *contributor.advisor* the value of this element would be *advisor*. Leaving this out means the input is for an unqualified DC element.
- **repeatable**: Value is *true* when multiple values of this field are allowed, *false* otherwise. When you mark a field repeatable, the UI servlet will add a control to let the user ask for more fields to enter additional values. Intended to be used for arbitrarily-repeating fields such as subject keywords, when it is impossible to know in advance how many input boxes to provide.
- **label** (Required): Text to display as the label of this field, describing what to enter, e.g. "*Your Advisor's Name*".
- **input-type** (Required): Defines the kind of interactive widget to put in the form to collect the Dublin Core value. Content must be one of the following keywords:
  - **onebox** – A single text-entry box.
  - **twobox** – A pair of simple text-entry boxes, used for *repeatable* values such as the DC *subject* item. *Note:* The 'twobox' input type is rendered the same as a 'onebox' in the XML-UI, but both allow for ease of adding multiple values.
  - **textarea** – Large block of text that can be entered on multiple lines, e.g. for an abstract.
  - **name** – Personal name, with separate fields for family name and first name. When saved they are appended in the format 'LastName, FirstName'
  - **date** – Calendar date. When required, demands that at least the year be entered.
  - **series** – Series/Report name and number. Separate fields are provided for series name and series number, but they are appended (with a semicolon between) when saved.
  - **dropdown** – Choose value(s) from a "drop-down" menu list. **Note:** You must also include a value for the *value-pairs-name* attribute to specify a list of menu entries from which to choose. Use this to make a choice from a restricted set of options, such as for the *language* item.

    - **qualdrop_value** – Enter a "qualified value", which includes *both* a qualifier from a drop-down menu and a free-text value. Used to enter items like alternate identifiers and codes for a submitted item, e.g. the DC *identifier* field. **Note:** As for the *dropdown* type, you must include the *value-pairs-name* attribute to specify a menu choice list.
    - **list** – Choose value(s) from a checkbox or radio button list. If the *repeatable* attribute is set to *true*, a list of checkboxes is displayed. If the *repeatable* attribute is set to *false*, a list of radio buttons is displayed. **Note:** You must also include a value for the *value-pairs-name* attribute to specify a list of values from which to choose.
- **hint** (Required): Content is the text that will appear as a "hint", or instructions, next to the input fields. Can be left empty, but it must be present.
- **required**: When this element is included with any content, it marks the field as a required input. If the user tries to leave the page without entering a value for this field, that text is displayed as a warning message. For example, *<required>You must enter a title.</required> Note that leaving the* required element empty will *not* mark a field as required, e.g.:*<required></required>*
- **visibility**: When this optional element is included with a value, it restricts the visibility of the field to the scope defined by that value. If the element is missing or empty, the field is visible in all scopes. Currently supported scopes are:
    - **workflow** : the field will only be visible in the workflow stages of submission. This is good for hiding difficult fields for users, such as subject classifications, thereby easing the use of the submission system.
    - **submit** : the field will only be visible in the initial submission, and not in the workflow stages. In addition, you can decide which type of restriction apply: read-only or full hidden the field (default behaviour) using the *otherwise* attribute of the *visibility* XML element. For example:*<visibility otherwise="readonly">workflow</visibility>* Note that it is considered a configuration error to limit a field's scope while also requiring it - an exception will be generated when this combination is detected.
      Look at the example *input-forms.xml* and experiment with a a trial custom form to learn this specification language thoroughly. It is a very simple way to express the layout of data-entry forms, but the only way to learn all its subtleties is to use it.

For the use of controlled vocabularies see the Configuring Controlled Vocabularies section.

Automatically Elided Fields

You may notice that some fields are automatically skipped when a custom form page is displayed, depending on the kind of item being submitted. This is because the DSpace user-interface engine skips Dublin Core fields which are not needed, according to the initial description of the item. For example, if the user indicates there are no alternate titles on the first "Describe" page (the one with a few checkboxes), the input for the *title.alternative* DC element is automatically elided, *even on custom submission pages.*

When a user initiates a submission, DSpace first displays what we'll call the "initial-questions page". By default, it contains three questions with check-boxes:

1. **The item has more than one title, e.g. a translated title** Controls *title.alternative* field.

2. **The item has been published or publicly distributed before** Controls DC fields:
    - *date.issued*
    - *publisher*
    - *identifier.citation*
3. **The item consists of more than one file** *Does not affect any metadata input fields.*

The answers to the first two questions control whether inputs for certain of the DC metadata fields will displayed, even if they are defined as fields in a custom page. Conversely, if the metadata fields controlled by a checkbox are not mentioned in the custom form, the checkbox is elided from the initial page to avoid confusing or misleading the user.

The two relevant checkbox entries are "The item has more than one title, e.g. a translated title", and "The item has been published or publicly distributed before". The checkbox for multiple titles trigger the display of the field with dc-element equal to 'title' and dc-qualifier equal to 'alternative'. If the controlling collection's form set does not contain this field, then the multiple titles question will not appear on the initial questions page.

Adding Value-Pairs

Finally, your custom form description needs to define the "value pairs" for any fields with input types that refer to them. Do this by adding a *value-pairs* element to the contents of *form-value-pairs*. It has the following required attributes:

- **value-pairs-name** – Name by which an *input-type* refers to this list.
- **dc-term** – Qualified Dublin Core field for which this choice list is selecting a value. Each *value-pairs* element contains a sequence of *pair* sub-elements, each of which in turn contains two elements:
- **displayed-value** – Name shown (on the web page) for the menu entry.
- **stored-value** – Value stored in the DC element when this entry is chosen. Unlike the HTML *select* tag, there is no way to indicate one of the entries should be the default, so the first entry is always the default choice.

Example

Here is a menu of types of common identifiers:

```
<value-pairs value-pairs-name="common_identifiers" dc-term="identifier">
     <pair>
       <displayed-value>Gov't Doc #</displayed-value>
       <stored-value>govdoc</stored-value>
     </pair>
     <pair>
       <displayed-value>URI</displayed-value>
       <stored-value>uri</stored-value>
     </pair>
     <pair>
       <displayed-value>ISBN</displayed-value>
```

```
      <stored-value>isbn</stored-value>
    </pair>
  </value-pairs>
```

It generates the following HTML, which results in the menu widget below. (Note that there is no way to indicate a default choice in the custom input XML, so it cannot generate the HTML *SELECTED* attribute to mark one of the options as a pre-selected default.)

```
<select name="identifier_qualifier_0">
    <option VALUE="govdoc">Gov't Doc #</option>
    <option VALUE="uri">URI</option>
    <option VALUE="isbn">ISBN</option>
</select>
```

13.4.4 Deploying Your Custom Forms

The DSpace web application only reads your custom form definitions when it starts up, so it is important to remember:

- *You must always restart Tomcat* (or whatever servlet container you are using) for changes made to the *input-forms.xml* file take effect.

Any mistake in the syntax or semantics of the form definitions, such as poorly formed XML or a reference to a nonexistent field name, will cause a fatal error in the DSpace UI. The exception message (at the top of the stack trace in the *dspace.log* file) usually has a concise and helpful explanation of what went wrong. Don't forget to stop and restart the servlet container before testing your fix to a bug.

13.5 Configuring the File Upload step

The *Upload* step in the DSpace submission process has two configuration options which can be set with your *[dspace]/config/dspace.cfg* configuration file. They are as follows:

- *upload.max* - The maximum size of a file (in bytes) that can be uploaded from the JSPUI (not applicable for the XMLUI). It defaults to 536870912 bytes (512MB). You may set this to -1 to disable any file size limitation.
    - *Note:* Increasing this value or setting to -1 does **not** guarantee that DSpace will be able to successfully upload larger files via the web, as large uploads depend on many other factors including bandwidth, web server settings, internet connection speed, etc.
- *webui.submit.upload.required* - Whether or not all users are *required* to upload a file when they submit an item to DSpace. It defaults to 'true'. When set to 'false' users will see an option to skip the upload step when they submit a new item.

13.6 Creating new Submission Steps

First, a brief warning: *Creating a new Submission Step requires some Java knowledge, and is therefore recommended to be undertaken by a Java programmer whenever possible*

That being said, at a higher level, creating a new Submission Step requires the following (in this relative order):

1. (**Required**) Create a new Step Processing class
    - This class **must** extend the abstract `org.dspace.submit.AbstractProcessingStep` class and implement all methods defined by that abstract class.
    - This class should be built in such a way that it can process the input gathered from *either* the XMLUI or JSPUI interface.
2. (*For steps using JSPUI*) Create the JSPs to display the user interface. Create a new JSPUI "binding" class to initialize and call these JSPs.
    - 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.xml` configuration file.
    - This may also require that you add an I18N (Internationalization) key for this step's *heading*. See the sections on Configuring Multilingual Support for JSPUI (see page 215) or Configuring Multilingual Support for XMLUI (see page 332) 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`.

13.6.1 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).

14 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:

14.1 Introduction

This manual describes the Digital Repository Interface (DRI) as it applies to the DSpace digital repository and XMLUI Manakin based interface. DSpace XML UI is a comprehensive user interface system. It is centralized and generic, allowing it to be applied to all DSpace pages, effectively replacing the JSP-based interface system. Its ability to apply specific styles to arbitrarily large sets of DSpace pages significantly eases the task of adapting the DSpace look and feel to that of the adopting institution. This also allows for several levels of branding, lending institutional credibility to the repository and collections.

Manakin, the second version of DSpace XML UI, consists of several components, written using Java, XML, and XSL, and is implemented in 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.

14.1.1 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.

14.1.2 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.

14.2 DRI in Manakin

The general process for handling a request in DSpace XML UI consists of two parts. The first part builds the XML Document, and the second part stylizes that Document for output. In Manakin, the two parts are not discrete and instead wrapped within two processes: Content Generation, which builds an XML representation of the page, and Style Application, which stylizes the resulting Document. Content Generation is performed by Aspect chaining, while Style Application is performed by a Theme.

14.2.1 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 covert 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 (see page 324) 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 singe arbitrary page or handle.

14.2.2 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.

14.3 Common Design Patterns

There are several design patterns used consistently within the DRI schema. This section identifies the need for and describes the implementation of these patterns. Three patterns are discussed: language and internationalization issues, standard attribute triplet (*id*, *n*, and *rend*), and the use of structure-oriented markup.

14.3.1 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.

14.3.2 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 the theme. There are several cases, however, where the content of the *rend* attribute is outlined in detail and its use is encouraged. Those cases are the emphasis element *hi*, the division element *div*, and the *list* element. Please refer to the Element Reference for more detail on these elements.

14.3.3 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.

14.4 Schema Overview

The DRI XML Document consists of the root element document and three top-level elements that contain two major types of elements. The three top-level containers are *meta*, *body*, and *options*. The two types of elements they contain are metadata and content, carrying metadata about the page and the contents of the page, respectively. Figure 1 depicts the relationship between these six components.

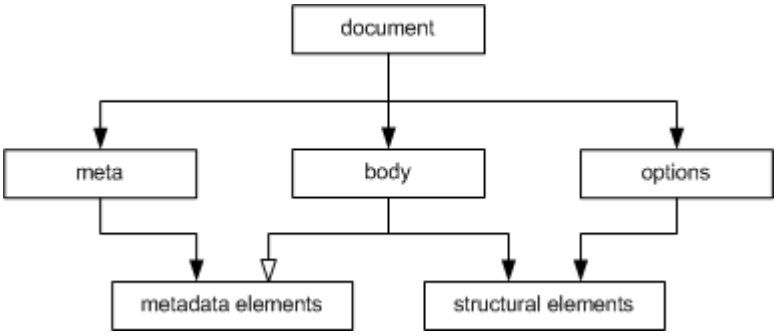


Figure 1: The two content types across three major divisions of a DRI page.

The *document* element is the root for all DRI pages and contains all other elements. It bears only one attribute, *version*, that contains the version number of the DRI system and the schema used to validate the produced document. At the time of writing the working version number is "1.1".

The *meta* element is a the top-level element under document and contains all metadata information about the page, the user that requested it, and the repository it is used with. It contains no structural elements, instead being the only container of metadata elements in a DRI Document. The metadata stored by the meta element is broken up into three major groups: *userMeta*, *pageMeta*, and *objectMeta*, each storing metadata information about their respective component. Please refer to the reference entries for more information about these elements.

The *options* element is another top-level element that contains all navigation and action options available to the user. The options are stored as items in list elements, broken up by the type of action they perform. The five types of actions are: browsing, search, language selection, actions that are always available, and actions that are context dependent. The two action types also contain sub-lists that contain actions available to users of varying degrees of access to the system. The *options* element contains no metadata elements and can only make use of a small set of structural elements, namely the *list* element and its children.

The last major top-level element is the *body* element. It contains all structural elements in a DRI Document, including the lists used by the *options* element. Structural elements are used to build a generic representation of a DSpace page. Any DSpace page can be represented with a combination of the structural elements, which will

in turn be transformed by the XSL templates into another format. This is the core mechanism that allows DSpace XML UI to apply uniform templates and styling rules to all DSpace pages and is the fundamental difference from the JSP approach currently used by DSpace.

