

DSpace 10.x Documentation

Contents

1. Introduction . . . 5
1.1 Release Notes . . . 6
1.2 Functional Overview . . . 13
1.3 Technology Overview . . . 25
2. Installing DSpace . . . 26
3. Upgrading DSpace . . . 46
3.1 Migrating DSpace to a new server . . . 53
3.2 Upgrading from DSpace 6.x or lower . . . 55
4. Using DSpace . . . 58
4.1 Audit Trail feature . . . 59
4.2 Authentication and Authorization . . . 65
4.2.1 Allow Authors to download restricted bitstreams . . . 66
4.2.2 Authentication Plugins . . . 68
4.2.3 Bulk Access Management . . . 89
4.2.4 Embargo . . . 92
4.2.4.1 Pre-3.0 Embargo Lifter Commands . . . 97
4.2.5 Managing User Accounts . . . 98
4.2.5.1 Email Subscriptions . . . 101
4.2.6 Request a Copy . . . 103
4.3 Authority framework functionalities . . . 116
4.3.1 Creation of linked entities . . . 121
4.3.2 Item details: layout . . . 127
4.4 CAPTCHA Verification . . . 130
4.5 Configurable Entities . . . 133
4.5.1 Entities Use Case: Compound Journal . . . 147
4.5.2 Entities Use Case: Research Management . . . 150
4.6 Content Management System . . . 153
4.6.1 Editing CMS metadata . . . 154
4.6.2 Editing User Agreement . . . 157
4.7 Curation System . . . 159
4.7.1 Bundled Tasks . . . 166
4.7.1.1 Bitstream Format Profiler Task . . . 167
4.7.1.2 Link Checker Tasks . . . 168
4.7.1.3 MetadataWebService Task . . . 169
4.7.1.4 MicrosoftTranslator Task . . . 171
4.7.1.5 NoOp Task . . . 172
4.7.1.6 Required Metadata Task . . . 173
4.7.1.7 Virus Scan Task . . . 174
4.8 Custom URL for Entities . . . 176
4.9 Edit Item in Submission mode . . . 180
4.10 Exporting Content and Metadata . . . 184
4.10.1 Signposting . . . 185
4.10.2 OpenAIRE4 Guidelines Compliancy . . . 186
4.10.3 OAI . . . 188
4.10.3.1 OAI 2.0 Server . . . 194
4.10.3.2 OAI-PMH Data Provider 2.0 (Internals) . . . 201
4.10.4 Exchanging Content Between Repositories . . . 203
4.10.5 SWORDv1 Client . . . 204
4.10.6 Linked (Open) Data . . . 205
4.10.7 Rioxx v3 schema compliance . . . 215
4.11 Ingesting Content and Metadata . . . 216
4.11.1 Submission User Interface . . . 217
4.11.1.1 Basic Duplicate Detection for Submission/Workflow . . . 234
4.11.1.2 Live Import from external sources . . . 238
4.11.1.3 Metadata security configuration . . . 247
4.11.1.4 Set a bitstream as primary . . . 249
4.11.1.5 Simple HTML Fragment Markup . . . 251
4.11.1.6 Supervision Orders . . . 252
4.11.2 Configurable Workflow . . . 255
4.11.3 Importing and Exporting Content via Packages . . . 263
4.11.4 Importing and Exporting Items via Simple Archive Format . . . 267
4.11.5 Registering Bitstreams via Simple Archive Format . . . 275
4.11.6 Importing Items via basic bibliographic formats (Endnote, BibTex, RIS, CSV, etc) and online services (arXiv, PubMed, CrossRef, CiNii, etc) . . . 277
4.11.7 Exporting and Importing Community and Collection Hierarchy . . . 279
4.11.8 SWORDv1 Server . . . 281
4.11.9 SWORDv2 Server . . . 286
4.11.10 Ingesting HTML Archives . . . 294
4.12 Items and Metadata . . . 295
4.12.1 Authority Control of Metadata Values . . . 296
4.12.1.1 ORCID Authority . . . 300
4.12.2 Batch Metadata Editing . . . 311
4.12.2.1 Batch Metadata Editing Configuration . . . 317
4.12.3 DOI Digital Object Identifier . . . 319
4.12.4 Item Level Versioning . . . 329
4.12.4.1 Bitstream versioning and replacement . . . 336
4.12.5 Mapping/Linking Items to multiple Collections . . . 338
4.12.6 Metadata Recommendations . . . 339

4.12.7 Moving Items . . . 340
4.12.8 PDF Citation Cover Page . . . 341
4.12.9 Request Withdrawn and Reinstate of an item . . . 344
4.12.10 Updating Items via Simple Archive Format . . . 352
4.13 Managing Community Hierarchy . . . 354
4.14 Nested metadata in DSpace items . . . 356
4.15 ORCID Integration . . . 360
4.15.1 ORCID Login: Guide to Repository access and Account linking . . . 375
4.16 Researcher Profiles . . . 380
4.17 Sending emails to fixed recipients . . . 383
4.18 Share Content via Social Network . . . 384
4.19 Shared submission / workspace . . . 386
4.20 Solr Data Management . . . 388
4.21 Statistics and Metrics . . . 390
4.21.1 Exchange usage statistics with IRUS . . . 391
4.21.2 DSpace Google Analytics Statistics . . . 393
4.21.3 SOLR Statistics . . . 396
4.21.3.1 SOLR Statistics Maintenance . . . 405
4.21.3.1.1 Testing Solr Shards . . . 410
4.21.4 Matomo Integration . . . 413
4.22 User Interface . . . 419
4.22.1 User Interface Configuration . . . 420
4.22.2 User Interface Customization . . . 436
4.22.3 Accessibility . . . 450
4.22.4 Browse . . . 452
4.22.5 Contextual Help Tooltips . . . 454
4.22.6 Discovery . . . 456
4.22.7 IIIF Configuration . . . 472
4.22.8 Multilingual Support . . . 476
4.22.9 User Interface Debugging . . . 478
5. Learning DSpace . . . 479
5.1 Community and Collection management . . . 481
5.1.1 'Assigning Roles' in Collections: How to Set Up Workflows and Grant Privileges . . . 482
5.1.2 Collection Management . . . 485
5.1.2.1 Create Collection . . . 486
5.1.2.2 Delete Collection . . . 490
5.1.2.3 Edit Collection . . . 494
5.1.3 Community Management . . . 520
5.1.3.1 Create a Community . . . 521
5.1.3.2 Delete Community . . . 527
5.1.3.3 Edit Community . . . 530
5.1.4 Export Metadata by Collection or Community . . . 546
5.2 Content (Item) management . . . 551
5.2.1 Add item . . . 552
5.2.2 Delete item . . . 563
5.2.3 Edit Item . . . 566
5.2.3.1 Authorizations (Manage access to an item) . . . 572
5.2.3.2 Collection Mapper . . . 579
5.2.3.3 Edit Bitstream . . . 585
5.2.3.4 Edit Metadata . . . 593
5.2.3.5 Edit Relationship . . . 609
5.2.3.6 Make an item discoverable . . . 616
5.2.3.7 Make an item non-discoverable . . . 620
5.2.3.8 Move an Item . . . 624
5.2.3.9 Versioned Item . . . 627
5.2.3.10 Withdraw an item . . . 634
5.2.4 Embargo an item . . . 639
5.2.5 Lease an item . . . 648
5.3 DSpace Demo Quick Start . . . 658
5.4 Management sidebar . . . 659
5.4.1 Administrator Reports (Beta feature) . . . 662
5.4.2 COAR Notify . . . 668
5.4.2.1 COAR Notify - Dashboard . . . 670
5.4.2.2 COAR Notify - LDN Services . . . 679
5.4.2.3 Use Cases - Getting two DSpace application to communicate with each other . . . 688
5.4.3 Notifications . . . 693
5.4.3.1 Publication Claim . . . 694
5.4.3.2 Quality Assurance . . . 708
5.4.3.2.1 COAR Notify Integration . . . 714
5.4.3.2.2 OpenAIRE Integration . . . 715
5.5 Menus . . . 725
5.6 Registry management . . . 727
5.6.1 Metadata Registry Management . . . 728
5.7 Request-a-copy . . . 741
5.7.1 Approve an access request . . . 742
5.7.2 Create a request . . . 744
5.8 Search - Advanced . . . 747
5.9 User management . . . 748
5.9.1 Add or Manage an E-Person . . . 749

5.9.2 Create or manage a user group . . . 769
6. System Administration . . . 784
6.1 AIP Backup and Restore . . . 785
6.1.1 DSpace AIP Format . . . 800
6.2 Ant targets and options . . . 811
6.3 Command Line Operations . . . 813
6.3.1 Executing streams of commands . . . 815
6.3.2 Database Utilities . . . 816
6.3.3 Miscellaneous Commands . . . 818
6.4 Handle.Net Registry Support . . . 819
6.5 Logical Item Filtering and DOI Filtered Provider for DSpace . . . 822
6.6 Mediafilters for Transforming DSpace Content . . . 827
6.6.1 ImageMagick Media Filters . . . 831
6.7 Performance Tuning DSpace . . . 836
6.8 Ping or Healthcheck endpoints for confirming DSpace services are functional . . . 841
6.9 Proxy Configuration . . . 842
6.10 Scheduled Tasks via Cron . . . 843
6.11 Search Engine Optimization . . . 846
6.11.1 Google Scholar Metadata Mappings . . . 852
6.12 Troubleshooting Information . . . 853
6.13 Validating CheckSums of Bitstreams . . . 854
7. DSpace Development . . . 858
7.1 Advanced Customisation . . . 859
7.1.1 DSpace Service Manager . . . 860
7.2 Batch Processing . . . 862
7.3 Curation Tasks . . . 863
7.3.1 Curation tasks in Jython . . . 867
7.4 Development Tools Provided by DSpace . . . 869
7.5 Menu providers (WIP) . . . 870
7.6 REST API . . . 874
7.7 Services to support Alternative Identifiers . . . 878
7.8 User Interface Design Principles & Accessibility . . . 882
7.9 Workflow . . . 887
8. DSpace Reference . . . 892
8.1 Architecture . . . 893
8.1.1 Application Layer . . . 895
8.1.2 Business Logic Layer . . . 897
8.1.3 DSpace Services Framework . . . 914
8.1.4 Storage Layer . . . 918
8.2 Configuration Reference . . . 927
8.3 Directories and Files . . . 983
8.4 DSpace Item State Definitions . . . 986
8.5 Metadata and Bitstream Format Registries . . . 988
8.6 History . . . 995
8.6.1 Changes in 10.x . . . 996
8.6.2 Changes in Older Releases . . . 997

Introduction

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

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

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

For people actually running a DSpace service, there is an installation guide, and sections on configuration and the directory structure. Support options are available in the DSpace Support Guide.

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 ar chitecture section.

Other good sources of information are:

- The DSpace Support Guide lists various places to ask for help, report bugs or security issues, etc.
- dspace.org has announcements and contains useful information about bringing up an instance of DSpace at your organization.
- 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.
- The DSpace REST API contract which documents the REST API behavior, etc. If you want source code docs, we also provide JavaDocs for the Java API layer which can be built by running `mvn javadoc:javadoc`
- The DSpace Community List. Join DSpace-Community 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-Community for news of software releases, user conferences, and announcements about DSpace.
- The DSpace Technical List. DSpace developers & fellow community members 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-Dev 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.

Release Notes

Upgrade from any past version of DSpace!

Installing DSpace provides an overview of the DSpace installation process and all prerequisite software. You should review this before attempting an upgrade, in order to ensure you are running the required versions of Java, Node, etc.

Upgrading DSpace provides a guide for upgrading from *any old version* of DSpace. As in the past, your data migrates automatically, no matter which older version you are running.

Migrating DSpace to a new server provides an alternative way to upgrade by installing the new version of DSpace and migrating your existing data over.

- 10.0 Release Notes
  - Security Fixes
  - New User Features
  - Breaking Changes
  - Major Updates and Improvements
  - New/Updated Language support
- 10.0 Acknowledgments
  - Frontend / User Interface Acknowledgments
  - Backend / REST API Acknowledgments
  - Reviewer Acknowledgments
  - Testathon Acknowledgements
  - Additional Thanks

10.0 Release Notes

DSpace 10.0 was released on May 28, 2026

To try out DSpace 10.0 immediately, see Try out DSpace 10.

To test an upgrade to DSpace 10.0 from 9.x *or any prior version*, see Upgrading DSpace

- To upgrade to 10.0, you MUST upgrade *both* the backend and frontend (user interface). Many bug fixes require updating *both.*

To install DSpace 10.0, see Installing DSpace.

- Download DSpace 10.0 Backend: https://github.com/DSpace/DSpace/releases/tag/dspace-10.0
- Download DSpace 10.0 User Interface: https://github.com/DSpace/dspace-angular/releases/tag/dspace-10.0

DSpace 10.0 is a major release of the DSpace platform. It provides new features and improvements, along with bug fixes. You should be aware that all major releases may provide some "breaking changes" (major changes that may impact your local customizations).

Security Fixes

- **Fix for GHSA-9x82-rm84-c6x7** *(high severity*). *NOTE: Fix is also included in 9.3, 8.4 and 7.6.7.* Remote Code Execution (RCE) possible in Velocity templates used by LDN (Linked Data Notifications) when COAR Notify is enabled. *(NOTE: A CVE ID has been requested but not yet assigned)*
- **Fix for GHSA-9qm4-rh6w-pq5x** *(moderate severity). NOTE: Fix is also included in 9.3, 8.4 and 7.6.7.* Path traversal vulnerability possible in LDN (Linked Data Notifications) message generation when COAR Notify is enabled. *(NOTE: A CVE ID has been requested but not yet assigned)*
- **Fix for GHSA-v66x-68f2-pxf5** *(moderate severity). NOTE: Fix is also included in 9.3, 8.4 and 7.6.7.* Path Traversal Vulnerability is possible in Curation Task Reporter output path. *(NOTE: A CVE ID has been requested but not yet assigned)*
- **Fix for GHSA-c827-pw3m-67w7** *(moderate severity)*. ORE resource URI does not validate scheme for non-web resources when harvesting OAI content. *(NOTE: A CVE ID has been requested but not yet assigned)*
- **Patch for CVE-2026-27739 in Angular SSR** *(critical severity).* All versions of Angular SSR (Server Side Rendering) contain a critical SSRF (Server-Side Request Forgery) vulnerability. This SSRF vulnerability may be possible to exploit in DSpace sites that are **not** running DSpace behind a well-configured proxy (see mailing list announcement). This release of DSpace updates to a patched version of Angular along with additional protection via the new "ui > baseUrl" setting (see "Breaking Changes" below)
- **Patch for CVE-2025-66516 / CVE-2025-54988 in Apache Tika** *(critical severity) NOTE: Fix was already included in 9.2, 8.3 and 7.6.6*. All versions of Apache Tika prior to version 3.2.2 contain a critical XML External Entity (XXE) vulnerability. This XXE vulnerability may be possible to exploit in DSpace if an attacker has submitter privileges. See #11678 for more details.
  - After upgrading, we recommend all sites recreate text extracted files. This is a safety measure to ensure that none of those text extracted files contain unexpected information because of a prior exploit of this XXE vulnerability. To recreate your text extracted files, run:

    ```
    # This command will force all current text-extracted files to be deleted and recreated.
    ./dspace filter-media -f -p "Text Extractor"
    ```

- **Fix for potential XPath Injection in Controlled Vocabulary lookup.** *NOTE: Fix was already included in 9.2, 8.3 and 7.6.6*. This vulnerability only allows an attacker access to the controlled vocabulary XML file(s), which are already public in source code. So, this vulnerability poses no security risk but it has been patched for future code safety. See #11042 and #11075 *(Reported by odaysec. Fixed by odaysec and Mark Wood)*
- **Fix for CVE-2025-53621** (*moderate severity) NOTE: Fix was already included in 9.1, 8.2 and 7.6.4*. XML External Entity (XXE) injection possible in import via Simple Archive Format (SAF) or import from external sources. See security advisory or mailing list “security notice” for details.

- **Fix for CVE-2025-53622** *(moderate severity). NOTE: Fix was already included in 9.1, 8.2 and 7.6.4*. Path traversal vulnerability in Simple Archive Format (SAF) package import via “contents” file. See security advisory or mailing list “security notice” for details.
- Updated LDAPAuthentication to Spring LDAP in order to remove insecure JNDI usage. See #11656 *(Reported by Sascha Szott. Fixed by Johnny Mendes do Carmo)*
- This release also includes many dependency updates in order to keep all DSpace sites secure. Some of these updates patch vulnerabilities that have been reported by those dependencies. But no exploits of these vulnerabilities have been confirmed in DSpace.