The *body* element directly contains only one type of element: *div*. The *div* element serves as a major division of content and any number of them can be contained by the *body*. Additionally, divisions are recursive, allowing *divs* to contain other *divs*. It is within these elements that all other structural elements are contained. Those elements include tables, paragraph elements *p*, and lists, as well as their various children elements. At the lower levels of this hierarchy lie the character container elements. These elements, namely paragraphs *p*, table *cells*, lists *items*, and the emphasis element *hi*, contain the textual content of a DSpace page, optionally modified with links, figures, and emphasis. If the division within which the character class is contained is tagged as interactive (via the *interactive* attribute), those elements can also contain interactive form fields. Divisions tagged as interactive must also provide *method* and *action* attributes for its fields to use.

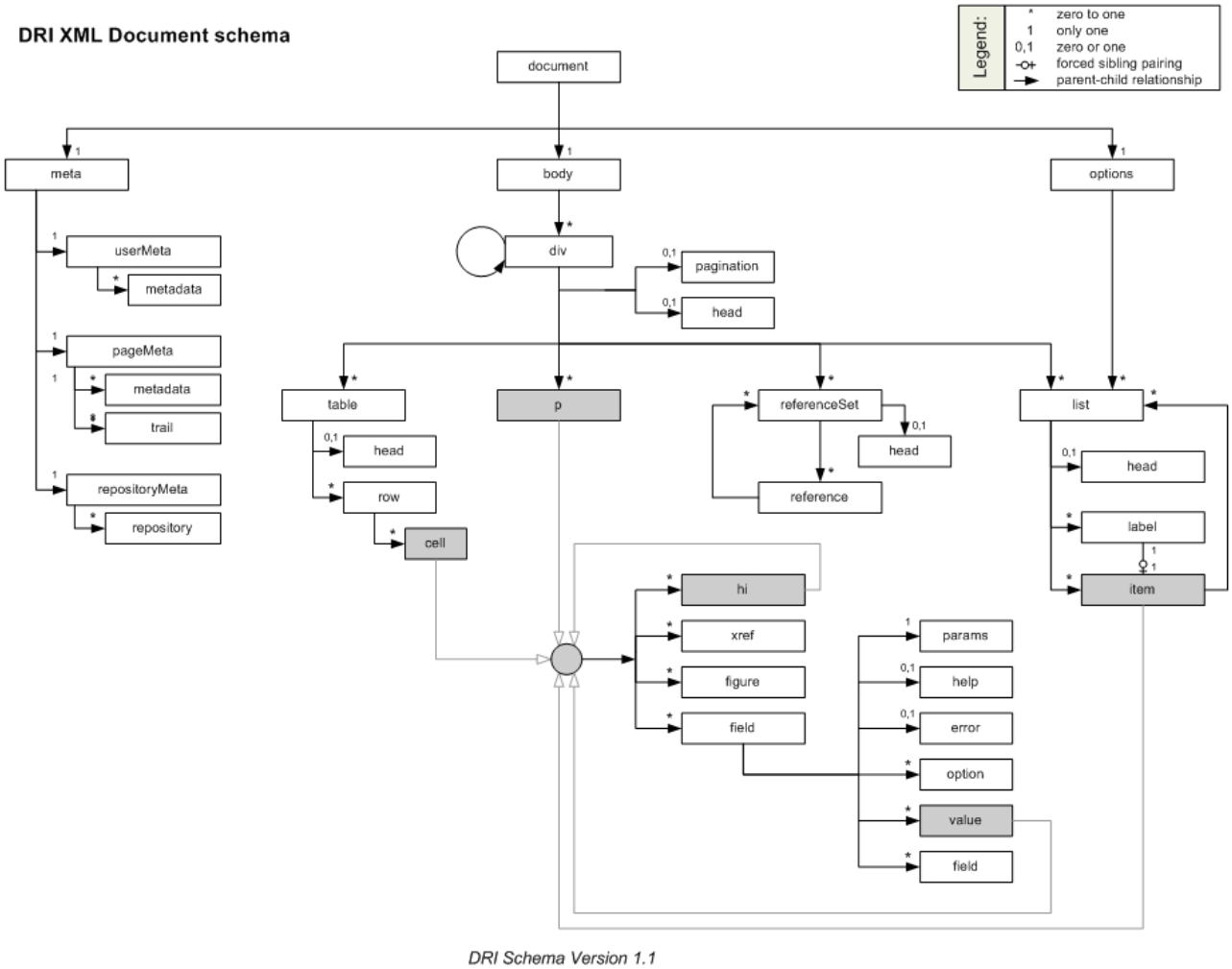


Figure 2: All the elements in the DRI schema (version 1.1).

14.5 Merging of DRI Documents

Having described the structure of the DRI Document, as well as its function in Manakin's Aspect chains, we now turn our attention to the one last detail of their use: merging two Documents into one. There are several situations where the need to merge two documents arises. In Manakin, for example, every Aspect is responsible for adding different functionality to a DSpace page. Since every instance of a page has to be a complete DRI Document, each Aspect is faced with the task of merging the Document it generated with the ones generated (and merged into one Document) by previously executed Aspects. For this reason rules exist that describe which elements can be merged together and what happens to their data and child elements in the process.

When merging two DRI Documents, one is considered to be the main document, and the other a feeder document that is added in. The three top level containers (*meta*, *body* and *options*) of both documents are then individually analyzed and merged. In the case of the *options* and *meta* elements, the children tags are taken individually as well and treated differently from their siblings.

The *body* elements are the easiest to merge: their respective *div* children are preserved along with their ordering and are grouped together under one element. Thus, the new *body* tag will contain all the *divs* of the main document followed by all the *divs* of the feeder. However, if two *divs* have the same *n* and *rend* attributes (and in case of an interactive *div* the same *action* and *method* attributes as well), those *divs* will be merged into one. The resulting div will bear the *id*, *n*, and *rend* attributes of the main document's div and contain all the *divs* of the main document followed by all the *divs* of the feeder. This process continues recursively until all the *divs* have been merged. It should be noted that two divisions with separate pagination rules cannot be merged together.

Merging the *options* elements is somewhat different. First, *list* elements under *options* of both documents are compared with each other. Those unique to either document are simply added under the new options element, just like *divs* under *body*. In case of duplicates, that is *list* elements that belong to both documents and have the same *n* attribute, the two *lists* will be merged into one. The new *list* element will consist of the main document's head element, followed *label-item* pairs from the main document, and then finally the *label-item* pairs of the feeder, provided they are different from those of the main.

Finally, the *meta* elements are merged much like the elements under body. The three children of *meta* - *userMeta*, *pageMeta*, and *objectMeta* - are individually merged, adding the contents of the feeder after the contents of the main.

14.6 Version Changes

The DRI schema will continue to evolve overtime as the needs of interface design require. The version attribute on the document will indicate which version of the schema the document conforms to. At the time Manakin was incorporated into the standard distribution of DSpac the current version was "1.1", however earlier versions of the Manakin interface may use "1.0".

14.6.1 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 refrencing metadata contained on the *objectMeta* element they reference metadata in external files. The *repository* and *repositoryMeta* elements were alse modified in a similar manner removing in-line metadata and refrencing external metadata documents.

14.7 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	
	rend	
DOCUMENT	version	required
field		
	disabled	
	id	required
	n	required
	rend	
	required	
	type	required
figure		
	rend	
	source	
	target	
head		
	id	
	n	
	rend	
help		
hi	rend	required
instance		
item		
	id	
	n	

	rend	
label		
	id	
	n	
	rend	
list		
	id	required
	n	required
	rend	
	type	
META		
metadata		
	element	required
	language	
	qualifier	
OPTIONS		
p		
	id	
	n	
	rend	
pageMeta		
params		
	cols	
	maxlength	
	multiple	
	operations	
	rows	

	size	
reference		
	url	required
	repositoryID	required
	type	
referenceSet		
	id	required
	n	required
	orderBy	
	rend	
	type	required
repository		
	repositoryID	required
	url	required
repositoryMeta		
row		
	id	
	n	
	rend	
	role	required
table		
	cols	required
	id	required
	n	required
	rend	
	rows	required
trail		
	rend	

	target	
userMeta	authenticated	required
value		
	optionSelected	
	optionValue	
	type	required
xref	target	required

14.7.1 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>
```

14.7.2 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>
```

14.7.3 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 exit across all paginated divs.
- **lastItemIndex**: (optional) For paginated divs, the lastItemIndex attribute indicates the index of the last item included in this div.
- **method**: (required for interactive) Accepted values are "get", "post", and "multipart". Determines the method used to pass gathered field values to the handler specified by the action attribute. The multipart method should be used for uploading files.
- **n**: (required) A local identifier used to differentiate the element from its siblings.
- **nextPage**: (optional) For paginated divs the nextPage attribute points to the URL of the next page of the div, if it exists.
- **pagesTotal**: (optional) For paginated divs, the pagesTotal attribute indicates how many pages the paginated divs spans.
- **pageURLMask**: (optional) For paginated divs, the pageURLMask attribute contains the mask of a url to a particular page within the paginated set. The destination page's number should replace the {pageNum} string in the URL mask to generate a full URL to that page.
- **pagination**: (optional) Accepted values are "simple", "masked". This attribute determines whether the div is spread over several pages. Simple paginated divs must provide previousPage, nextPage, itemsTotal, firstItemIndex, lastItemIndex attributes. Masked paginated divs must provide currentPage, pagesTotal, pageURLMask, itemsTotal, firstItemIndex, lastItemIndex attributes.
- **previousPage**: (optional) For paginated divs the previousPage attribute points to the URL of the previous page of the div, if it exists.
- **rend**: (optional) A rendering hint used to override the default display of the element. In the case of the div tag, it is also encouraged to label it as either "primary" or "secondary". Divs marked as primary contain content, while secondary divs contain auxiliary information or supporting fields.

```
<body>
     <div n="division-example"
 id="XMLExample.div.division-example">
      <head> Example Division </head>
      <p> This example shows the use of divisions. </p>
      <table ...>
        ...
      </table>
      <referenceSet ...>
        ...
      </referenceSet>
      <list ...>
        ...
      </list>
       <div n="sub-division-example"
```

```
 id="XMLExample.div.sub-division-example">
        <p> Divisions may be nested </p>
        ...
       </div>
      ...
     </div>
    ...
</body>
```

14.7.4 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>
```

14.7.5 field

Text Container

Structural Element

The *field* element is a container for all information necessary to create a form field. The required *type* attribute determines the type of the field, while the children tags carry the information on how to build it. Fields can only occur in divisions tagged as "interactive".

Parent

- cell
- p
- hi
- item

Children

- params (one)
- help (zero or one)
- error (any)
- option (any - only with the select type)
- value (any - only available on fields of type: select, checkbox, or radio)
- field (one or more - only with the composite type)
- valueSet (any)

Attributes

- **disabled**: (optional) Accepted values are "yes", "no". Determines whether the field allows user input. Rendering of disabled fields may vary with implementation and display media.
- **id**: (required) A unique identifier for a field element.
- **n**: (required) A non-unique local identifier used to differentiate the element from its siblings within an interactive division. This is the name of the field use when data is submitted back to the server.
- **rend**: (optional) A rendering hint used to override the default display of the element.
- **required**: (optional) Accepted values are "yes", "no". Determines whether the field is a required component of the form and thus cannot be left blank.
- **type**: (required) A required attribute to specify the type of value. Accepted types are:
    - **button**: A button input control that when activated by the user will submit the form, including all the fields, back to the server for processing.
    - **checkbox**: A boolean input control which may be toggled by the user. A checkbox may have several fields which share the same name and each of those fields may be toggled independently. This is distinct from a radio button where only one field may be toggled.
    - **file**: An input control that allows the user to select files to be submitted with the form. Note that a form which uses a file field must use the multipart method.

- **hidden**: An input control that is not rendered on the screen and hidden from the user.
- **password**: A single-line text input control where the input text is rendered in such a way as to hide the characters from the user.
- **radio**: A boolean input control which may be toggled by the user. Multiple radio button fields may share the same name. When this occurs only one field may be selected to be true. This is distinct from a checkbox where multiple fields may be toggled.
- **select**: A menu input control which allows the user to select from a list of available options.
- **text**: A single-line text input control.
- **textarea**: A multi-line text input control.
- **composite**: A composite input control combines several input controls into a single field. The only fields that may be combined together are: checkbox, password, select, text, and textarea. When fields are combined together they can posses multiple combined values.

```
<p>
  <hi> ... </hi>
  <xref> ... </xref>
  <figure> ... </figure>
  ...
   <field id="XMLExample.field.name" n="name" type="text"
 required="yes">
    <params size="16" maxlength="32"/>
    <help>Some help text with <i18n>localized
 content</i18n>.</help>
    <value type="raw">Default value goes
 here</value>
   </field>
</p>
```

14.7.6 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>
```

14.7.7 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>
```

14.7.8 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>
```

14.7.9 hi

Rich Text Container

Structural Element

The *hi* element is used for emphasis of text and occurs inside character containers like *p* and *list* item. It can be mixed freely with text, and any text within the tag itself will be emphasized in a manner specified by the required *rend* attribute. Additionally, *hi* element is the only text container component that is a rich text container itself, meaning it can contain other tags in addition to plain text. This allows it to contain other text containers, including other *hi* tags.