New User Features

- Features ported to DSpace from DSpace-CRIS via the ongoing merger.
  - **Edit item in Submission mode.** Items now have a new "Edit" menu which allows you to edit the details of the item using the submission form. See #11964 and #5145 *(Donated by 4Science)*
  - **Metadata-level security controls.** Individual metadata fields can be configured to support security settings (e.g. public/private) via the new "Edit" form. For example you can enable this for Researcher Profiles to allow researchers to control whether their birthday or email address are public/private. Included in #11964 and #5145 *(Donated by 4Science)*
  - **Shared Workspaces.** It is possible to enable "Shared workspaces" for individual collections. When enabled, all collection submitters can share their in-progress submissions, allowing for collaborative editing/finalization of submissions to that collection. Included in #11964 and #5145 *(Donated by 4Science)*
  - **Submission form linked to collection via UI.** It is possible to select which submission form to use for a collection using a dropdown on the collection edit page; previously only accessible through config. Included in #11964 and #5145 *(Donated by 4Science)*
  - **Custom URLs for Items.** Items can have human-friendly URLs instead of UUID-based paths. When enabled, custom URLs can be manually set by a user or auto-generated from metadata. See #11549 and #4814 for more details. *(Donated by 4Science)*
  - **Share Content via Social Network**. An optional sharing bar can be enabled on the homepage, search results and item pages. This bar allows users to quickly share DSpace content on social networks, via email, etc. See #5112 *(Ported by PCG Academia, written by 4Science)*
  - **Basic Content Management System features for Homepage & User Agreement.** You can now edit the Homepage header, footer and news section via an Administrative UI form that supports HTML and Markdown. This includes editing translations of that content to other supported languages via the same form. Similarly the End User Agreement can also be edited via that form. See #5106 *(Ported by PCG Academia, written by 4Science)*
  - **Audit Trails for item changes.** The auditing system logs events that modify the state of DSpace objects, and administrators are able to inspect and filter logs. This feature can be toggled in config via `audit.enabled`. See #11072 and #4576 for more details. *(Donated by 4Science)*
  - **Download panel on Item pages.** Optionally, you can enable display of all files at the top of the Item page via a Download box/panel by enabling the new "showDownloadLinkAsAttachment" UI configuration. See #5132 (D*onated by 4Science)*
  - **Author access to their own restricted files.** Optionally, you can allow authors access to download restricted (or embargoed) files that they submitted. See #5132 (D*onated by 4Science)*
  - **Authority framework enhancements.** Several authority framework enhancements have been migrated to DSpace including management of relationships and virtual metadata via authority (both similar to DSpace-CRIS). These features allow for optional early migration of DSpace-CRIS entities into DSpace 10.0. See #11718 and #4956 *(Donated by 4Science)*
  - **Nested metadata for items.** It is now possible to group related metadata fields together via "group" or "inline-group" input types in Submission forms. This can be used to couple author names with their affiliations (or similar), similar to DSpace-CRIS. See #11945 and #5097 *(Donated by 4Science)*
  - **Sending emails to fixed recipients.** When the relevant email configurations are enabled, all emails are sent to "catchAll" recipients instead of original recipients. Especially useful for testing or development environments. See #11447 *(Donated by 4Science)*
- **Unified treatment of DOIs.** DOIs generated within DSpace are now stored consistently in `dc.identifier.doi`, and publisher or externally generated DOIs are stored in `dc.relation.hasversion`. There is also a new `doi-migrate` command to help migrate DOI metadata between fields and normalize their format, and DOI is shown on publication pages. See #11545 and #5133 *(Donated by The Library Code with the support of Technische Universität Dortmund and Deutsche Forschungsgemeinschaft DFG)*
  - **Registration of DOIs with Crossref.** It is possible to register new DOIs in DSpace with Crossref. See #11962 *(Donated by The Library Code with the support of Crossref.)*
- **Editor relationship.** There is a new default entity relationship for editors of a Publication. These can be Persons or OrgUnits. See #11881 *(Donated by The Library Code with the support of Technische Universität Dortmund and Deutsche Forschungsgemeinschaft DFG)*
- **Dataset entity type.** There is a new entity model for Dataset, in order to support more institutions using DSpace as a repository for Research Data. See #11969 and #5137 *(Donated by Eike Löhden)*
- **Asynchronous batch deletion.** In order to improve performance for large-scale deletions of Communities, Collections, and Items, a deletion process now runs in the background. See #11068 and #4570 *(Donated by 4Science)*
- **New request item strategy.** It is now possible to send emails based on bitstream's contact metadata. See #10436 *(Donated by Paulo Graça)*

Breaking Changes

The following major changes may negatively impact or "break" your local customizations to prior versions of DSpace. Please be aware of them before upgrading.

- **Upgrade to Angular 20.** This requires *removing* all `standalone: true` declarations from custom components (`@Component`) . If you forget to make this change, then lint checks fail during your Angular build and let you know what needs to be updated. #4804
- **Node v20.19+ is now required.** Because of the upgrade to Angular 20, you **must** use Node v20.19+, v22.x or v24.x to build and run the DSpace frontend (UI).
- **JDK 21 is now required on the backend.** The DSpace REST API / backend now requires Java/JDK version 21 (at a minimum) to build and run. This has been upgraded from JDK 17 in past releases. #11739
- **Frontend's new "baseUrl" setting is required for Server Side Rendering (SSR).** The frontend configuration has a newly added "ui > baseUrl" setting which should match the value of "dspace.ui.url" on the backend. This setting is REQUIRED for Server-Side-Rendering (SSR), robots.txt and redirects to function properly. Along with an update to the latest version of Angular 20, this new setting also helps to patch against the Angular SSR vulnerability CVE-2026-27739.
- **Theme configuration has been updated and simplified.** You will need to update your existing themes to use the new 10.0 theme configuration. Notes on the upgrade process are available in the "Upgrading from 9.x to 10.x" section of the User Interface Customization page.

- **The "Edit this item" menu on an Item page has been renamed "Administer".** While visiting an Item page (while logged in as an Administrator), you'll see the options available have changed. The "Edit" menu now brings you to the new Edit item in Submission mode feature, while the "Administer" menu will bring you to all previously existing Item editing tools.
- **Rename "`defaultLanguage`" configuration to "`fallbackLanguage`".** This just aligns with the naming convention for the `@ngx-translate/core` module that we use. See User Interface Configuration
- **Replaced "webui.content_disposition_format" with "webui.content_disposition_inline"** - In prior versions the "webui.content_disposition_format" setting (in dspace.cfg) listed MIME Types which are **not allowed** be displayed inline (in a user's browser). This setting has been replaced by a "webui.content_disposition_inline" setting which lists MIME types which **are allowed** to be displayed inline. By default PDF, GIF, JPEG, PNG, MPG and MP4 are allowed to be displayed inline. This improves security of unknown or custom formats by ensuring they don't default to being displayed inline. See https://github.com/DSpace/DSpace/pull/11956
- **Migration from SHERPA/RoMEO API to Jisc Open Policy Finder API.** If you are using the SHERPA/RoMEO integration for submission forms, DSpace has been updated to use the new Jisc Open Policy Finder API because the Sherpa API is being retired. This means that the older "sherpa.romeo.*" settings (in dspace.cfg) have been replaced by "openpolicyfinder.*" settings (in external-providers.cfg). The sample Authority Control Settings (in authority.cfg) for Publisher and Journal Title have also been updated. #12005
- **Must select whether to use traditional DSpace Entity Relationships (with virtual metadata) or new DSpace-CRIS Entity Relationships (using authority framework).** See https://github.com/DSpace/DSpace/pull/11718
    - Set `item.enable-virtual-metadata = true` *(default value)* to use traditional DSpace Entity Relationships (with virtual metadata). Sites which are upgrading from an older version of DSpace should use this option, as it ensures any existing entities are displayed properly.
    - Set `item.enable-virtual-metadata = false` to use new DSpace-CRIS Entity Relationships (via authority framework). Sites which may be migrating from DSpace-CRIS to DSpace 10 should likely use this option, as it ensures any existing DSpace-CRIS entities are displayed properly. NOTE: This will result in traditional DSpace Entity Relationships no longer being displayed in the user interface.
- **License and Text extraction bundles/files are now admin-only by default.** Previously, license files and extracted text would inherit the permissions of the Item itself. Specific bundles (LICENSE, TEXT, SWORD) of files are now access-restricted by default. You can control which bundles are access restricted via the new "core.authorization.restricted-bundle" configuration in dspace.cfg. #12306
    - NOTE: After your upgrade, if you wish to automatically access restrict all TEXT bundles in your DSpace, you can do so by running the "filter-media" script in *force mode* (-f). This will result in all TEXT bundles and files being recreated with the new access restricted settings. For example: `[dspace]/bin/dspace filter-media -f`
- **Solr "search" core needs to be cleared and reindexed.** In order to resolve an error in Solr 9 (see #11475), we've had to update the Solr schema.xml for the "search" core (see #10526). This minor update to the schema is unfortunately **incompatible** with prior "search" core indexes. In testing, we've found this requires destroying all data in the current index, and reindexing via a process like this:
    - First, destroy all data in the current "search" core by running this command-line tool (using the "-d" flag): `./dspace index-discovery -d`
    - Next, restart Solr
    - Finally, perform a full reindex/rebuild (using the "-b" flag): `./dspace index-discovery -b`
    - WARNING: If you skip these steps, you may see errors like this on your next reindex or Item submission:
        - ```
          Exception writing document id Item-[uuid] to the index; possible analysis error:
          cannot change field "admin" from index options=DOCS_AND_FREQS_AND_POSITIONS to inconsistent
          index options=DOCS
          ```

Major Updates and Improvements

- General user enhancements and fixes
    - Fixed bug where item pages would throw an Apache gateway error when too many bitstreams were added to the header. #8943 *(Donated by Nicholas Woodward)*
    - Structure Builder now allows import beneath a parent community. #2559 *(Donated by The Library Code, with support from Deutsches Institut für Urbanistik (DIFU))*
    - Fixed an authorization issue where security details in SOLR could go out of sync without a manual reindexing. #2853 *(Donated by Toni Prieto)*
    - Item pages can now display generic rights information based on license names and links given in metadata. #3338 *(Donated by Abel Gómez)*
    - HTML tags are now escaped in content injected into the DOM using the innerHTML attribute. #1495 *(Donated by Atmire)*
    - Fixed bug where bitstreams with embargo lift date in metadata were not appearing on Item Page. #8640 *(Donated by Vir Softech)*
    - When an Entity's Item Page displays tabbed search, switching tabs no longer retains the query parameters from the prior tab. #2729 *(Donated by Atmire)*
    - Fixed issues with downloading bitstreams whose filenames contained non-ASCII characters using the Safari browser. #11191 *(Donated by Neki-IT)*
    - The language selected in a user's profile is now the default language displayed by DSpace unless a different language is selected via the header. #4241 *(Donated by Atmire)*
    - Fixed bug where embargo status badge would disappear from Item page if the page was reloaded. #4811 *(Donated by Atmire)*
    - Fixed bug where the IIIF viewer was not working due to a missing `mime.types` file. #11804 (*Donated by 4Science)*
    - Fixed an issue where deleted bitstreams returned HTTP 401 unauthorized instead of 404. #11629 *(Donated by Jesiel Viana)*
    - Fixed bug where metadata export was no longer respecting `metadata.hide.*` properties #11197 *(Donated by Atmire)*
    - Fixed bug with the "Show more" functionality of truncatable component for content containing HTML #4948 *(Donated by 4Science)*
    - Fixed an issue where memory errors would occur when downloading large bitstreams from an S3 remote. #12468 *(Donated by 4Science)*
    - Fixed bug where hierarchical & advanced search on community and collection pages redirected user to the global search route. #5210 *(Donated by Atmire)*
    - Fixed various small issues with the display of authors and other metadata on item pages. #5665 *(Donated by Atmire)*
    - Improved the FileSizePipe to take localization into account when formatting. #5054 *(Donated by Sascha Szott)*
    - Improved email subscription functionality to include more context, better formatting, and easier template configuration. #9994 (*Donated by Atmire, with the support of the International Livestock Research Institute*)
    - There is now an "eye" icon in the password field of login to enable toggling password visibility. #4988 *(Donated by Arya Khochare)*
- Submission / Workflow enhancements and fixes
          ```

    - Fixed an issue where loading many mapped collection forms via `item-submission` could cause a Fetch error. #10750 *(Donated by Paulo Graça)*
    - There is now a configurable limit on the number of items that can be added or edited in a single CSV metadata import. #9663 *(Donated by Neki-IT)*
    - Fixed bug where a Submitter could not deposit items via SWORD when the item had an embargo defined. #10404 *(Donated by dataquest)*
    - Fixed bug where dynamic tag input field was wrongly splitting values on unexpected characters (like "<"). Now it only splits on pressing Enter or adding a comma. #4664 *(Donated by The Library Code)*
    - Fixed bug where a newly created Researcher Profile could not be selected as an author of a new Publication Entity. #4648 *(Donated by Jesiel Viana)*
    - Fixed bug where submission form status icons didn't update properly when type-bind was used with a required field. #2572 and #2742 *(Donated by The Library Code)*
    - Fixed bug where file uploads would fail if the "assetstore.dir" was set to a location using a symbolic link. #11169 *(Donated by 4Science)*
    - Fixed memory leak in submission form which could occur if you closed and reopened the form several times in a row. #4645 *(Donated by Jukka Lipka)*
    - Fixed bug where output of "submission-forms-migrate" script was an invalid submission configuration. #8443 *(Donated by Neki-IT)*
    - Fixed bug where importing a multi-paragraph abstract from PubMed would result in it being split into many abstracts in DSpace. #10559 *(Donated by Atmire)*
    - Fixed bug where removing and reordering the entries of a repeatable field in submission form could lead to incorrect values. #4418 *(Donated by Jukka Lipka)*
    - Fixed an issue where it was not possible to download bitstreams during submission while impersonating a user. #8957 *(Donated by Atmire)*
    - Fixed an issue where controlled vocabulary lookup was not working for text with accented characters. #12097 *(Donated by Istvan Vig)*
    - Optionally, you can allow non-admin users to import external entities into a specified collection to which they have submitter privileges. #12279 *(Donated by Eike Löhden)*
    - ArXiv importer began failing because its configuration didn't use the HTTPS URL. #10996 *(Donated by Atmire)*
    - Alphabetized all external sources in the import dropdown by source name. #10806
- Integration fixes
    - ORCID iDs are now included in the metadata that gets sent to DataCite from DSpace. #9883 *(Donated by Eike Löhden)*
    - Fixed bug where ORCID push failed for works that share an ISSN external-id with existing work. #10783 *(Donated by The Library Code)*
    - ORCID iD icons and links are now displayed according to ORCID's Display guidelines. #4656 *(Donated by 4Science with additions by Nicholas Woodward)*
    - Fixed bug where import of a Person Entity from ORCID was not working. #11674 *(Donated by 4Science)*
    - Fixed bug where ORCID Integration would reject a work if it referenced an ISSN that already exists (e.g. an article referencing a journal via SSN). #10738 *(Donated by The Library Code)*
    - Fixed bug where ORCID queue list would include items that are still in workspace or workflow. #4654 *(Donated by Pierre Lasou)*
    - Improved mappings between ORCID document type and DSpace's dc.type field #11253 *(Donated by Pierre Lasou)*
    - Fixed bug occurring during OAI update process when batch size exceeded and items had metadata-level embargoes. #12112 *(Donated by Toni Prieto)*
    - Fixed bug where OAI-PMH could crash if it encountered forbidden XML characters in metadata. #10721 *(Donated by Neki-IT)*
    - Fixed OAI-PMH validation errors because of an invalid `responseDate`. #11129 *(Donated by LucasVaroneW)*
    - OAI-PMH now serves pre-transformed HTML if the client sends an Accept header requesting text/HTML. #11648 *(Donated by The Library Code)*
    - Fixed bug where OAI identifier prefix could revert back to un-interpolated value when not defined in config. #12505 *(Donated by Atmire)*
    - Fixed several failures occurring with the OpenAIRE Search API. #11967 *(Donated by dataquest)*
    - OpenAIRE crosswalk now supports types Animation, Book chapter, Learning Object, and Presentation. #12171 *(Donated by Atmire)*
    - Enabled sending a `Client-Id` header in requests to the ROR (Research Organization Registry) API. #11653 *(Donated by 4Science)*
    - Updated to use version 2 of the ROR (Research Organization Registry) API. #11019 *(Donated by Jesiel Viana)*
    - Import from PubMed now supports adding your PubMed API Key via a new "pubmed.apiKey" configuration in "external-providers.cfg". #10356 *(Donated by Atmire)*
    - Fixed performance of SWORDv2 servicedocument when you have a large number of collections. #10740 (Donated by fribeiro-fccn)
    - Fixed bug where SWORDv2 read/update/delete actions were failing with a 500 error. #11547 *(Donated by Marsa Haoua)*
    - Fixed bug where SWORDv1 submission failed if the user was not an Administrator. #8647
    - Fixed bug where bitstream downloads would fail when using a script blocker on sites with Matomo integration. #4991 *(Donated by PCG Academia)*
    - Fixed an issue with the IIIF viewer handling of the `dspace.iiif.enabled` field. #12303 *(Donated by Marsa Haoua)*
- Administrative enhancements and fixes
    - Fixed bug where admin privileges not immediately inherited from community to collection on creation #9652 *(Donated by Toni Prieto)*
    - Fixed an issue where the search function from "Edit" in the sidebar listed items which the user is not allowed to edit. #1331 *(Donated by Toni Prieto)*
    - Fixed an issue where after creating a new EPerson, the application would become unresponsive. #5000 *(Donated by PCG Academia)*
    - Fixed an issue where the menu reducer could throw an error when you tried to delete an already deleted menu section. #4539 *(Donated by Atmire)*
    - Fixed an issue where the checksum checker could fail to complete due to memory constraints in repositories with many bitstreams. #7322 *(Donated by Miika Nurminen)*
    - Fixed an issue where the curation task CreateMissingIdentifiers would not work for an item without a handle. #11676 *(Donated by The Library Code)*
    - Enhanced the configuration possibilities for basic duplicate detection #11349 *(Donated by Andreas Mahnke)*
    - Fixed several bugs where Community Administrators would see objects they didn't have permissions on when using the "edit" or "new" sidebar menus. #8916, #8917, #8918 *(Donated by Vir Softech)*
    - Fixed bug where Item Versioning could fail if special characters were added to the "Summary" field. #4327 *(Donated by Jukka Lipka)*
    - Fixed bug where creating an EPerson would fail for an email containing uppercase letters. #4338 *(Donated by PCG Academia)*
    - When creating a new Bundle all custom, non-admin policies are now inherited from the parent Item. #9212 *(Donated by The Library Code)*
    - Fixed bug where client-side rendering (CSR) of the user interface could fail if the "ui.nameSpace" configuration was customized in "config.*.yml". #4637 *(Donated by Milan Majchrak)*
    - Fixed bug where backend would fail to startup when "citation-page.enabled_communities" configuration was enabled. #11167 *(Donated by Miika Nurminen)*
    - Amazon S3 integration has been updated to use AWS SDK v2. #10819 *(Donated by Mark Patton)*