Parent

- cell
- p
- item
- hi

Children

- hi (any)
- xref (any)
- figure (any)
- field (any)

Attributes

- **rend**: (required) A required attribute used to specify the exact type of emphasis to apply to the contained text. Common values include but are not limited to "bold", "italic", "underline", and "emph".

```
<p>
    This text is normal, while  <hi rend="bold">this text is bold and
 this text is <hi rend="italic">bold and
 italic.</hi></hi>
</p>
```

14.7.10 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.
```

14.7.11 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>
```

14.7.12 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>
```

14.7.13 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>
```

14.7.14 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>
```

14.7.15 metadata

Text Container

Structural Element

The *metadata* element carries generic metadata information in the form on an attribute-value pair. The type of information it contains is determined by two attributes: *element*, which specifies the general type of metadata stored, and an optional *qualifier* attribute that narrows the type down. The standard representation for this pairing is element.qualifier. The actual metadata is contained in the text of the tag itself. Additionally, a *language* attribute can be used to specify the language used for the metadata entry.

Parent

- userMeta
- pageMeta

Children

- none

Attributes

- **element**: (required) The name of a metadata field.
- **language**: (optional) An optional attribute to specify the language used in the metadata tag.
- **qualifier**: (optional) An optional postfix to the field name used to further differentiate the names.

```
<meta>
  <userMeta>
     <metadata element="identifier" qualifier="firstName"> Bob
 </metadata> <metadata element="identifier" qualifier="lastName"> Jones
 </metadata> <metadata ...> ...
 </metadata>
    ...
  </userMeta>
  <pageMeta>
     <metadata element="rights"
 qualifier="accessRights">user</metadata> <metadata ...> ...
 </metadata>
    ...
  </pageMeta>
</meta>
```

14.7.16 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>
```

14.7.17 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>
```

14.7.18 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 form 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>
```

14.7.19 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>
```

14.7.20 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>
```

14.7.21 referenceSet

Metadata Reference Element

The *referenceSet* element is a container of artifact or repository references.

Parent

- div
- reference

Children

- head (zero or one)
- reference (any)

Attributes

- **id**: (required) A unique identifier of the element
- **n**: (required) Local identifier used to differentiate the element from its siblings
- **orderBy**: (optional) A reference to the metadata field that determines the ordering of artifacts or repository objects within the set. When the Dublin Core metadata scheme is used this attribute should be the element.qualifier value that the set is sorted by. As an example, for a browse by title list, the value should be sortedBy=title, while for browse by date list it should be sortedBy=date.created
- **rend**: (optional) A rendering hint used to override the default display of the element.
- **type**: (required) Determines the level of detail for the given metadata. Accepted values are:
    - **summaryList**: Indicates that the metadata from referenced artifacts or repository objects should be used to build a list representation that is suitable for quick scanning.
    - **summaryView**: Indicates that the metadata from referenced artifacts or repository objects should be used to build a partial view of the referenced object or objects.
    - **detailList**: Indicates that the metadata from referenced artifacts or repository objects should be used to build a list representation that provides a complete, or near complete, view of the referenced objects. Whether such a view is possible or different from summaryView depends largely on the repository at hand and the implementing theme.
    - **detailView**: Indicates that the metadata from referenced artifacts or repository objects should be used to display complete information about the referenced object. Rendering of several references included under this type is up to the theme.

```
<div ...>
            <head> Example Division </head>
            <p> ... </p>
```

```
            <table> ... </table>
            <list>
            ...
            </list>
             <referenceSet n="browse-list"
id="XMLTest.referenceSet.browse-list" type="summaryView"
informationModel="DSpace">
            <head>A header for the includeset</head>
            <reference
url="/metadata/handle/123/34/mets.xml"/>
            <reference
url=""metadata/handle/123/34/mets.xml/>
             </referenceSet>
            ...
            </p>
```

14.7.22 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>
```

14.7.23 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>
```

14.7.24 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>
```

14.7.25 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>
```

14.7.26 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>
```

14.7.27 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>
```

14.7.28 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 of other source, and the interpreted element which stores the value in a format appropriate for display to the user, possibly including rich text markup.

Parent

- field

Children

- hi (any)
- xref (any)
- figure (any)

Attributes

- **optionSelected**: (optional) An optional attribute for select, checkbox, and radio fields to determine if the value is to be selected or not.
- **optionValue**: (optional) An optional attribute for select, checkbox, and radio fields to determine the value that should be returned when this value is selected.
- **type**: (required) A required attribute to specify the type of value. Accepted types are:
    - **raw**: The raw type stores the unprocessed value directly from the user of other source.

- **interpreted**: The interpreted type stores the value in a format appropriate for display to the user, possibly including rich text markup.
- **default**: The default type stores a value supplied by the system, used when no other values are provided.

```
<p>
  <hi> ... </hi>
  <xref> ... </xref>
  <figure> ... </figure>
  <field id="XMLExample.field.name" n="name" type="text"
 required="yes">
    <params size="16" maxlength="32"/>
    <help>Some help text with <i18n>localized
 content</i18n>.</help>
     <value type="default">Author,
 John</value>
  </field>
</p>
```