    - Fixed an issue where S3 connection remained open after HEAD requests on a bitstream download URL or sending files for Request a copy. #12244 *(Donated by Atmire)*
    - Removed automated checks for the "pgcrypto" plugin in PostgreSQL. As of PostgreSQL v13, this plugin is supported out-of-the-box and no longer needs to be installed separately. #10853 *(Donated by Francisco Carvalho)*
    - Fixed issue where some Solr 9 queries would throw a "Query contains too many nested clauses" error. #11475 *(Donated by Sascha Szott)*
    - Fixed bug where community/collection subscription emails were not working as an error occurred when running "./subscription-send" #11001 *(Donated by Alan Orth)*
    - Deleting a Group now requires confirmation (via a confirmation popup). #4346 *(Donated by Jukka Lipka)*
    - A strong warning dialog now displays to administrators before permanently deleting an item. #5044 *(Donated by eScire)*
    - Fixed bug where it was not possible to set the DELETE resource policy. #10741 *(Donated by DSquare Technologies)*
- Command-line script fixes
    - Fixed bug where "checker" and "bitstore-migrate" command-line scripts would not save updates / current status if the script was interrupted. #10507 and #10489 *(Donated by David Steelman)*
    - Fixed bug where "subscription-send" command-line script was not sending updates about restricted items when the user has permissions on those restricted items. #10506 *(Donated by dataquest)*
    - Fixed bug where "filter-media" command-line script did not understand orientation metadata in jpeg images. #10532 *(Donated by 4Science)*
    - Fixed bug where running "export" command-line script on an Item in workspace or workflow would return an error even though the export succeeds. #11074 *(Donated by Martin Walk)*
    - Fixed bug where "solr-export-statistics" command-line script was throwing date-based errors. #11192 *(Donated by Daniel Coelho)*
    - Fixed open connection leak in "import" command-line script. #10886 *(Donated by Atmire)*
    - Fixed bug where "cleanup" command-line script would fail if a request-for-copy existed in the database for the deleted bitstream. #9674 *(Donated by Atmire)*
    - Fixed bug where "cleanup" command-line script would fail if deleted bitstreams still had orphaned rows in `bundle2bitstream`. #11009 *(Donated by Atmire and The Library Code)*
- Authentication enhancements and fixes
    - Added a specialgroup configuration for OIDC authentication, similar to those provided by LDAP and Password plugins. #11040 *(Donated by Daniel Coelho)*
    - Fixed a bug where authentication methods would not appear if page refreshed before authentication token expired. #4662 *(Donated by Atmire)*
    - Fixed bug where LDAP authentication would fail with when a user's LDAP entry had no email field. #11292 *(Donated by dataquest)*
    - Fixed database connection leak that could occur via many unsuccessful logins using password login or ORCID login. #11202 *(Donated by Atmire)*
    - Fixed issues with correctly populating special groups for several authentication methods. #11633 *(Donated by 4Science)*
    - Fixed bug in Shibboleth authentication where EPerson metadata was not being updated after login. #11568 *(Donated by Neki-IT)*
    - Fixed bug where the registration page and forgot password page were accessible to authenticated users. #4592 *(Donated by 4Science)*
    - Support for X.509 certificate authentication has been removed as it appears to be unused and non-functional. #11334
- Accessibility improvements
    - Fixed serious accessibility issues on submission form by adding aria attributes and live region messages. #1187 *(Built by 4Science, funded via the DSpace Development Fund)*
    - Fixed an accessibility issue on submission form fields by removing placeholders and replacing them with labels where needed. #4198 *(Built by 4Science, funded via the DSpace Development Fund)*
    - Added new bitstream metadata in order to include an audio transcript and/or a video description to an audio or video bitstream. #5393 *(Built by 4Science, funded via the DSpace Development Fund)*
    - Integrated @ng-dynamic-forms library into code base, resolving an accessibility issue related to ngbTypehead #1272 (*Built by 4Science, funded via the DSpace Development Fund)*
    - Fixed an accessibility bug where visually disabled buttons failed to block interaction via `space` keypress. #5645 *(Donated by Neki-IT)*
    - Fixed on scrollable dropdown - selection highlighting, dynamic loading, and keyboard navigation issue #4958 *(Donated by Solomon Kahsai)*
    - Fixed an ARIA error related to the edit menu on community and item pages. #5488 *(Donated by eScire)*
- Usability improvements and fixes
    - The New and Edit sidebar search modals now show full hierarchical path instead of only immediate parent, and tooltips are improved. #1602 *(Donated by dataquest)*
    - Fixed an issue where the Subscribe button on a community or collection did not update after user subscribed. #4195 *(Donated by eScire)*
    - Fixed an issue where keyword links on item pages were leading unexpectedly to the subject category browse page. #5314 *(Donated by eScire)*
    - Fixed an issue where edited group details were not immediately reflected in the UI after saving. #5454 *(Donated by eScire)*
    - The "Back to results" button logic on item pages has been improved to prevent stale navigation when users visit items from different contexts. #5170 *(Donated by dataquest)*
    - Fixed bugs where the end user agreement would not save until the page was refreshed and stall indefinitely on submit. #5085 *(Donated by PCG Academia)*
    - Fixed an issue where the status icon of the CC license showed a warning if the section was empty but not required. #4845 *(Donated by 4Science)*
    - Fixed an issue where the "New process" dropdown would not fully load when browser zoom was adjusted #4577 *(Donated by 4Science)*
    - Error handling on changing a mapped collection has been improved to display a clear 403 Forbidden when applicable. #5435 *(Donated by eScire)*
    - Fixed an issue in display of abstracts where words with dots were always treated as links. #2789 *(Donated by eScire)*
    - Fixed an issue with the "drop files" overlay on the submission form in Firefox. #1268 *(Donated by eScire)*
    - Pagination controls have been improved to center the active page and make it easier to browse nearby pages. #5494 *(Donated by eScire)*
- Performance and stability improvements:
    - Improved handling of details in the Event object to improve stability and reduce errors. #10683 *(Donated by 4Science)*
    - Improved performance on item pages with many bitstreams by reducing bitstream authorization requests. #5028 *(Donated by Tina Schönborn)*
    - Improved performance by optimizing the SQL query in the `findByEPerson` method. #11472 *(Donated by Toni Prieto)*
    - Improved loading times and cache behavior for the community list page. #9911 *(Donated by 4Science)*
    - Improved performance on the signposting endpoint by adding a count query. #12305 *(Donated by Tina Schönborn)*
    - Improved performance for relationship queries by caching the tables that store entity and relationship types. #12511 *(Donated by Atmire)*

    - DOIOrganiser bulk operations can now scale for hundreds of items due to adding DAO-level pagination. #9622 *(Donated by Atmire)*
- Browse and search fixes
    - Fixed bug in "Browse by Title" where searching for a title beginning with certain articles (e.g. A, As, O, Os) did not return accurate results. #4166 *(Donated by 4Science)*
    - Fixed bug on "Browse by" pages where the search box no longer worked after clicking on a value on the page. #2739 *(Donated by Atmire)*
    - Fixed bug on Browse by Issue Date where the date was treated as a filter rather than a start date for decades. #10055 *(Donated by Atmire)*
    - Fixed bug where exporting search results didn't work if the search contained quotation marks. #9589 *(Donated by Vir Softech)*
    - Fixed a bug in hierarchical vocabulary browse where only the first 20 matches to a query were rendered. #4500 *(Donated by Atmire)*
- Search Engine Optimization (SEO) fixes
    - Fixed bug where private (non-discoverable) item pages were missing the "noindex" meta tag that tells crawlers to avoid indexing the item. #4835 *(Donated by Atmire)*
    - Added 'rel="nofollow"' to all search facets on the search page to tell well-behaved bots not to follow those links. #4785 *(Donated by Alan Orth)*
- Statistics enhancements and fixes
    - Add several "usage-statistics.*" configurations to make it easier to quickly customize the statistical reports. #10142 *(Donated by Nicholas Woodward)*
    - Updated Solr-based internal statistics to only track downloads of bitstreams that are in a bundle listed in the existing "solr-statistics.query.filter.bundles" configuration. This defaults to only tracking downloads of bitstreams in the "ORIGINAL" bundle. #11033 *(Donated by Neki-IT)*
    - Fixed bug where IRUS statistics event processor used the wrong OAI-PMH url. #8852 *(Donated by Neki-IT)*
- REST API fixes
    - Fixed bug where patching metadata on unknown field cleared all DSO metadata. #10817 *(Donated by Atmire)*
    - Fixed a REST API bug where sending a PATCH request for an unknown metadata field could clear all object metadata. (This only impacts REST API and not the UI.) #10817(*Donated by Atmire)*
    - REST API has been updated to eager load CSRF tokens again, similar to DSpace 7.x. #11154 *(Donated by Neki-IT)*
- Fixed bug where PM2 was failing in cluster mode for localhost because it defaulted to IPv6 over IPv4. #4960 *(Donated by Atmire)*
- Fixed several CSS / style issues found since upgrading to Bootstrap 5. #4538 *(Donated by Atmire)*
- Fixed bug where Geospatial Map javascript code was loaded even when it was not used. #4447 *(Donated by Atmire)*
- Added missing Entity-related metadata fields to the Metadata Registry. #3353 *(Donated by Tim Donohue)*
- Fixed a large number of other small bugs. See Changes in 10.x for a list of all changes.
- *Major dependency updates*: Frontend updated to Angular 20. *(Donated by Atmire)*

New/Updated Language support

- Arabic () language updates donated by Laith Rastanawi (LaisRast)
- Bengali () language updates donated by Md. Topu Raihan (raihantopu)
- Traditional Chinese () added & donated by eScire
- Czech (eština) language updates donated by dataquest
- French (Français) language updates donated by Pierre Lasou (pilasou) and Carolyn Sullivan (Peredwel)
- German (Deutsch) language updates donated by Sascha Szott (saschaszott) and Eike Löhden (EikLoe)
- Gujarati () added & donated by DSquare Technologies
- Hungarian (Magyar) language updates donated by Monguz-Qulto and Nagy Akos (akoscomp)
- Italian (Italiano) language updates donated by 4Science
- Malayalam () added & donated by DSquare Technologies
- Marathi () added & donated by DSquare Technologies
- Odia () added & donated by DSquare Technologies
- Persian () added & donated by Shafi Habibi (sh-habibi)
- Portuguese (Português) language updates donated by Neki-IT, Ricardo Saraiva (rsaraivac), and Alfeu U. Tavares (autavares-dev)
- Russian () added & donated by Arvo Consultores
- Spanish (Español) language updates donated by Arvo Consultores
- Tamil () added & donated by DSquare Technologies
- Telugu () added & donated by DSquare Technologies
- Ukrainian (Y) language updates donated by Olexandr Shaposhnyk (oshaposhnyk)

10.0 Acknowledgments

DSpace 10.0 had 371,282 lines of code changed and 106 unique individuals contributing to either the frontend or backend.

Frontend / User Interface Acknowledgments

The following 68 individuals have contributed directly to the new DSpace (Angular) User Interface in this release (ordered by number of GitHub commits): Francesco Molinaro (FrancescoMolinaro), Alan Orth (alanorth), Tim Donohue (tdonohue), Alexandre Vryghem (alexandrevryghem), Giuseppe Digilio (atarix83), Guillermo Escalante (guillermo-escire), Zoltán Kanász-Nagy (kanasznagyzoltan), Jesiel Viana (jesielviana), Kim Shepherd (kshepherd), Mykhaylo Boychuk (Micheleboychuk), Jukka Lipka (jlipka), Eike Löhden (EikLoe), Jens Vannerum (jensvannerum), Pawel Zlakowski (pzlakowski), Oscar Chacón (oscar-escire), Matus Kasak, Maciej Kleban (Dawnkai), Sascha Szott (saschaszott), Gaurav Patel (GauravD2t), Shafi Habibi (sh-habibi), Yury Bondarenko (ybnd), Nicholas Woodward (nwoodward), Joran De Braekeleer (gingyx), Milan Majchrak (milanmajchrak), Andrea Barbasso (AndreaBarbasso), Andreas Awouters (AAwouters), Sergio Fernández Celorio (sergius02), Toni Prieto (toniprieto), Zahraa Chreim (ZahraaChreim-Atmire), im-shubham-vish, Abhinav Sidharthan (AbhinavS96), Pierre Lasou (pilasou), Bram Maegerman (bram-maegerman), Arya Khochare (Aryakoste), Md. Topu Raihan (raihantopu), Carolyn Sullivan (Peredwel), Amy Pillow (strawburster), Kristof De Langhe (Atmire-Kristof), Daniel Coelho (danielcoelhocgu), Johnny Mendes do Carmo (JohnnyMendesC), Kasinhou, Bram Luyten (bram-atmire), Abel Gómez (abelgomez), Adamo Fapohunda (AdamF42), Art Lowel (artlowel), John Abrahams (jabrah), Laith Rastanawi (LaisRast), Marcin Miosz (MMilosz), Marsa Haoua (marsaoua), Nima Behfourouz (nimabehforouz), Ricardo Saraiva (rsaraivac), Stefano Maffei (steph-ieffam), Nagy Akos (akoscomp), William Welling (tamu-sad-iii), Alfeu U. Tavares (autavares-dev), Jorge Rojas, Juraj Roka, Mattia Vianelli (Sondissimo), Olexandr Shaposhnyk (oshaposhnyk), Michaela Paurikova (Paurikova2), Roshan Singh (Roshan-Singh-

07), Shivani Rupnawar (ShivaniNR), Solomon Kahsai (solomona2003), Tina Schönborn (tinsch), Francesco Bacchelli (frabacche), Lakshmi Sravya Vedantham (LakshmiSravyaVedantham), Mukunda Rao Katta (MukundaKatta), Krzysztof Kubiak (pcg-kk).

The above contributor list was determined based on contributions to the "dspace-angular" project in GitHub between 9.0 (after May 23, 2025) and 10.0 using "git shortlog" on the `main` branch and excluding all merge commits: `git shortlog -s -n -e --no-merges --since 2025-05-23`

Backend / REST API Acknowledgments

The following 69 individuals have contributed directly to the DSpace backend (REST API, Java API, OAI-PMH, etc) in this release (ordered by number of commits): Tim Donohue (tdonohue), Adamo Fapohunda (AdamF42), Kim Shepherd (kshepherd), Alan Orth (alanorth), Johnny Mendes do Carmo (JohnnyMendesC), Vincenzo Mecca (vins01-4science), Mykhaylo Boychuk (Micheleboychuk), Stefano Maffei (steph-ieffam), Piaget Bouaka Donfack (PiagetBouaka), Quasipoint, Sascha Szott (saschaszott), Abhinav Sidharthan (AbhinavS96), Jesiel Viana (jesielviana), Nicholas Woodward (nwoodward), Miika Nurminen (minurmin), Paulo Graça (paulo-graca), Mark Wood (mwoodiupui), Bram Luyten (bram-atmire), Eike Löhden (EikLoe), Marcin Miosz (MMilosz), Toni Prieto (toniprieto), Alexandre Vryghem (alexandrevryghem), David Steelman (dsteelma-umd), Mark Patton (markpatton), Yury Bondarenko (ybnd), Max Nuding (max-nuding), Jens Vannerum (jensvannerum), Andreas Mahnke (mahnkong), Daniel Coelho (danielcoelhocgu), Elios Buzo (eliosbz), Joran De Braekeleer (gingyx), Kristof De Langhe (Atmire-Kristof), Lucas Varone (LucasVaroneW), Pierre Lasou (pilasou), Zahraa Chreim (ZahraaChreim-Atmire), Marsa Haoua (marsaoua), Milan Majchrak (milanmajchrak), Nathan Buckingham (ConfusionOrb221), Andreas Czerniak (ACz-UniBi), April Herron (aprilherron), Bram Maegerman (bram-maegerman), Francisco Carvalho (ciscocarvalho), Marie Verdonck (MarieVerdonck), Martin Walk (MW3000), Michaela Paurikova (Paurikova2), fribeiro-fccn, Igor Baptista da Costa (IgorBaptist4), im-shivamb, im-shubham-vish, Istvan Vig (parphis), Pascal-Nicolas Becker (pnbecker), jeonyunhwan, Jukka Lipka (jlipka), Shivam Sharma (shvmsrc), Raúl Soto (soto-raul), Andrea Barbasso (AndreaBarbasso), Alphonse Bendt (abendt), Christian Bethge (ChrisBethgster), Drew Heles (dheles), Francesco Pio Scognamiglio (francescopioscognamiglio), GHX5T-SOL, Jozef Misutka (vidiecan), Mattia Vianelli (Sondissimo), Ondej Košarko (kosarko), Pablo Picurelli Ortiz (superpegaso2703), Tina Schönborn (tinsch), Aliaksei Sheuchyk (aliaksei), Mohamed Eskander (eskander17), Uur Tafral (utafrali).

The above contributor list was determined based on contributions to the "DSpace" project in GitHub between 9.0 (after May 23, 2025) and 10.0 using "git shortlog" on the `main` branch and excluding all merge commits: `git shortlog -s -n -e --no-merges --since 2025-05-23`

Reviewer Acknowledgments

Thank you to everyone who has contributed by reviewing or testing contributions! The following 43 individuals have reviewed a pull request in the DSpace Backend or User Interface (ordered by number of pull requests reviewed): Tim Donohue (tdonohue), Nicholas Woodward (nwoodward), Bram Luyten (bram-atmire), Alan Orth (alanorth), Kim Shepherd (kshepherd), Marcin Miosz (MMilosz), Nathan Lourenço (nathanmlf), Lia Eggleston (lgeggleston), Stefano Maffei (steph-ieffam), Art Lowel (artlowel), Mark Wood (mwoodiupui), Eike Löhden (EikLoe), Gaurav Patel (GauravD2t), Vincenzo Mecca (vins01-4science), Yury Bondarenko (ybnd), Alexandre Vryghem (alexandrevryghem), Toni Prieto (toniprieto), Krzysztof Kubiak (pcg-kk), Kevin Van de Velde (KevinVdV), Ben Bosman (benbosman), Martin Walk (MW3000), Paulo Graça (paulo-graca), Giuseppe Digilio (atarix83), Andrea Barbasso (AndreaBarbasso), Alfeu Tavares (autavares-dev), Mykhaylo Boychuk (Micheleboychuk), Marie Verdonck (MarieVerdonck), Katharina Kaiser (kskaiser), Jen Cwiok (jrcwiok), Marsa Haoua (marsaoua), Francesco Molinaro (FrancescoMolinaro), Tina Schönborn (tinsch), Mark Patton (markpatton), Zhuotong Zhou (jxfzzzt), Philipp Rumpf (philipprumpf), Miika Nurminen (minurmin), Pierre Lasou (pilasou), April Herron (aprilherron), Thiago Rodrigues (t-rodrigues), Jesiel Viana (jesielviana), Nima Behforouz (nimabehforouz).

The above reviewer list was determined based on comments made as part of a review submission on Github, and does not include comments on PRs made outside of that process. One review is counted per reviewer per pull request. For additional details, see the script used to retrieve this list via Github API.