14.7.29 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>
```

15 Appendices

15.1 Appendix A

15.1.1 Default Dublin Core Metadata Registry

contributor		A person, organization, or service responsible for the content of the resource. Catch-all for unspecified contributors.
contributor	advisor	Use primarily for thesis advisor.
contributor[1]	author	
contributor	editor	
contributor	illustrator	
contributor	other	
coverage	spatial	Spatial characteristics of content.
coverage	temporal	Temporal characteristics of content.
creator		Do not use; only for harvested metadata.
date		Use qualified form if possible.
date[1]	accessioned	Date DSpace takes possession of item.
date[1]	available	Date or date range item became available to the public.
date	copyright	Date of copyright.
date	created	Date of creation or manufacture of intellectual content if different from date.issued.
date[1]	issued	Date of publication or distribution.
date	submitted	Recommend for theses/dissertations.
identifier		Catch-all for unambiguous identifiers not defined by qualified form; use identifier.other for a known identifier common to a local collection instead of unqualified form.
identifier[1]	citation	Human-readable, standard bibliographic citation of non-DSpace format of this item

identifier[1]	govdoc	A government document number
identifier[1]	isbn	International Standard Book Number
identifier[1]	issn	International Standard Serial Number
identifier	sici	Serial Item and Contribution Identifier
identifier[1]	ismn	International Standard Music Number
identifier[1]	other	A known identifier type common to a local collection.
identifier[1]	uri	Uniform Resource Identifier
description[1]		Catch-all for any description not defined by qualifiers.
description[1]	abstract	Abstract or summary.
description[1]	provenance	The history of custody of the item since its creation, including any changes successive custodians made to it.
description[1]	sponsorship	Information about sponsoring agencies, individuals, or contractual arrangements for the item.
description	statementofresponsibility	To preserve statement of responsibility from MARC records.
description	tableofcontents	A table of contents for a given item.
description	uri	Uniform Resource Identifier pointing to description of this item.
format[1]		Catch-all for any format information not defined by qualifiers.
format[1]	extent	Size or duration.
format	medium	Physical medium.
format[1]	mimetype	Registered MIME type identifiers.
language		Catch-all for non-ISO forms of the language of the item, accommodating harvested values.
language[1]	iso	Current ISO standard for language of intellectual content, including country codes (e.g. "en_US").
publisher[1]		Entity responsible for publication, distribution, or imprint.
relation		Catch-all for references to other related items.
relation	isformatof	References additional physical form.
relation	ispartof	References physically or logically containing item.
relation[1]	ispartofseries	Series name and number within that series, if available.

relation	haspart	References physically or logically contained item.
relation	isversionof	References earlier version.
relation	hasversion	References later version.
relation	isbasedon	References source.
relation	isreferencedby	Pointed to by referenced resource.
relation	requires	Referenced resource is required to support function, delivery, or coherence of item.
relation	replaces	References preceeding item.
relation	isreplacedby	References succeeding item.
relation	uri	References Uniform Resource Identifier for related item
rights		Terms governing use and reproduction.
rights	uri	References terms governing use and reproduction.
source		Do not use; only for harvested metadata.
source	uri	Do not use; only for harvested metadata.
subject[1]		Uncontrolled index term.
subject	classification	Catch-all for value from local classification system. Global classification systems will receive specific qualifier
subject	ddc	Dewey Decimal Classification Number
subject	lcc	Library of Congress Classification Number
subject	lcsh	Library of Congress Subject Headings
subject	mesh	MEdical Subject Headings
subject	other	Local controlled vocabulary; global vocabularies will receive specific qualifier.
title[1]		Title statement/title proper.
title[1]	alternative	Varying (or substitute) form of title proper appearing in item, e.g. abbreviation or translation
type[1]		Nature or genre of content.

[1]Used by system. **DO NOT REMOVE**

15.1.2 Default Bitstream Format Registry

Mimetype	Short Description	Description	Support Level	Internal	Extensions
application/octet-stream[1]	Unknown	Unknown data format	Unknown	false	
text/plain[1]	License	Item-specific license agreed upon to submission	Known	true	
application/marc	MARC	Machine-Readable Cataloging records	Known	false	
application/mathematica	Mathematica	Mathematica Notebook	Known	false	ma
application/msword	Microsoft Word	Microsoft Word	Known	false	doc
application/pdf	Adobe PDF	Adobe Portable Document Format	Known	false	pdf
application/postscript	Postscript	Postscript Files	Known	false	ai, eps, ps
application/sgml	SGML	SGML application (RFC 1874)	Known	false	sgm, sgml
application/vnd.ms-excel	Microsoft Excel	Microsoft Excel	Known	false	xls
application/vnd.ms-powerpoint	Microsoft Powerpoint	Microsoft Powerpoint	Known	false	ppt
application/vnd.ms-project	Microsoft Project	Microsoft Project	Known	false	mpd, mpp, mpx
application/vnd.visio	Microsoft Visio	Microsoft Visio	Known	false	vsd
application/wordperfect5.1	WordPerfect	WordPerfect 5.1 document	Known	false	wpd
application/x-dvi	TeX dvi	TeX dvi format	Known	false	dvi
application/x-filemaker	FMP3	Filemaker Pro	Known	false	fm
application/x-latex	LateX	LaTeX document	Known	false	latex
application/x-photoshop	Photoshop	Photoshop	Known	false	pdd, psd
application/x-tex	TeX	Tex/LateX document	Known	false	tex
audio/basic	audio/basic	Basic Audio	Known	false	au, snd

audio/x-aiff	AIFF	Audio Interchange File Format	Known	false	aif, aifc, aiff
audio/x-mpeg	MPEG Audio	MPEG Audio	Known	false	abs, mpa, mpega
audio/x-pn-realaudio	RealAudio	RealAudio file	Known	false	ra, ram
audio/x-wav	WAV	Broadcase Wave Format	Known	false	wav
image/gif	GIF	Graphics Interchange Format	Known	false	gif
image/jpeg	JPEG	Joint Photographic Experts Group/JPEG File Interchange Format (JFIF)	Known	false	jpeg, jpg
image/png	image/png	Portable Network Graphics	Known	false	png
image/tiff	TIFF	Tag Image File Format	Known	false	tif, tiff
image/x-ms-bmp	BMP	Microsoft Windows bitmap	Known	false	bmp
image/x-photo-cd	Photo CD	Kodak Photo CD image	Known	false	pcd
text/css	CSS	Cascading Style Sheets	Known	false	css
text/html	HTML	Hypertext Markup Language	Known	false	htm, html
text/plain	Text	Plain Text	Known	false	asc, txt
text/richtext	RTF	Rich Text Format	Known	false	rtf
text/xml	XML	Extensible Markup Language	Known	false	xml
video/mpeg	MPEG	Moving Picture Experts Group	Known	false	mpe, mpeg, mpg
video/quicktime	Video Quicktime	Video Quicktime	Known	false	mov, qt

[1] Used by system: do not remove

16 History

16.1 Changes in DSpace 1.8.2

16.1.1 General Improvements

Improvements in 1.8.2 **(2 issues)**			
Key	Summary	Assignee	Reporter
DS-949	Curation needs to document queueing with workflow configuration	Richard Rodgers	Wendy Bossons
DS-1124	Increase the default upload limit to the maximum allowed by cocoon (2GB)	Scott Phillips	Scott Phillips

16.1.2 Bug Fixes

Bugs Fixed in 1.8.2 **(8 issues)**			
Key	Summary	Assignee	Reporter
DS-910	Encoding of discovery facet urls	Kevin Van de Velde	Jennifer Whalan
DS-918	Concurrent task claiming and editing of metadata possible for same item in submission workflow	Kevin Van de Velde	Bill Hays
DS-1048	superfluous warning in dspace.log	Kevin Van de Velde	Ivan Masár
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	Kevin Van de Velde

16.2 Changes in DSpace 1.8.1

16.2.1 General Improvements

Improvements in 1.8.1 **(1 issues)**			
Key	Summary	Assignee	Reporter
DS-1099	Bulgarian Translation for DSpace 1.8.1	Claudia Jürgen	Vladislav Zhivkov

16.2.2 Bug Fixes

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	Bram Luyten (@mire)
DS-1052	Items without date.accessioned are perminantly 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 muliple 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	Kevin Van de Velde
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

16.3 Changes in DSpace 1.8.0

16.3.1 New Features

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
DS-602	Marker ticket for developing a Sword client for DSpace.	Robin Taylor	Robin Taylor
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	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

16.3.2 General Improvements

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
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
DS-884	Upgrade DSpace XMLUI to use Spring 3.0.5.RELEASE	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
DS-905	upgrade PDFBox and associated jempbox and fontbox to version 1.5.0	Richard Rodgers	Bill Hays
DS-935	REST-API Has a dependency on the dspace-jspui-api, it would be nice if this could be removed.	Robin Taylor	Robin Taylor
DS-938	filter-media doesn't always say what object was unfilterable	Mark H. Wood	Mark H. Wood
DS-939	Enhancements to Curation Framework (Site-wide Tasks & Associated EPerson with Tasks)	Tim Donohue	Tim Donohue
DS-941	Marker ticket for splitting up the dspace.cfg.	Unassigned	Robin Taylor
DS-963	DSpaceI18NTransformer.java logs too aggressively	Stuart Lewis	Stuart Lewis
DS-964	Implement Creative Commons Web Service API with the Submission Process	Robin Taylor	Wendy Bossons
DS-965	Add DSpace Version to XMLUI Control Panel	Tim Donohue	Tim Donohue

DS-969	Add ability to monitor & clear Cocoon's Cache from XMLUI Control Panel	Tim Donohue	Tim Donohue
DS-971	New Discovery configuration system	Kevin Van de Velde	Kevin Van de Velde
DS-974	REST API - Tidy up pom.xml and update version to be 1.8.0-SNAPSHOT	Robin Taylor	Robin Taylor
DS-975	REST API - remove some stray files from the module	Robin Taylor	Robin Taylor
DS-979	Move 'dspace-solr-stats.cfg' to /config/modules/solr-statistics.cfg	Mark Diggory	Tim Donohue
DS-980	Upgraded solr & lucene to version 3.3.0	Tim Donohue	Kevin Van de Velde
DS-982	Usability Improvements to Curation System	Richard Rodgers	Richard Rodgers
DS-988	Consumers called in order of configuration	Richard Rodgers	Kevin Van de Velde
DS-989	Module directory maven property support	Ben Bosman	Kevin Van de Velde
DS-990	Change 'ant update' to default to "overwrite=true" mode	Hardy Pottinger	Tim Donohue
DS-991	Tidy up code on org.dspace.search.Harvest.list()	Robin Taylor	Robin Taylor
DS-999	Allow SWORDMETSIngester to comply with the mets.default.ingest.useCollectionTemplate setting	Stuart Lewis	Stuart Lewis
DS-1000	Move example/out-of-box curation tasks from org.dspace.curate to org.dspace.ctask.general	Kim Shepherd	Kim Shepherd
DS-1003	Add ability to exclude bundles from AIP Export	Tim Donohue	Tim Donohue
DS-1011	Various minor improvements for Mirage	Ben Bosman	Ben Bosman
DS-1016	Use consistent parameter naming in the configurable workflow	Kevin Van de Velde	Kevin Van de Velde
DS-1036	Cleanup Documentation and README files (Maintaining HTML Docs in SVN creates "merge noise" for enduser developers.)	Robin Taylor	Mark Diggory

16.3.3 Bug Fixes

Bugs Fixed in 1.8.0 **(81 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	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-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			Kim Shepherd

	SWORD deposits fail when ingest events are fired if Discovery event consumer is configured	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)
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-816	Order of search types in XMLUI Advanced Search is arbitrary	Peter Dietz	Peter Dietz
DS-822	Default dc metadata schema not used when selecting fields from DIM for default item display	Robin Taylor	Claudia Jürgen
DS-823	DatabaseManager is no longer Oracle compliant	Ben Bosman	Ben Bosman
DS-826	LDAPServlet.java incorrectly redirects to incorrect.jsp rather than ldap-incorrect.jsp	Stuart Lewis	Terry Burton
DS-827	Autoregister in LDAPHierarchicalAuthentication is incompatible with ldap.netid_email_domain option	Stuart Lewis	Terry Burton
DS-834	CSV import dialog doesn't handle csv errors gracefully	Stuart Lewis	Timo Aalto
DS-843	Autocomplete in authority control contains small errors in Mirage	Ben Bosman	Ben Bosman
DS-847	Items without title or author display incorrectly in browse lists	Peter Dietz	stuart yeates
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
DS-860	SWORD still uses dspace.url rather than dspace.baseUrl	Kim Shepherd	Stuart Lewis
DS-863	Still possible to register despite xmlui.user.registration set to false		

		Tim Donohue	Samuel Ottenhoff
DS-866	OAI requests with resumption token start at the wrong offset if non-public items are included and there are withdrawn items	Unassigned	Andrea Schweer
DS-867	typo in messages.xml	Claudia Jürgen	Konstantinos V. Paraskevopoulos 1
DS-868	Collection trail with discovery module is not working	Mark Diggory	Joonas Kesäniemi
DS-869	Community/Collection admin cannot perform curation tasks on individual items.	Tim Donohue	Peter Dietz
DS-871	XMLUI caches community / collection page which doesn't show a recently submitted item immediately	Peter Dietz	Peter Dietz
DS-873	Creative commons license bundle on embargoed item will cause internal system errors on item pages	Robin Taylor	Kevin Van de Velde
DS-874	NullPointerException possible during item browsing	Robin Taylor	Denys Slipetskyy
DS-875	DSpace Configuration service error when using "dspace" script.	Mark Diggory	Kevin Van de Velde
DS-878	update_geolite timesout when downloading GeoLiteCity.dat.gz	Tim Donohue	Lighton Phiri
DS-879	Spelling Error for "dspace help" output - structure-builder description	Claudia Jürgen	Lighton Phiri
DS-881	DSpace doesn't build properly with Maven 3	Mark Diggory	Tim Donohue
DS-891	SWORD web.xml initialization of Service Manager.	Unassigned	Mark Diggory
DS-897	Selecting "This Collection" in the sidebar search box produces a Page Not Found	Andrea Schweer	Bram Luyten (@mire)
DS-898	Caching error in browse, search and discovery results	Ben Bosman	Ben Bosman
DS-904	Non consistant naming of Creative Commons license bundle	Peter Dietz	Peter Dietz
DS-907	COinS not recognized in listings and single item view by Zotero or Citavi	Robin Taylor	Claudia Jürgen

DS-915	BitstreamReader.java will always close the database connection even if the bitstream is being pulled into the theme.	Scott Phillips	Scott Phillips
DS-922	REST-API In an Item query the metadata is exposed as one String rather than as discrete Xml elements	Robin Taylor	Robin Taylor
DS-930	Several DSpace Crosswalks create a new Context and never close it	Tim Donohue	Tim Donohue
DS-931	METS Disseminator never resets its counter for METS @ID attributes	Tim Donohue	Tim Donohue
DS-953	Packager restore-mode fails when no handle is found.	Scott Phillips	Scott Phillips
DS-957	themepath override regex wrong	Tim Donohue	James Russell
DS-959	XMLUI login failure when using Tomcat 7.0.16	Tim Donohue	Stuart Lewis
DS-967	Item view in Mirage theme generates self-closing when abstract is empty	Ben Bosman	David Chandek-Stark
DS-973	Improve error reporting during CSV import	Stuart Lewis	Ivan Masár
DS-985	JUnit Tests do not properly start the ServiceManager and Kernel	Kevin Van de Velde	Mark Diggory
DS-987	By default, Solr Schemas & Configs don't upgrade properly & may cause instability	Tim Donohue	Tim Donohue
DS-993	DSpace fails to start with spurious space character in dspace.cfg	Andrea Schweer	Andrea Schweer
DS-997	Discovery errors out if an item has multiple 'dc.date.accessioned' values	Unassigned	Tim Donohue
DS-998	Shibboleth login - improve handling of empty attribute headers	Stuart Lewis	Stuart Lewis
DS-1010	Item count (collection strengths) do not update on the community-collection list.	Scott Phillips	Scott Phillips
DS-1014	The ORE Dissimenator incorrectly escapes the forward slash character.	Scott Phillips	Scott Phillips
DS-1018	'dspace-1.8.0-rc1-release.zip' is structured differently than 'dspace-1.8.0-rc1-src-release.zip'	Mark Diggory	Tim Donohue

DS-1019	Update code usages of ConfigurationManager.getProperty to reflect changes in config/modules	Robin Taylor	Peter Dietz
DS-1022	Cannot select an option from Authority Control in Chrome or FF	Robin Taylor	Tim Donohue
DS-1024	DTD of input forms file is incomplete	Andrea Schweer	Juan García
DS-1025	Mouseover any title in XMLUI displays COINS info (see Advanced Search or Browse by Title	Robin Taylor	Tim Donohue
DS-1026	License selection shows error "Submission failed due to Invalid answer in answers."	Robin Taylor	Gareth Waller
DS-1027	CC License selection view now supports non-CC licenses - misleading text	Robin Taylor	Gareth Waller
DS-1029	AIP Backup & Restore unable to auto-create metadata fields on restore/ingest ('createMetadataFields' option is ignored)	Tim Donohue	Tim Donohue
DS-1031	Collection OAI Harvesting Settings cannot be saved & Metadata formats is empty (XMLUI)	Robin Taylor	Onivaldo Rosa Junior
DS-1034	Incorrectly active breakcrumb links (XMLUI)	Mark H. Wood	Tim Donohue
DS-1035	Breadcrumb trail for "Home" is incorrect (XMLUI, Mirage Theme)	Mark H. Wood	Tim Donohue
DS-1037	When selecting CC Licence no radio buttons are pre-selected	Richard Rodgers	Robin Taylor
DS-1041	Selecting a CC licence from JSPUI doesn't seem to do anything, no bitstream and nothing in metadata.	Richard Rodgers	Robin Taylor
DS-1042	Item Import - errors when no contents file and metadata_*.xml	Stuart Lewis	Stuart Lewis
DS-1045	The cannot register page in XMLUI has some invalid messages in the trail	Kevin Van de Velde	Kevin Van de Velde
DS-1051	authentication-ip.cfg	Stuart Lewis	Jordan Pišanc
DS-1058	Empty discovery sidebar facet when no facets are present	Kevin Van de Velde	Kevin Van de Velde

16.4 Changes in DSpace 1.7.2

16.4.1 Bug Fixes

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

16.5 Changes in DSpace 1.7.1

16.5.1 General Improvements

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
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

16.5.2 Bug Fixes

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 unifished submission/workflow task wron link in collection column	Claudia Jürgen	Claudia Jürgen
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	AbstractMETSIngester 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
DS-860	SWORD still uses dspace.url rather than dspace.baseUrl	Kim Shepherd	Stuart Lewis

16.6 Changes in DSpace 1.7.0

16.6.1 New Features

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
DS-733	Tools for (load) testing	Graham Triggs	Graham Triggs

16.6.2 General Improvements

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 section expired	Tim Donohue	Antero Neto
DS-618	Recommended versions of prerequisites becoming outdated	Mark H. Wood	Mark H. Wood
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' commmand)	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
DS-660	Tidy up DCDate and DCDateTest.	Robin Taylor	Robin Taylor
DS-662	DSpace import does not use a predictable ordering when running an import.	Scott Phillips	Scott Phillips
DS-664	The registery loader command does not load schemas.	Scott Phillips	Scott Phillips
DS-670	Relax performance tests in org.dspace.content.CommunityCollectionIntegrationTest.	Robin Taylor	Robin Taylor
DS-694	Add inner class for metadata to Item, delay metadata loading until required, reduce impact of changes	Graham Triggs	Graham Triggs
DS-695	Separate DatabaseManager from registering / creating database pool, allow datasource to be retrieved from JNDI	Graham Triggs	Graham Triggs
DS-696	Reduce browse prune cost	Graham Triggs	Graham Triggs
DS-697	Extend supported input formats for thumbnail generation in MediaFilter	Graham Triggs	Graham Triggs
DS-699	ChoiceAuthority plugin for old-style controlled vocabularies	Jeffrey Trimble	Michael B. Klein
DS-703	Making the OAI max amount of records configurable	Ben Bosman	Ben Bosman
DS-704	Update pdfbox library to improve performance and out-of-box support for pdf extraction	Andrea Bollini	Andrea Bollini
DS-706	Alternative to dri2xhtml, with the goal to be more developer-friendly	Ben Bosman	Ben Bosman
DS-707	Fix/improve DSpace through static code inspections	Unassigned	Graham Triggs
DS-713	dspace.log only logs remoteAddr, needs extending to include X-Forwarded-For	Stuart Lewis	Stuart Lewis
DS-715	Unification of license treatment in xmlui and jspui	Claudia Jürgen	Claudia Jürgen
DS-718	JSPUI redirect users to requested page after authentication		

		Kim Shepherd	Kim Shepherd
DS-719	solr.statistics.logBots isn't documented in dspace.cfg	Kim Shepherd	Kim Shepherd
DS-732	Improve performance of browse	Graham Triggs	Graham Triggs
DS-734	Improve database efficiency	Graham Triggs	Graham Triggs
DS-735	Allow IPAuthentcation to work with proxies (examine X-Forwarded-For header)	Stuart Lewis	Stuart Lewis
DS-736	ItemImport - nicer handling of no contents file, and more efficient handling of no handle file	Graham Triggs	Graham Triggs
DS-738	sword.cfg seems no longer used after 1.5	Stuart Lewis	Bill Hays
DS-739	Improve performance of Lucene indexing	Graham Triggs	Graham Triggs
DS-741	Ability to store an incoming package as a file in the event that the ingest fails	Sands Fish	Sands Fish
DS-750	Cleanup display of Curation Admin UI to make more human readable/understandable	Richard Rodgers	Tim Donohue
DS-770	New Japanese messages for 1.7.0	Claudia Jürgen	Keiji Suzuki
DS-773	Reduce pressure on memory by ensuring that classes with a finalize method make their fields available for garbage collection at earliest possible opportunity	Graham Triggs	Graham Triggs
DS-779	Upgrade steps (1.5->1.6) need reordering	Jeffrey Trimble	Stuart Lewis
DS-783	Typos and confusing text in OpenSearch Support section	Tim Donohue	Jason Stirnaman
DS-784	Incorrect DSpace Statistics section references throughout documentation	Tim Donohue	Jason Stirnaman

16.6.3 Bug Fixes

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 beiing configured to be public	Claudia Jürgen	Claudia Jürgen
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 ArrayIndexOutOfBounds 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		John

		Kim Shepherd	
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
DS-544	Removal of mapped items can lead to NPE	Kim Shepherd	Claudia Jürgen
DS-545	Item Mapper List of Mapped Items Usability	Mark H. Wood	Claudia Jürgen
DS-546	XMLUI 'Notice's are always added by the Administrative aspect even if the content was generated by another aspect.	Robin Taylor	Robin Taylor
DS-547	Value for Recent Submissions is not workin in the XMLUI	Claudia Jürgen	Marvin Pollard
DS-548	Removing repeatable values in DescribeStep does not properly test for authority control	Kim Shepherd	Kim Shepherd
DS-551	Export directories dspace.cfg and build.xml out of sync	Claudia Jürgen	Claudia Jürgen
DS-553	Display date generation only works on dates without time granularity	Robin Taylor	Claudia Jürgen
DS-556	Add Xalan to Solr pom.xml as dependency	Mark Diggory	Mark Diggory
DS-562	Community admin or user with WRITE, ADD and ADMIN policy on collection cannot delete that collection due to bug in AuthorizeUtil.authorizeManageTemplateItem(context,collection)	Andrea Bollini	Andrew Taylor
DS-563	Multiple spaces in between words in advanced search will make the search return nothing	Claudia Jürgen	Flávio Botelho
DS-565	Fixed for Empty description column in Itemview Page - General-Handler.xsl	Kim Shepherd	Yin Yin Latt
DS-566			

	Fixed for side bar menu dropping when there is license text in collection - DIM-Handler.xsl	Kim Shepherd	Yin Yin Latt
DS-569	Trailing white spaces lead to wrong order of texts in browse	Claudia Jürgen	Claudia Jürgen
DS-573	NPE resuming submission for item with an empty bundle original	Andrea Bollini	Andrea Bollini
DS-574	DSpaceMETSIngester creates empty original bundle	Andrea Bollini	Andrea Bollini
DS-579	Required fields in submissions display wrong error message	Ben Bosman	Ben Bosman
DS-580	DIDL format include HTML element if the item has no files	Andrea Bollini	Andrea Bollini
DS-581	DIDL doesn't respect the hidden fields and the oai_dc metadata section is different than the simple oai_dc implementation	Andrea Bollini	Andrea Bollini
DS-583	typo in messages.xml	Unassigned	Ivan Masár
DS-584	start-handle-server script broken - Error in launcher.xml: Invalid class name	Unassigned	Keith Gilbertson
DS-585	The Content Disposition configuration is ignored by unpublished items	Andrea Bollini	Andrea Bollini
DS-594	DCDate parsing thread synchronization issue	Robin Taylor	Andrew Taylor
DS-596	Cannot Delete Community which has two or more levels of SubCommunities	Claudia Jürgen	Tim Donohue
DS-604	Errors in 1.5.x -> 1.6.x and 1.6.0 - 1.6.1 upgrade steps	Jeffrey Trimble	Kim Shepherd
DS-605	Blank screen on XMLUI Dspace 1.6.0	Sands Fish	Antero Neto
DS-607	Invalid identifiers not escaped	Stuart Lewis	Stuart Lewis
DS-608	Batch metadata import missing item headers	Stuart Lewis	Stuart Lewis
DS-614	[XMLUI] Multi-page forms error	Mark H. Wood	Pere Villega

DS-617	[XMLUI] Breadcrumbs dissapear in submission	Mark H. Wood	Pere Villega
DS-619	Item exports not deleted on deletion of eperson	Robin Taylor	Claudia Jürgen
DS-627	Documentation refers to install-configs script	Jeffrey Trimble	Nick Nicholas
DS-629	ItemImporter stops with "Too many open files" error	Keith Gilbertson	Keith Gilbertson
DS-632	Batch Metadata Import needs to validate metadata fields specified in CSVs	Stuart Lewis	Kim Shepherd
DS-633	NPE when deleting object returned by EPerson.findAll()	Mark H. Wood	Mark H. Wood
DS-636	Error Check Needed for handle.prefix in Dspace.cfg	Mark H. Wood	Tim Donohue
DS-637	Browse index bug/fix ONLY for authority index: first "too low" confidence value stop current item metadata to be indexed in the autority index	Andrea Bollini	Reinhard Engels
DS-639	sword swap ingest crosswalk: attribute test for dc.identifier.uri incorrect	Robin Taylor	Bill Hays
DS-644	Broken link in Shibboleth settings in 1.6.2 Documentation	Jeffrey Trimble	Tim Donohue
DS-649	SWORD deposits do not remove temporary package files in the upload directory	Tim Donohue	Bill Hays
DS-654	"Name collision" error in logs for QDCCrosswalk	Tim Donohue	Tim Donohue
DS-656	Minor fix for CommunityTest.	Tim Donohue	Robin Taylor
DS-663	Some OAI-PMH providers will wrap their metadata inside a wrapper element.	Scott Phillips	Scott Phillips
DS-665	SImple search loses it's scope (community or collection) when navigating between search pagination pages.	Scott Phillips	Scott Phillips
DS-666	NPE exception possible when adding a null i18n parameter as content	Scott Phillips	Scott Phillips
DS-677	xmlui BitstreamReader holds database connections open while large files download, exhausting connection pool	Keith Gilbertson	Keith Gilbertson

DS-686	Incorrect use of UTF8 in DSpaceCSV	Stuart Lewis	Stuart Lewis
DS-688	AuthorizeManager.isAdmin(Context, DSpaceObject) throws exception when EPerson is null	Kim Shepherd	Kim Shepherd
DS-689	DisplayStatisticsServlet throws NPE if requested handle is null, should throw 404	Kim Shepherd	Kim Shepherd
DS-691	file size math in the mets:file template in General-Handler.xsl (XMLUI) is incorrect	Kim Shepherd	Hardy Pottinger
DS-698	BitstreamStorageManager fails when deleting primary bitstream (constraint violation)	Graham Triggs	Graham Triggs
DS-709	ArrayIndexOutOfBoundsException when editing group roles	Claudia Jürgen	Andreas Schwander
DS-712	DSpace Fails to Check if a Handle is already assigned before assigning a new handle	Tim Donohue	Tim Donohue
DS-721	RecentSubmissions missing from Collection view (XMLUI trunk @ r5578)	Unassigned	Kim Shepherd
DS-727	JSPUI-style Community and Collection logo URLs do not work in XMLUI	Tim Donohue	Tim Donohue
DS-729	update-sequences.sql script errors out, as it refers to an old, obsolete 'dctyperegistry' table	Tim Donohue	Tim Donohue
DS-742	Embargo Fails with Null Pointer Exception on Item Install.	Mark Diggory	Mark Diggory
DS-743	ItemImport fails on large batches with OutOfMemory exception	Unassigned	Graham Triggs
DS-748	COinS in XMLUI has invalid referrer Id and dc metadata; spans are not properly constructed	Kim Shepherd	Bill Hays
DS-751	Item marked NOT in_archive yields inconsistent METS packages	Mark H. Wood	Mark H. Wood
DS-752	Invalid METS generated for Items with no Bundles	Tim Donohue	Tim Donohue
DS-753	Fix CSV parsing that got broken by refactoring	Stuart Lewis	Stuart Lewis
DS-755	BitstreamReader file descriptor leak	Robin Taylor	Gareth Waller

DS-760	Broken authority-control icons in Mirage theme	Peter Dietz	Peter Dietz