Testathon Acknowledgements

Thank you to everyone who participated in the DSpace 10.0 Testathon! The following 26 individuals reported testing results via the Test Plan: Lara Babic, Igor Baptista, Rafael Bértoli, Sean Carte, Victoria Cruz, Lia Eggleston, Mosqueira Felipe, Nathan Ferreira, Victor Ferreira, Kent Gerber, Aleksandra Gontarz, Fatih Güne, Marsa Haoua, Sina Haselmann, Marcella Huggard, Matus Kasak, Pierre Lasou, Holger Lenz, Colin Lukens, Johnny Mendes do Carmo, Daniel Pais, Jordan Pišanc, Grazia Quercia, Tina Schönborn, Max Sullivan, Peter Sutton-Long.

Additional Thanks

Additional thanks to our DSpace Leadership Group and DSpace Steering Group for their ongoing DSpace support and advice.

Thanks also to the various developer & community Working Groups who have worked diligently to help make DSpace 10 a reality. These include:

- DSpace Developers Team - This volunteer-based team leads the development of new releases. Anyone is welcome to join the Developer Meetings
- DSpace Community Advisory Team (DCAT) - This team helped organize/lead the DSpace 10.0 Testathon (to bang on the system to find any last bugs), and they also provided us with advice on features, etc.

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. Acknowledgments to those left off will be made in future releases.

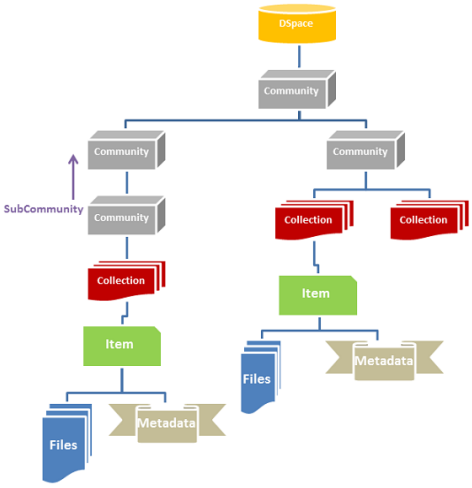
Functional Overview

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

1 Online access to your digital assets
    1.1 Full-text search
    1.2 Navigation
    1.3 Supported file types
    1.4 Optimized for Google Indexing
    1.5 OpenURL Support
    1.6 Support for modern browsers
2 Metadata Management
    2.1 Metadata
    2.2 Choice Management and Authority Control
3 Licensing
    3.1 Collection and Community Licenses
    3.2 License granted by the submitter to the repository
    3.3 Creative Commons Support for DSpace Items
4 Persistent URLs and Identifiers
    4.1 Handles
    4.2 Bitstream 'Persistent' Identifiers
5 Getting content into DSpace
    5.1 The Manual DSpace Submission and Workflow System
        5.1.1 Workflow Steps
        5.1.2 Submission Workflow in DSpace
    5.2 Command line import facilities
    5.3 Registration for externally hosted files
    5.4 SWORD Support
6 Getting content out of DSpace
    6.1 OAI Support
    6.2 Signposting
    6.3 Command Line Export Facilities
    6.4 Packager Plugins
    6.5 Crosswalk Plugins
    6.6 Supervision and Collaboration
7 User Management
    7.1 User Accounts (E-Person)
    7.2 Subscriptions
    7.3 Groups
8 Access Control
    8.1 Authentication
    8.2 Authorization
9 Usage Metrics
    9.1 Item, Collection and Community Usage Statistics
    9.2 System Statistics
10 Digital Preservation
    10.1 Checksum Checker
11 System Design
    11.1 Data Model
    11.2 Amazon S3 Support

Online access to your digital assets

The online presentation of your content in an organized tree of Communities and Collections is a main feature of DSpace. Users can access pages for individual items, these are metadata descriptions together with files available for download. The structure is summarised in this diagram (click to see the image at full size).



Full-text search

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

Navigation

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

- **Searching** for one or more keywords in metadata or extracted full-text
- **Faceted browsing** through any field provided in the item description.
- Through **external reference**, such as a Handle
- By clicking on Community and Collection titles to explore their contents

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.

For more information on Search/Browse functionality in DSpace, see Discovery.

Supported file types

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

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

By default, DSpace only recognizes specific file types, as defined in its Bitstream Format Registry. The default Bitstream Format Registry recognizes many common file formats, but it can be enhanced at your local institution via the Admin User Interface.

Optimized for Google Indexing

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

OpenURL Support

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

Support for modern browsers

The DSpace developer community aims to rely on modern web standards and well tested libraries where possible. As a rule of thumb, users can expect that the DSpace web interfaces work on modern web browsers. DSpace developers routinely test new interface developments on recent versions of Firefox, Safari, Chrome and Microsoft Edge. Because of fast moving, automatic, incremental updates to these browsers, support is no longer targeted at specific versions of these browsers. (Please note that we do not recommend or support using Internet Explorer as it is considered "end of life" by Microsoft.)

Metadata Management

Metadata

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

- **Descriptive Metadata**: DSpace can support multiple flat metadata schemas for describing an item. A qualified Dublin Core metadata schema loosely based on the Library Application Profile set of elements and qualifiers is provided by default. This default schema is described in more detail in Metadata and Bitstream Format Registries. 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.
- **Administrative Metadata**: This includes preservation metadata, provenance and authorization policy data. Most of this is held within DSpace's relational DBMS schema. Provenance metadata (prose) is stored in Dublin Core records. Additionally, some other administrative metadata (for example, bitstream byte sizes and MIME types) is replicated in Dublin Core records so that it is easily accessible outside of DSpace.
- **Structural Metadata**: This includes information about how to present an item, or bitstreams within an item, to an end-user, and the relationships between constituent parts of the item. As an example, consider a thesis consisting of a number of TIFF images, each depicting a single page of the thesis. Structural metadata would include the fact that each image is a single page, and the ordering of the TIFF images/pages. Structural metadata in DSpace is currently fairly basic; within an item, bitstreams can be arranged into separate bundles as described above. A bundle may also optionally have a *primary bitstream*. This is currently used by the HTML support to indicate which bitstream in the bundle is the first HTML file to send to a browser. In addition to some basic technical metadata, a bitstream also has a 'sequence ID' that uniquely identifies it within an item. This is used to produce a 'persistent' bitstream identifier for each bitstream. Additional structural metadata can be stored in serialized bitstreams, but DSpace does not currently understand this natively.

Choice Management and Authority Control

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

Introduction and Motivation

Definitions

Choice Management

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

Authority Control

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

About Authority Control

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

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

Authority control is different from the controlled vocabulary of keywords already implemented in the submission UI:

1. **Authorities are external to DSpace.** The source of authority control is typically an external database or network resource.
    - Plug-in architecture makes it easy to integrate new authorities without modifying any core code.
2. This authority proposal impacts all phases of metadata management.
    - The keyword vocabularies are only for the submission UI.

- Authority control is asserted everywhere metadata values are changed, including unattended/batch submission, 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.

Licensing

DSpace offers support for licenses on different levels

Collection and Community Licenses

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

License granted by the submitter to the repository

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

Creative Commons Support for DSpace Items

DSpace provides support for Creative Commons licenses to be attached to items in the repository. They represent an alternative to traditional copyright. To learn more about Creative Commons, visit their website. Support for license selection is controlled by a site-wide configuration option, and since license selection involves interaction with the Creative Commons website, additional parameters may be configured to work with a proxy server. If the option is enabled, users may select a Creative Commons license during the submission process, or select to don't assign a Creative Commons license at all. If a selection is made, metadata and a copy of the license in the RDF format 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. The RDF license is embedded in the html page of the item to allow machine understanding of the licensing terms. For specifics of how to configure and use Creative Commons licenses, see the configuration section.

Persistent URLs and Identifiers

Handles

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

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

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

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

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

Handles can be written in two forms:

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

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

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

Bitstream 'Persistent' Identifiers

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

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

dspace url/bitstream/handle/sequence ID/filename

For example:

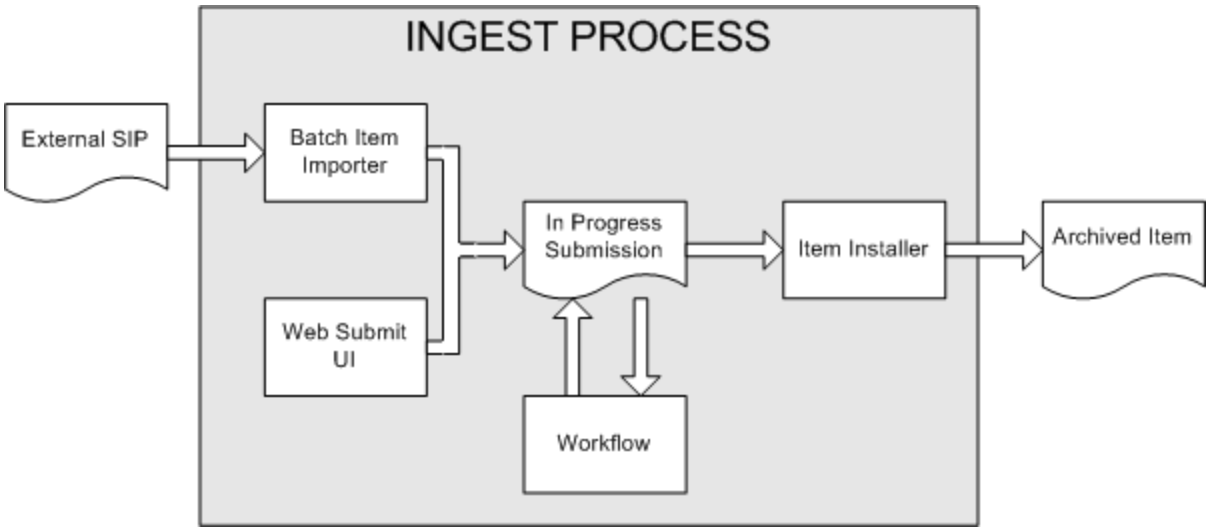
```
https://dspace.myu.edu/bitstream/123.456/789/24/foo.html
```