DS-763	StringIndexOutOfBoundsException occurred in DCDate.java	Robin Taylor	Keiji Suzuki
DS-769	Mirage theme: Repeatable oneboxes in submission do not show "Add more" buttons	Ben Bosman	Kim Shepherd
DS-772	AIP Restore process fails to consistently restore Item Mappings	Tim Donohue	Tim Donohue
DS-774	NoClassDefFoundError when running filter media on an item that contains a PDF file	Ben Bosman	Kevin Van de Velde
DS-777	PackageUtils.translateGroupNameForExport returns null unexpectedly	Tim Donohue	Mark H. Wood
DS-786	need to increase version of ojdbc in prerequisites	Tim Donohue	Hardy Pottinger

16.7 Changes in DSpace 1.6.2

16.7.1 General Improvements

Improvements in 1.6.2 **(1 issues)**			
Key	Summary	Assignee	Reporter
DS-595	Czech localization of 1.6.1	Claudia Jürgen	Ivan Masár

16.7.2 Bug Fixes

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

16.8 Changes in DSpace 1.6.1

16.8.1 General Improvements

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

16.8.2 Bug Fixes

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
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 ArrayIndexOutOfBounds 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
DS-542	verbose output for stats-log-importer displays spurious city/country from previous committed entry	Peter Dietz	Peter Dietz
DS-543	Harvest not internationalized	Claudia Jürgen	Claudia Jürgen
DS-544	Removal of mapped items can lead to NPE	Kim Shepherd	Claudia Jürgen
DS-547	Value for Recent Submissions is not workin in the XMLUI	Claudia Jürgen	Marvin Pollard
DS-548	Removing repeatable values in DescribeStep does not properly test for authority control	Kim Shepherd	Kim Shepherd
DS-551	Export directories dspace.cfg and build.xml out of sync	Claudia Jürgen	Claudia Jürgen
DS-556	Add Xalan to Solr pom.xml as dependency	Mark Diggory	Mark Diggory
DS-558	Error in update sequence script 1.5 to 1.6 Oracle	Claudia Jürgen	Claudia Jürgen
DS-565	Fixed for Empty description column in Itemview Page - General-Handler.xsl	Kim Shepherd	Yin Yin Latt
DS-566	Fixed for side bar menu dropping when there is license text in collection - DIM-Handler.xsl	Kim Shepherd	Yin Yin Latt
DS-572	Batch metadata editor fails to notice change of item's owning collection	Jeffrey Trimble	Stuart Lewis
DS-573	NPE resuming submission for item with an empty bundle original	Andrea Bollini	Andrea Bollini
DS-574	DSpaceMETSIngester creates empty original bundle	Andrea Bollini	Andrea Bollini
DS-579	Required fields in submissions display wrong error message	Ben Bosman	Ben Bosman

DS-580	DIDL format include HTML element if the item has no files	Andrea Bollini	Andrea Bollini
DS-581	DIDL doesn't respect the hidden fields and the oai_dc metadata section is different than the simple oai_dc implementation	Andrea Bollini	Andrea Bollini
DS-610	SyndicationFeed expects dc.date.issued to be available as a java.util.Date	Robin Taylor	Robin Taylor

16.9 Changes in DSpace 1.6.0

16.9.1 New Features

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
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

16.9.2 General Improvements

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 descritpion.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
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	Stuart Lewis	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
DS-315	Enhance readability of embedded metadata in html head	Stuart Lewis	Stuart Lewis
DS-316	Make SWORD app:accepts configurable	Jeffrey Trimble	Stuart Lewis

DS-319	Replace '/dspace/bin/dsrun org.dspace.browse.ItemCounter' with /dspace/bin/itemcounter	Jeffrey Trimble	Stuart Lewis
DS-333	Adjust SWORD ingest crosswalk to store bibliographic citation	Stuart Lewis	Stuart Lewis
DS-335	Minho Statistics Database Manager additions.	Mark Diggory	Mark Diggory
DS-336	DSpace Services 2.0 Support	Mark Diggory	Mark Diggory
DS-347	Add --quiet option to MediaFilterManager to disable debug/monitoring output	Larry Stone	Larry Stone
DS-356	Antispam for suggest item feature	Unassigned	Toni Prieto
DS-361	Merge + Improve Generation of Syndication Feeds	Larry Stone	Larry Stone
DS-364	Script to convert legacy dspace.log stats into solr stats records	Jeffrey Trimble	Stuart Lewis
DS-369	Implement harvest.includerestricted.rss in xmlui	Unassigned	Stuart Lewis
DS-372	New verbose option for [dspace]/bin/dspace cleanup script	Claudia Jürgen	Stuart Lewis
DS-382	Proposal: Add 'dc.creator' to Author browse index by default	Tim Donohue	Tim Donohue
DS-386	Allow user to specify which <dmdSec> is used by the METS Ingester when importing METS from Packager script	Tim Donohue	Tim Donohue
DS-389	Misleading label: "Submit to This Collection"	Stuart Lewis	Stuart Lewis
DS-407	Install or Upgrade on existing server throws errors for "mvn package"	Jeffrey Trimble	Tim Donohue
DS-410	Updates to Upgrade Instructions necessary	Jeffrey Trimble	Tim Donohue
DS-412	Xpdf MediaFilter: generate UTF-8 text, and improve error reporting	Larry Stone	Larry Stone
DS-425	JSP UI cosmetics: horizontal scroll bar	Kim Shepherd	Bram Luyten (@mire)
DS-442	Enable RSS feeds by default	Jeffrey Trimble	Stuart Lewis
DS-445	New Bitstream.findAll() method		Stuart Lewis

		Stuart Lewis	
DS-446	New ant step - test_database	Jeffrey Trimble	Stuart Lewis
DS-455	Remove dspace/config/log4j.xml	Jeffrey Trimble	Stuart Lewis
DS-460	Change logging from RollingFileAppender to DailyFileAppender	Jeffrey Trimble	Stuart Lewis
DS-461	Add information about setting web proxies for maven to install docs	Jeffrey Trimble	Stuart Lewis
DS-567	Czech localization of 1.6.0	Claudia Jürgen	Ivan Masár

16.9.3 Bug Fixes

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
DS-114	Links not working due to trailing white space in dspace.url	Claudia Jürgen	Claudia Jürgen
DS-118	File preview link during submission leeds 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
DS-216	Migrating items that use additional metadata schemas causes an NPE	Unassigned	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
DS-223	Submission process show previous button in JSPUI also if the step is the first "visible" step	Andrea Bollini	Andrea Bollini
DS-225	dc.description.provenance - public display	Stuart Lewis	Janice Silveira
DS-226	confirmation page of edit profile has an invalid link	Ben Bosman	Ben Bosman
DS-227	Values with double apos doesn't work in dropdown and list input type	Andrea Bollini	Andrea Bollini
DS-231	Missig file	Claudia Jürgen	Fred Dubiel
DS-232	DCPersonName parses name incorrectly (fix included)	Larry Stone	Larry Stone
DS-240	Item validityKey not complete	Unassigned	Stuart Lewis
DS-246	Fix configurable browse parameter encoding (XMLUI)	Unassigned	Kim Shepherd

DS-248	Missing admin column in community table in database-schema.sql - community admin patch	Claudia Jürgen	Claudia Jürgen
DS-249	sub-daily utility script does not pass arguments to Java (fix included)	Stuart Lewis	Larry Stone
DS-250	Invalid identifers are not escaped	Stuart Lewis	Stuart Lewis
DS-253	NullPointerException in HttpServletResponseBufferingWrapper (Cocoon bug?)	Mark Diggory	Mark H. Wood
DS-254	Bitstream (and item-export) download service does not correctly sense authenticated user	Larry Stone	Larry Stone
DS-255	CompleteStep in submission LOSES SUBMISSION if an exception is thrown	Larry Stone	Larry Stone
DS-256	Item Export ignores metadata language qualifier	Stuart Lewis	Claudia Jürgen
DS-258	Item View Thumbnails not displaying in XMLUI	Stuart Lewis	Steven Williams
DS-260	Template item some times has owningCollection filled and some times not	Andrea Bollini	Andrea Bollini
DS-262	Bug in DS-118, new patch included	Stuart Lewis	Larry Stone
DS-264	XMLUI misses logging UsageEvent on requests fulfilled from the cache (with proposed fix)	Mark Diggory	Larry Stone
DS-265	IndexBrowse dies fatally when confronting badly-formatted date	Graham Triggs	Samuel Ottenhoff
DS-269	Oracle JDBC connection string wrong in dspace.cfg - ID: 2722093	Claudia Jürgen	Claudia Jürgen
DS-274	Typo in XSL breaks rendering of dri:xref with class	Mark H. Wood	Larry Stone
DS-275	License files not listed on Item Summary page; XSL bug with patch	Stuart Lewis	Larry Stone
DS-276	Patch to fix spelling error in Exception page	Claudia Jürgen	Larry Stone
DS-280	build.xml fails for ant versions below 1.7 (patch included)	Jeffrey Trimble	Larry Stone
DS-281	Invalid Link to "Go to DSpace Home" on Page Not Found	Tim Donohue	Tim Donohue
DS-282	"Starts with" navigation block should not display when browsing by specific value	Unassigned	Samuel Ottenhoff

DS-284	Some rows if presented in the item summary will be wrongly considered odd or even.	Larry Stone	Flávio Botelho
DS-285	Item and Bitstream pages do not provide Last-Modified HTTP header, nor recognize If-Modified-Since	Larry Stone	Larry Stone
DS-290	[dspace]/exports is not created during fresh install	Stuart Lewis	Stuart Lewis
DS-303	Export migrate option incorrectly removes non-handle identifier.uris	Stuart Lewis	Stuart Lewis
DS-309	Shiboleth default roles are applied also to anonymous user and user logged-in with other methods	Andrea Bollini	Andrea Bollini
DS-310	UTF-8 encoding in community and collection text	Ben Bosman	Ben Bosman
DS-318	JSPUI: Left over text in edit item about format	Stuart Lewis	Stuart Lewis
DS-320	java.util.NoSuchElementException: Timeout waiting for idle objec	Unassigned	Flávio Botelho
DS-327	SWORD temp upload directory missing trailing slash	Stuart Lewis	Stuart Lewis
DS-328	SWORD service documents do not include atom:generator element	Stuart Lewis	Stuart Lewis
DS-337	A bug related with adding new -EPersons	Claudia Jürgen	Eija Airio
DS-338	Bitstream download allows caching of content that requires authorization to read	Larry Stone	Larry Stone
DS-340	DSpace services log to the command line	Mark Diggory	Stuart Lewis
DS-344	Apostrophe in email address prevents EPerson from being selected	Graham Triggs	Graham Triggs
DS-349	Edit Item in admin UI does not allow setting Bitstream to an Internal BitstreamFormat	Larry Stone	Larry Stone
DS-353	Missing commits in XMLUI server-side javascript code.	Unassigned	Flávio Botelho
DS-354	Make-handle-server configuration fails. Using manual commands instead of script is successful.	Jeffrey Trimble	Jeffrey Trimble
DS-365	New DSpace OAI-PMH Harvester doesn't support OAI gateways that do not use "sets"	Ben Bosman	Mark Diggory
DS-370	E Mail Sent On Item Export Error Message	Unassigned	Hardik Mishra
DS-373	"Letter" links have broken URLs in 2nd-stage Browse	Larry Stone	Larry Stone

DS-378	XMLUI Submission Interface messes up in IE7 after an empty <hint> in input_forms.xml	Tim Donohue	Tim Donohue
DS-379	open-search in jspui won't return description.xml	Richard Rodgers (OLD acct)	Stuart Lewis
DS-381	community and collection homepage	Ben Bosman	Ben Bosman
DS-385	Packager script is unable to import the same METS + DIM package that was exported	Tim Donohue	Tim Donohue
DS-392	Error messages in the submission do not disappear if e.g. one of the two errors are solved	Ben Bosman	Ben Bosman
DS-393	The issue date in the submission lowers each time the describe page is being displayed	Larry Stone	Ben Bosman
DS-395	DSpace Objects (communities, collections, items, bitstreams) only accessible to logged in users	Tim Donohue	Claudia Jürgen
DS-398	Submission license displayed on collection and item homepage	Tim Donohue	Claudia Jürgen
DS-399	Special characters in collection license lead to parse error	Tim Donohue	Claudia Jürgen
DS-400	Webui item browse (date, title or similar) reduces displayed issue date by one day	Larry Stone	Claudia Jürgen
DS-406	View Statistics button does not work in item page	Kim Shepherd	Dan Ishimitsu
DS-409	JSPUI Statistics Display ignores "statistics.item.authorization.admin"	Kim Shepherd	Kim Shepherd
DS-414	solr statistics file downloads listed in statistics display of communites and collections	Kim Shepherd	Claudia Jürgen
DS-415	Create groups via admin UI authorization denied	Andrea Bollini	Claudia Jürgen
DS-418	i18n broken in jspui	Kim Shepherd	Claudia Jürgen
DS-419	Setting embargo.field.terms to an unqualified field throws uncaught exception on item submission	Larry Stone	Scott Hanrath
DS-421			Scott Hanrath

	Setting solr.metadata.item.X property to an unqualified field generates exception in SolrLogger.post on item view and prevents Solr from logging the event	Mark Diggory	
DS-422	Directory 'etc' missing from Ant target 'init_installation'.	Richard Rodgers (OLD acct)	Robin Taylor
DS-423	Ant target 'clean_database' doesn't drop all tables'.	Tim Donohue	Robin Taylor
DS-424	Export metadata button displayed in JSPUI Administration List of withdrawn items	Stuart Lewis	Claudia Jürgen
DS-427	Item license per default displayed in item display of the xmlui	Richard Rodgers (OLD acct)	Claudia Jürgen
DS-428	Wrong link for bitstreams during submission	Unassigned	Flávio Botelho
DS-432	No mention of config/news-xmlui.xml in manual	Jeffrey Trimble	Larry Stone
DS-436	SWORD Authenticator doesn't support the special groups infrastructure	Andrea Bollini	Andrea Bollini
DS-437	Oracle DB Schema has artifacts from past releases	Larry Stone	Larry Stone
DS-438	JSPUI stats - filename incorrect on second and subsequent files	Kim Shepherd	Stuart Lewis
DS-440	spiders.txt empty	Mark Diggory	Stuart Lewis
DS-459	OAI PMH is not delivering continuation tokens	Unassigned	Mark Diggory
DS-468	CLONE - Foreign characters broken in group names.	Kim Shepherd	Flávio Botelho
DS-472	'fresh_install' broken on a completely clean system	Stuart Lewis	Graham Triggs
DS-474	handle.canonical.prefix undocumented	Kim Shepherd	Fabien COMBERNOUS
DS-477	LDAPHierarchicalAuthentication fails when the LDAP returns mixed case email address	Graham Triggs	Graham Triggs
DS-480	Exception is thrown when removing the last file after the item is rejected during review.	Mark H. Wood	Flávio Botelho

DS-597	Once added, the description of a bitstream can not be removed	Unassigned	Eija Airio
DS-610	SyndicationFeed expects dc.date.issued to be available as a java.util.Date	Robin Taylor	Robin Taylor

16.10 Changes in DSpace 1.5.2

16.10.1 New Features

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

16.10.2 General Improvements

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
DS-78	Italian translation xmlui - ID: 1984513	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
DS-95	Visiting an item page with a trailing slash returns a 404 message	Tim Donohue	Tim Donohue
DS-97	XMLUI does not come with a robots.txt out-of-the-box. It also does not provide suggestions for loading global static content.	Tim Donohue	Tim Donohue

DS-100	XMLUI OpenURL support	Mark Diggory	Mark Diggory
DS-109	Consistent treatment to users in special groups	Andrea Bollini	Bruc Liong
DS-127	Item exporter does not export bitsteam descriptions	Stuart Lewis	Stuart Lewis
DS-144	Patch for Feature Request 2609564 group delete confirm - ID: 2612341	Claudia Jürgen	Charles Kiplagat
DS-145	messages_el.xml for DSpace 1.5.1 XMLUI - ID: 2615647	Claudia Jürgen	Charles Kiplagat
DS-146	Feature Request 2560839 Make sitemap directory configurable - ID: 2560974	Claudia Jürgen	Charles Kiplagat
DS-147	Messages_th.properties for DSpace 1.5.1 JSPUI - ID: 2540683	Claudia Jürgen	Charles Kiplagat
DS-148	Updated Messages_th.properties for DSpace 1.4.2 - ID: 2537799	Claudia Jürgen	Charles Kiplagat
DS-150	Usage event (statistics) plugin hook for 1.5 - ID: 2025998	Mark H. Wood	Charles Kiplagat
DS-154	Make a backup of the config dir on Ant update	Mark Diggory	Andrea Bollini
DS-174	LDAP_Support for hierarchical LDAP servers	Unassigned	Charles Kiplagat
DS-182	Speed up SWORD deposits	Stuart Lewis	Stuart Lewis
DS-183	XPDF support for filtering PDFs for text extraction/search.	Mark Diggory	Mark Diggory
DS-375	[Dspace 1.5.2] Russian translation	Claudia Jürgen	Roman
DS-515	Czech localization of 1.5.2	Claudia Jürgen	Claudia Jürgen
DS-582	Basque translation	Claudia Jürgen	Javier Garrido

16.10.3 Bug Fixes

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
DS-10	Implicit group for all registered users - ID: 1587270	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
DS-28	Itemmap-browse.jsp throws Exception on items without date - ID: 1745573	Claudia Jürgen	Charles Kiplagat
DS-29	Imporper display of Umlauts / Encoding of messages_de.xml - ID: 2413800	Unassigned	Charles Kiplagat
DS-32	Export Item email hardcoded strings in sources - ID: 2411653	Claudia Jürgen	Charles Kiplagat
DS-33	checksum checker can not retrieve very large bitstream - ID: 2016130	Unassigned	Charles Kiplagat
DS-37	Submission Forms don't preserve order of values - ID: 2541285	Unassigned	Charles Kiplagat
DS-38	XMLUI Submission forms display errors during add/remove - ID: 2543413	Unassigned	Charles Kiplagat
DS-40	SubmissionController is not thread safe	Andrea Bollini	Andrea Bollini
DS-41	Small JSP cleanups	Mark H. Wood	Mark H. Wood
DS-43	Manakin RSS feed generator cache timeout can't be adjusted - ID: 2593393	Andrea Bollini	Charles Kiplagat
DS-58	Double quote problem in some fields of JSPUI (continued) - ID: 2528065	Unassigned	Andrea Bollini
DS-59	Item Export via UI leads to malformed request - ID: 2520279	Stuart Lewis	Andrea Bollini

DS-60	netid.toLowerCase() in different classes and methods - ID: 2458187	Unassigned	Andrea Bollini
DS-61	IP Authentication only works on logged in users - ID: 2088431	Unassigned	Andrea Bollini
DS-62	Not right encoding for license and extracted texts - ID: 2234532	Andrea Bollini	Andrea Bollini
DS-64	xmlui selection of authentication method - ID: 2086673	Andrea Bollini	Andrea Bollini
DS-66	bitstream format registry, setting bitstream internal - ID: 1896055	Andrea Bollini	Andrea Bollini
DS-67	Impossible retry/resume after upload a file with internal format	Andrea Bollini	Andrea Bollini
DS-81	Subscription not sent correctly - ID: 2667590	Andrea Bollini	Andrea Bollini
DS-83	Hardcoded behaviour of Initial question step in the submission	Andrea Bollini	Andrea Bollini
DS-84	Improve API for "IgnoreAuthorization" blocks - ID: 1687783	Unassigned	Andrea Bollini
DS-86	XMLUI Feedback form does not include any protection from spamming	Tim Donohue	Tim Donohue
DS-88	Anonymous group don't works as subgroup - ID: 1688445	Andrea Bollini	Andrea Bollini
DS-90	Workflow doesn't skipped if the wf group contains empty subgroups	Andrea Bollini	Andrea Bollini
DS-92	Bitstream access rights inheritenc, editing in_archive items - ID: 1993036	Andrea Bollini	Andrea Bollini
DS-96	XMLUI Wing-Framework 'Radio' class doesn't support adding radio buttons which are unselected by default.	Tim Donohue	Tim Donohue
DS-98	Key jsp.mydspace.request.export.community not set in Messages.properties	Stuart Lewis	Stuart Lewis
DS-99	Non-admin user and admin menu options (1.5.1 XMUI only) - ID: 2353606	Andrea Bollini	Stuart Lewis
DS-101	Impossible to complete Registration in JSPUI	Andrea Bollini	Claudia Jürgen

DS-102	Submission without file possible with the skip upload functionality disabled	Andrea Bollini	Andrea Bollini
DS-103	Use of the Progress bar button don't save the current page modification	Andrea Bollini	Andrea Bollini
DS-104	Cancel/save button during edit of a workflow item doesn't work	Andrea Bollini	Andrea Bollini
DS-105	Initial Questions button in the progress bar doesn't work in workflow	Unassigned	Andrea Bollini
DS-106	Save & exit button doesn't properly work in workflow	Unassigned	Andrea Bollini
DS-107	DIDL Crosswalk thrown NPE	Andrea Bollini	Andrea Bollini
DS-112	Scandinavian caharacters mess up Front page full text search	Tim Donohue	Mika Stenberg
DS-113	Export item buttons	Unassigned	Nilde De Paoli
DS-115	metadata not showed in verify step	Andrea Bollini	Nilde De Paoli
DS-116	File description not correct	Andrea Bollini	Nilde De Paoli
DS-117	File uploads >=100KB lead to an error in the xmlui	Unassigned	Claudia Jürgen
DS-120	Cocoon 2.2 "hijacks" the dspace.log file	Tim Donohue	Tim Donohue
DS-122	Problems with repeatable metadata	Andrea Bollini	Nilde De Paoli
DS-126	Request to [dspace-base-url]/bitstream throw NPE	Andrea Bollini	Andrea Bollini
DS-129	OpenURL support exposes dc.description.provenance information	Mark Diggory	Claudia Jürgen
DS-130	XMLUI Browse by Author doesn't work for names with special characters (for example: é, è, ö, etc.)	Tim Donohue	Tim Donohue
DS-131		Unassigned	

	Browse By Title displays all titles beginning with punctuation at the top of the list		Tim Donohue
DS-132	XMLUI overall UTF-8 encoding is inconsistent and forms do not use UTF-8	Tim Donohue	Tim Donohue
DS-138	Authors re-ordered when item edited - ID: 2655052	Stuart Lewis	Charles Kiplagat
DS-143	'ant init_configs' copies wrong dspace.cfg - ID: 2620307	Mark H. Wood	Charles Kiplagat
DS-149	Search with "%" causes NullPointer in XMLUI - ID: 2557654	Tim Donohue	Charles Kiplagat
DS-151	XMLUI Submission forms display errors during add/remove - ID: 2543413	Tim Donohue	Charles Kiplagat
DS-152	Submission verify page handles dc.identifier.* incorrectly - ID: 2155479	Tim Donohue	Charles Kiplagat
DS-153	Submission Forms don't preserve order of values - ID: 2541285	Tim Donohue	Charles Kiplagat
DS-155	Maven war plugin changes cause war to include libraries as well	Mark Diggory	Mark Diggory
DS-185	bin files in DOS format	Unassigned	Andrea Bollini
DS-186	NPE during edit of eperson in XMLUI	Andrea Bollini	Andrea Bollini

16.11 Changes in DSpace 1.5.1

16.11.1 General Improvements and Bug Fixes

(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 durring 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.xsl
- 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&quot
- Fixed Oracle bugs with ILIKE operators and LIMIT/OFFSET clauses

16.12 Changes in DSpace 1.5

16.12.1 General Improvements

- Highly configurable and theme-able new user interface (Manakin).
- Apache Maven-based modular build system.
- LNI (Lightweight Network Interface) service. Allows programmatic ingest of content via WebDAV or SOAP.
- SWORD (Simple Web-service Offering Repository Deposit): repository-standard ingest service using Atom Publishing Protocol.
- Highly configurable item web submission system. All submission steps are configurable not just metadata pages.
- Browse functionality allowing customisation of the available indexes via dspace.cfg and pluggable normalisation of the sort strings. Integration with both JSP-UI and XML-UI included.
- Extensible content event notification service.
- Generation of Google and HTML sitemaps

16.12.2 Bug fixes and smaller patches

- New options for ItemImporter to support bitstream permissions and descriptions.
- 1824710 Fix - Change in Creative Commons RDF.
- 1794700 Fix - Stat-monthly and stat-report-monthly
- 1566820 Patch - Authentication code moved to new org.dspace.authenticate package, add IP AUth
- 1670093 Patch - More stable metadata and schema registry import Option to generate community and collection "strength" as a batch job
- 1659868 Patch - Improved database level debugging
- 1620700 Patch - Add Community and Sub-Community to OAI Sets
- 1679972 Fix - OAIDCCrosswalk NPE and invalid character fix, also invalid output prevented
- 1549290 Fix - Suggest Features uses hard coded strings
- 1727034 Fix - Method MetadataField.unique() is incorrect for null values
- 1614546 Fix - Get rid of unused mets_bitstream_id column
- 1450491 Patch - i18n configurable multilingualism support
- 1764069 Patch - Replace "String" with "Integer" in PreparedStatement where needed
- 1743188 Patch - for Request #1145499 - Move Items
- 179196 Patch - Oracle SQL in Bitstream Checker
- 1751638 Patch - Set http disposition header to force download of large bitstreams
- 1799575 Patch - New EPersonConsumer event consumer
- 1566572 Patch - Item metadata in XHTML head elements
- 1589429 Patch - "Self-Named" Media Filters (i.e. MediaFilter Plugins) (updated version of this patch)
- 1888652 Patch - Statistics Rewritten In Java
- 1444364 Request - Metadata registry exporter
- 1221957 Request - Admin browser for withdrawn items
- 1740454 Fix - Concurrency

- 1552760 Fix - Submit interface looks bad in Safari
- 1642563 Patch - bin/update-handle-prefix rewritten in Java
- 1724330 Fix - Removes "null" being displayed in community-home.jsp
- 1763535 Patch - Alert DSpace administrator of new user registration
- 1759438 Patch - Multilingualism Language Switch - DSpace Header

16.13 Changes in DSpace 1.4.1

16.13.1 General Improvements

- Error pages now return appropriate HTTP status codes (e.g. 404 not found)
- Bad filenames in /bitstream/ URLs now result in 404 error – prevents infinite URL spaces confusing crawlers and bad "persistent" bitstream IDs circulating
- Prevent infinite URL spaces in HTMLServlet
- InstallItem no longer sets dc.format.extent, dc.format.mimetype; no longer sets default value for dc.language.iso if one is not present
- Empty values in drop-down submit fields are not added as empty metadata values
- API methods for searching epeople and groups
- Support stats from both 1.3 and 1.4
- [dspace]/bin/update-handle-prefix now runs index-all
- Remove cases of System.out from code executed in webapp
- Change "View Licence" to "View License" in Messages.properties
- dspace.cfg comments changed to indicate what default.language actually means
- HandleServlet and BitstreamServlet support If-Modified-Since requests
- Improved sanity-checking of XSL-based ingest crosswalks
- Remove thumbnail filename from alt-text
- Include item title in HTML title element
- Improvements to help prevent spammers and sploggers
- Make cleanup() commit outstanding work every 100 iterations
- Better handling where email send failed due to wrong address for new user
- Include robots.txt to limit bots navigating author, date and browse by subject pages
- Add css styles for print media
- RSS made more configurable and provide system-wide RSS feed, also moves text to Messages.properties
- Jar file updates (includes required code changes for DSIndexer and DSQuery and new jars fontbox.jar and serializer.jar)
- Various documentation additions and cleanups
- XHTML compliance improvements
- Move w3c valid xhtml boiler image into local repository
- Remove uncessary Log4j Configuration in CheckerCommand
- Include Windows CLASSPATH in dsrun.bat

16.13.2 Bug fixes

- 1604037 - UIUtil.encodeBitstream() now correctly encodes URLs (no longer incorrectly substitutes '+' for spaces in non-query segment
- 1592984 - Date comparisons strip time in org.dspace.harvest.Harvest
- 1589902 - Duplicate field checking error on input-forms.xml
- 1596952 - Collection Wizard create Template missing schema
- 1596978 - View unfinished submissions - collection empty
- 1588625 - Incorrect text on item mapper screen
- 1597805 - DIDL Crosswalk: wrong resource management
- 1605635 - NPE in Utils.java
- 1597504 - Search result page shows shortened query string
- 1532389 - Item Templates do not work for non-dc fields
- 1066771 - Metadata edit form dropping DC qualifier
- 1548738 - Multiple Metadata Schema, schema not shown on edit item page
- 1589895 - Not possible to add unqualified Metadata Field
- 1543853 - Statistics do not work in 1.4
- 1541381 - Browse-by-date and browse-by-title not working
- 1556947 - NullPointerException when no user selected to del/edit
- 1554064 - Fix exception handling for ClassCastException in BitstreamServlet
- 1548865 - Browse errors on withdrawn item
- 1554056 - Community/collection handle URL with / redirects to homepage
- 1571490 - UTF-8 encoded characters in licence
- 1571519 - UTF-8 in statistics
- 1544807 - Browse-by-Subject/Author paging mechanism broken
- 1543966 - "Special" groups inside groups bug
- 1480496 - Cannot turn off "ignore authorization" flag!
- 1515148 - Community policies not deleting correctly
- 1556829 - Docs mention old SiteAuthenticator class
- 1606435 - Workflow text out of context
- Fix for bitstream authorization timeout
- Fix to make sure cleanup() doesn't fail with NullPointerException
- Fix for removeBitstream() failing to update primary bitstream
- Fix for Advanced Search ignoring conjunctions for arbitrary number of queries
- Fix minor bug in Harvest.java for Oracle users
- Fix missing title for news editor page
- Small Messages.properties modification (change of DSpace copyright text)
- fix PDFBox tmp file issue
- Fix HttpServletRequest encoding issues
- Fix bug in TableRow toString() method where NPE is thrown if tablename not set
- Update DIDL license and change coding style to DSpace standard

16.14 Changes in DSpace 1.4

16.14.1 General Improvements

- Content verification through periodic checksum checking
- Support for branded preview image
- Add/replace Creative Commons in 'edit item' tool
- Customisable item listing columns and browse indices
- Script for updating handle prefixes (e.g. for moving from development to production)
- Configurable boolean search operator
- Controlled vocabulary patch to provide search on classification terms, and addition of terms during submission.
- Add 'visibility' element to input-forms.xml
- Browse by subject feature
- Log4J enhancement to use XML configuration
- QueryArgs class can support any number of fields in advanced search.
- Community names no longer have to be unique
- Enhanced Windows support
- Support for multiple (flat) metadata schemas
- Suggest an item page
- RSS Feeds
- Performance enhancements
- Stackable authentication methods
- Plug-in manager
- Pluggable SIP/DIP support and metadata crosswalks
- Nested groups of e-people
- Expose METS and MPEG-21 DIDL DIPs via OAI-PMH
- Configurable Lucene search analyzer (e.g. for Chinese metadata)
- Support for SMTP servers requiring authentication

16.14.2 Bug fixes

- 1358197 - Edit Item, empty DC fields not removable
- 1363633 - Submission step 1 fails when there are no collections
- 1255264 - Resource policy eperson value was set to wrong column
- 1380494 - Error deleting an item with multiple metadata schema support
- 1443649 - Cannot configure unqualified elements for advanced search index
- 1333687 - Browse-(title|date) fails on withdrawn item
- 1066713 - Two (sub)communities cannot have one name

- 1284055 - Two Communities of same name throws error
- 1035366 - Bitstream size column should be bigint
- 1352257 - Selecting a Group for GroupToGroup while Creating Collection
- 1352226 - Navigation and Sorting in Group List (Select Groups) fails
- 1348276 - Null in collection name causes OAI ListSets to fail
- 1160898 - dspace_migrate removes Date.Issued from prev published items
- 1261191 - Malformed METS metadata exported

16.15 Changes in DSpace 1.3.2

16.15.1 General Improvements

- DSpace UI XHTML/WAI compliant
- Configure metadata fields shown on simple item display
- Supervisor/workspace help documentation

16.15.2 Bug fixes

- Oracle compatibility fixes
- Item exporter now correctly exports metadata in UTF-8
- fixed to handle 'null' values passed in

16.16 Changes in DSpace 1.3.1

16.16.1 Bug fixes

- 1252153 - Error on fresh install

16.17 Changes in DSpace 1.3

16.17.1 General Improvements

- Initial i18n Support for JSPs - Note: the implementation of this feature required changes to almost all JSP pages
- LDAP authentication support
- Log file analysis and report generation

- Configurable item licence viewing
- Supervision order/collaborative workspace administrative tools
- Basic workspace for submissions in progress, with support for supervision
- SRB storage system option
- Updated handle server system
- Database optimisations
- Latest versions of Xerces, Xalan and OAICAT jars
- Various documentation additions and cleanups

16.17.2 Bug fixes

- 1161459 - ItemExporter fails with Too many open files
- 1167373 - Email date field not populated
- 1193948 - New item submit problem
- 1188132 - NullPointerException when Adding EPerson
- 1188016 - Cannot Edit an Eperson
- 1219701 - Unable to open unfinished submission
- 1206836 - community strengths not reflecting sub-community
- 1238262 - Submit UI nav/progress buttons no longer show progress
- 1238276 - Double quote problem in some fields in submit UI
- 1238277 - format support level not shown in "uploaded file" page
- 1242548 - Uploading non-existing files
- 1244743 - Bad lookup key for special case of DC Title in ItemTag.java
- 1245223 - Subscription Emailer fails
- 1247508 - Error when browsing item with no content/bitstream collections
- Set the content type in the HTTP header
- Fix issue where EPerson edit would not work due to form indexing (partial fix)
- POST handling in HTMLServlet
- Missing ContentType directives added to some JSPs
- Name dependency on Collection Admin and Submitter groups fixed
- Fixed OAI-PMH XML encoding

16.18 Changes in DSpace 1.2.2

16.18.1 General Improvements

- Customisable submission forms added
- Configurable number of index terms in Lucene for full-text indexing
- Improved scalability in media filter
- Submit button on collection pages only appears if user has authorisation

- PostgreSQL 8.0 compatibility
- Search scope retention to improve browsing
- Community and collection strengths displayed
- Upgraded OAICat software

16.18.2 Bug fixes

- Fix for Oracle too many cursors problem.
- Fix for UTF-8 encoded searches in advanced search.
- Fix for handling "\" in bitstream names.
- Fix to prevent delete of "unknown" bitstream format
- Fix for ItemImport creating new handles for replaced items

16.18.3 Changes in JSPs

- *collection-home.jspchanged*
- *community-home.jspchanged*
- *community-list.jspchanged*
- *home.jspchanged*
- *dspace-admin/list-formats.jspchanged*
- *dspace-admin/wizard-questions.jspchanged*
- *search/results.jspchanged*
- *submit/cancel.jspchanged*
- *submit/change-file-description.jspchanged*
- *submit/choose-file.jspchanged*
- *submit/complete.jspchanged*
- *submit/creative-commons.jspchanged*
- *submit/edit-metadata.jspnew*
- *submit/get-file-format.jspchanged*
- *submit/initial-questions.jspchanged*
- *submit/progressbar.jspchanged*
- *submit/review.jspchanged*
- *submit/select-collection.jspchanged*
- *submit/show-license.jspchanged*
- *submit/show-uploaded-file.jspchanged*
- *submit/upload-error.jspchanged*
- *submit/upload-file-list.jspchanged*

16.19 Changes in DSpace 1.2.1

16.19.1 General Improvements

- Oracle support added
- Thumbnails in item view can now be switched off/on
- Browse and search thumbnail options
- Improved item importer
  - can now import to multiple collections
  - added --test flag to simulate an import, without actually making any changes
  - added --resume flag to try to resume the import in case the import is aborted
- Configurable fields for the search index
- Script for transferring items between DSpace instances
- Sun library JARs (JavaMail, Java Activation Framework and Servlet) now included in DSpace source code bundle

16.19.2 Bug fixes

- A logo to existing collection can now be added. Fixes SF bug #1065933
- The community logo can now be edited. Fixes SF bug #1035692
- MediaFilterManager doesn't 'touch' every item every time. Fixes SF bug #1015296
- Supported formats help page, set the format support level to "known" as default
- Fixed various database connection pool leaks

16.19.3 Changed JSPs

- *collection-homechanged*
- *community-homechanged*
- *display-itemchanged*
- *dspace-admin/confirm-delete-collectionmoved to tools/ and changed*
- *dspace-admin/confirm-delete-communitymoved to tools/ and changed*
- *dspace-admin/edit-collectionmoved to tools/ and changed*
- *dspace-admin/edit-communitymoved to tools/ and changed*
- *dspace-admin/indexchanged*
- *dspace-admin/upload-logochanged*
- *dspace-admin/wizard-basicinfochanged*
- *dspace-admin/wizard-default-itemchanged*
- *dspace-admin/wizard-permissionschanged*
- *dspace-admin/wizard-questionschanged*
- *help/formats.htmlremoved*
- *help/formatschanged*
- *indexchanged*

- *layout/navbar-adminchanged*

16.20 Changes in DSpace 1.2

16.20.1 General Improvments

- Communities can now contain sub-communities
- Items may be included in more than one collection
- Full text extraction and searching for MS Word, PDF, HTML, text documents
- Thumbnails displayed in item view for items that contain images
- Configurable MediaFilter tool creates both extracted text and thumbnails
- Bitstream IDs are now persistent - generated from item's handle and a sequence number
- Creative Commons licenses can optionally be added to items during web submission process

16.20.2 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)

16.20.3 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

16.20.4 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

16.20.5 JSP file changes between 1.1 and 1.2

This list generated with *cvs -Q rdiff -s -r dspace-1_1 dspace* and a sprinkling of perl.

- Changed: dspace/jsp/collection-home.jsp
- Changed: dspace/jsp/community-home.jsp
- Changed: dspace/jsp/community-list.jsp
- Changed: dspace/jsp/display-item.jsp
- Changed: dspace/jsp/index.jsp
- Changed: dspace/jsp/home.jsp
- Changed: dspace/jsp/styles.css.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/authorize-advanced.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/authorize-collection-edit.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/authorize-community-edit.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/authorize-item-edit.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/authorize-main.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/authorize-policy-edit.jsp
- Moved to dspace-admin: dspace/jsp/admin/collection-select.jsp
- Moved to dspace-admin: dspace/jsp/admin/community-select.jsp
- Moved to dspace-admin: dspace/jsp/admin/confirm-delete-collection.jsp
- Moved to dspace-admin: dspace/jsp/admin/confirm-delete-community.jsp
- Moved to dspace-admin: dspace/jsp/admin/confirm-delete-dctype.jsp
- Moved to dspace-admin: dspace/jsp/admin/confirm-delete-eperson.jsp
- Moved to dspace-admin: dspace/jsp/admin/confirm-delete-format.jsp
- Moved to dspace/jsp/tools: dspace/jsp/admin/confirm-delete-item.jsp
- Moved to dspace/jsp/tools: dspace/jsp/admin/confirm-withdraw-item.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/edit-collection.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/edit-community.jsp
- Moved to dspace/jsp/tools and changed: dspace/jsp/admin/edit-item-form.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/eperson-browse.jsp
- Moved to dspace-admin: dspace/jsp/admin/eperson-confirm-delete.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/eperson-edit.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/eperson-main.jsp
- Moved to dspace/jsp/tools and changed: dspace/jsp/admin/get-item-id.jsp
- Moved to dspace/jsp/tools and changed: dspace/jsp/admin/group-edit.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/group-eperson-select.jsp
- Moved to dspace/jsp/tools and changed: dspace/jsp/admin/group-list.jsp
- Moved to dspace-admin: dspace/jsp/admin/index.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/item-select.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/list-communities.jsp

- Moved to dspace-admin and changed: dspace/jsp/admin/list-dc-types.jsp
- Removed: dspace/jsp/admin/list-epeople.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/list-formats.jsp
- Moved to dspace/jsp/tools: dspace/jsp/admin/upload-bitstream.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/upload-logo.jsp
- Moved to dspace-admin: dspace/jsp/admin/workflow-abort-confirm.jsp
- Moved to dspace-admin and changed: dspace/jsp/admin/workflow-list.jsp
- Changed: dspace/jsp/browse/authors.jsp
- Changed: dspace/jsp/browse/items-by-author.jsp
- Changed: dspace/jsp/browse/items-by-date.jsp
- Changed: dspace/jsp/browse/no-results.jsp
- New: dspace-admin/eperson-deletion-error.jsp
- New: dspace/jsp/dspace-admin/news-edit.jsp
- New: dspace/jsp/dspace-admin/news-main.jsp
- New: dspace/jsp/dspace-admin/wizard-basicinfo.jsp
- New: dspace/jsp/dspace-admin/wizard-default-item.jsp
- New: dspace/jsp/dspace-admin/wizard-permissions.jsp
- New: dspace/jsp/dspace-admin/wizard-questions.jsp
- Changed: dspace/jsp/components/contact-info.jsp
- Changed: dspace/jsp/error/internal.jsp
- New: dspace/jsp/help/formats.jsp
- Changed: dspace/jsp/layout/footer-default.jsp
- Changed: dspace/jsp/layout/header-default.jsp
- Changed: dspace/jsp/layout/navbar-admin.jsp
- Changed: dspace/jsp/layout/navbar-default.jsp
- Changed: dspace/jsp/login/password.jsp
- Changed: dspace/jsp/mydspace/main.jsp
- Changed: dspace/jsp/mydspace/perform-task.jsp
- Changed: dspace/jsp/mydspace/preview-task.jsp
- Changed: dspace/jsp/mydspace/reject-reason.jsp
- Changed: dspace/jsp/mydspace/remove-item.jsp
- Changed: dspace/jsp/register/edit-profile.jsp
- Changed: dspace/jsp/register/inactive-account.jsp
- Changed: dspace/jsp/register/new-password.jsp
- Changed: dspace/jsp/register/registration-form.jsp
- Changed: dspace/jsp/search/advanced.jsp
- Changed: dspace/jsp/search/results.jsp
- Changed: dspace/jsp/submit/cancel.jsp
- New: dspace/jsp/submit/cc-license.jsp
- Changed: dspace/jsp/submit/choose-file.jsp
- New: dspace/jsp/submit/creative-commons.css
- New: dspace/jsp/submit/creative-commons.jsp
- Changed: dspace/jsp/submit/edit-metadata-1.jsp

- Changed: dspace/jsp/submit/edit-metadata-2.jsp
- Changed: dspace/jsp/submit/get-file-format.jsp
- Changed: dspace/jsp/submit/initial-questions.jsp
- Changed: dspace/jsp/submit/progressbar.jsp
- Changed: dspace/jsp/submit/review.jsp
- Changed: dspace/jsp/submit/select-collection.jsp
- Changed: dspace/jsp/submit/show-license.jsp
- Changed: dspace/jsp/submit/show-uploaded-file.jsp
- Changed: dspace/jsp/submit/upload-error.jsp
- Changed: dspace/jsp/submit/upload-file-list.jsp
- Changed: dspace/jsp/submit/verify-prune.jsp
- New: dspace/jsp/tools/edit-item-form.jsp
- New: dspace/jsp/tools/eperson-list.jsp
- New: dspace/jsp/tools/itemmap-browse.jsp
- New: dspace/jsp/tools/itemmap-info.jsp
- New: dspace/jsp/tools/itemmap-main.jsp

16.21 Changes in DSpace 1.1.1

16.21.1 Bug fixes

- non-administrators can now submit again
- installations now preserve file creation dates, eliminating confusion with upgrades
- authorization editing pages no longer create null entries in database, and no longer handles them poorly (no longer gives blank page instead of displaying policies.)
- registration page Invalid token error page now displayed when an invalid token is received (as opposed to internal server error.) Fixes SF bug #739999
- eperson admin 'recent submission' links fixed for DSpaces deployed somewhere other than at / (e.g. /dspace).
- help pages Link to help pages now includes servlet context (e.g. '/dspace'). Fixes SF bug #738399.

16.21.2 Improvements

- *bin/dspace-info.pl* now checks jsp and asset store files for zero-length files
- *make-release-package* now works with SourceForge CVS
- eperson editor now doesn't display the spurious text 'null'
- item exporter now uses Jakarta's cli command line arg parser (much cleaner)
- item importer improvements:
    - now uses Jakarta's cli command line arg parser (much cleaner)
    - imported items can now be routed through a workflow

- more validation and error messages before import
- can now use email addresses and handles instead of just database IDs
- can import an item to a collection with the workflow suppressed

16.22 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.)