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

Getting content into DSpace

The Manual DSpace Submission and Workflow System

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



DSpace Ingest Process

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

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

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

- Assigns an accession date
- Adds a "date.available" value to the Dublin Core metadata record of the item

- Adds an issue date if none already present
- Adds a provenance message (including bitstream checksums)
- Assigns a Handle persistent identifier
- Adds the item to the target collection, and adds appropriate authorization policies
- Adds the new item to the search and browse index

Workflow Steps

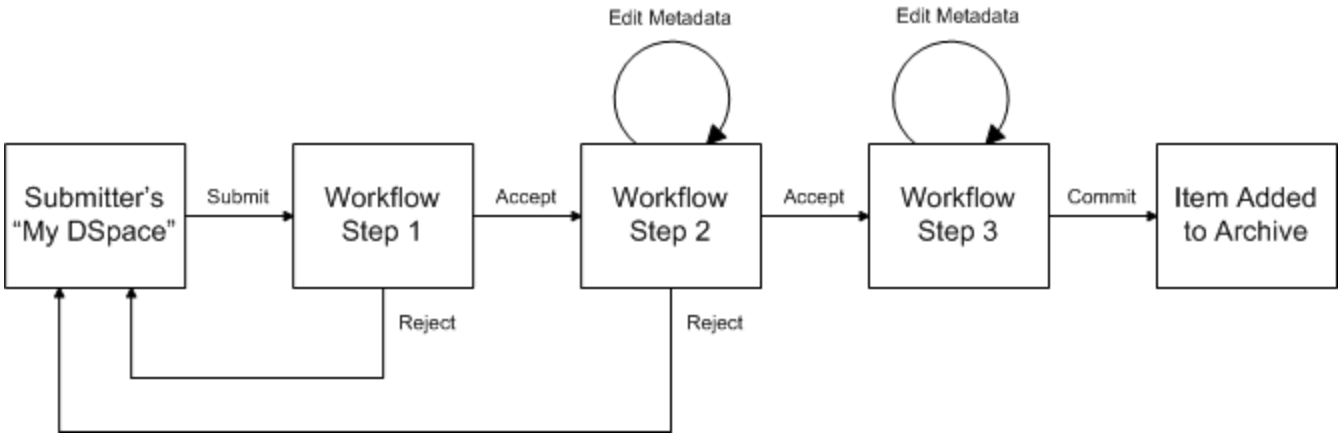
By default, a collection's workflow may have up to three steps. Each collection may have an associated e-person group for performing each step; if no group is associated with a certain step, that step is skipped. If a collection has no e-person groups associated with any step, submissions to that collection are installed straight into the main archive. Keep in mind, however, that this is only the **default** behavior, and the workflow process can be configured /customized easily, see Configurable Workflow.

In other words, the default 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
review (step 1)	Can accept submission for inclusion, or reject submission.
edit (step 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.
finaledit (step 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.

Command line import facilities

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

DSpace also includes various package importer tools, which support many common content packaging formats like METS. For more information see Package Importer and Exporter. Additionally, DSpace can import/export Archival Information Packages (AIPs), see AIP Backup and Restore.

Registration for externally hosted files

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

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. See also SWORDv1 Server and SWORDv2 Server.

Getting content out of DSpace

OAI Support

The Open Archives Initiative has developed a protocol for metadata harvesting. This allows sites to programmatically retrieve or 'harvest' the metadata from several sources, and offer services using that metadata, such as indexing or linking services. Such a service could allow users to access information from a large number of sites from one place.

DSpace exposes the Dublin Core metadata for items that are publicly (anonymously) accessible. Additionally, the collection structure is also exposed via the OAI protocol's 'sets' mechanism. OCLC's open source OAICat framework is used to provide this functionality.

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

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

Signposting

DSpace supports FAIR Signposting Profile at Level 2: By supporting the FAIR Signposting Profile at Level 2, your platform demonstrates a commitment to improving the machine accessibility, interoperability, and reusability of scholarly resources. It ensures that the information you provide is standardized, consistent, and easily navigable by both human users and machine agents, contributing to a more efficient and FAIR scholarly web ecosystem. For more information see Signposting.

Command Line Export Facilities

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

DSpace also includes various package exporter tools, which support many common content packaging formats like METS. For more information see Package Importer and Exporter. Additionally, DSpace can import/export Archival Information Packages (AIPs), see AIP Backup and Restore.

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), so it is easy to add new packagers specific to the needs of your site. You do not have to supply both an ingester and disseminator for each format; it is perfectly acceptable to just implement one of them.

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

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

Crosswalk Plugins

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

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

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

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

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

Supervision and Collaboration

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

- Full editorial control
- View item contents

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.

See Supervision Orders for more details.

User Management

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

User Accounts (E-Person)

DSpace holds the following information about each e-person:

- E-mail address
- First and last names
- Whether the user is able to log in to the system via the Web UI, and whether they must use an X509 certificate to do so;
- A password (encrypted), if appropriate
- A list of collections for which the e-person wishes to be notified of new items
- Whether the e-person 'self-registered' with the system; that is, whether the system created the e-person record automatically as a result of the end-user independently registering with the system, as opposed to the e-person record being generated from the institution's personnel database, for example.
- The network ID for the corresponding LDAP record, if LDAP authentication is used for this E-Person.

Subscriptions

Not yet been implemented. This listed in Tier 3 (see #6 in Tier 3): DSpace Release 7.0 Status#Tier3:MediumPriority

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

Groups

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

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

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

Access Control

Authentication

Authentication is when an application session positively identifies itself as belonging to an E-Person and/or Group. In DSpace, 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.

For more information see Authentication Plugins

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.

Usage Metrics

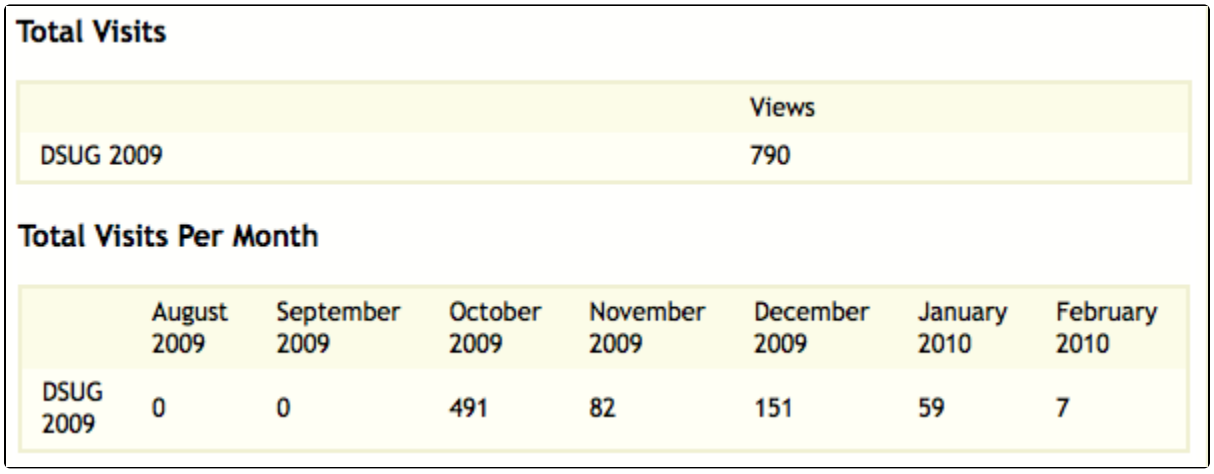
DSpace is equipped with SOLR based infrastructure to log and display pageviews and file downloads.

Item, Collection and Community Usage Statistics

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

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

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



Total Visits

	Views
DSUG 2009	790

Total Visits Per Month

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

System Statistics

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

The report includes following sections

- A customizable general overview of activities in the archive, by default including:
  - Number of items archived
  - Number of bitstream views
  - Number of item page views
  - Number of collection page views
  - Number of community page views
  - Number of user logins
  - Number of searches performed
  - Number of license rejections
  - Number of OAI Requests
- Customizable summary of archive contents
- Broken-down list of item viewings
- A full break-down of all performed actions
- User logins
- Most popular searches
- Log Level Information
- Processing information!stats_genrl_overview.png!
  The results of statistical analysis can be presented on a by-month and an in-total report, and are available via the user interface. The reports can also either be made public or restricted to administrator access only.

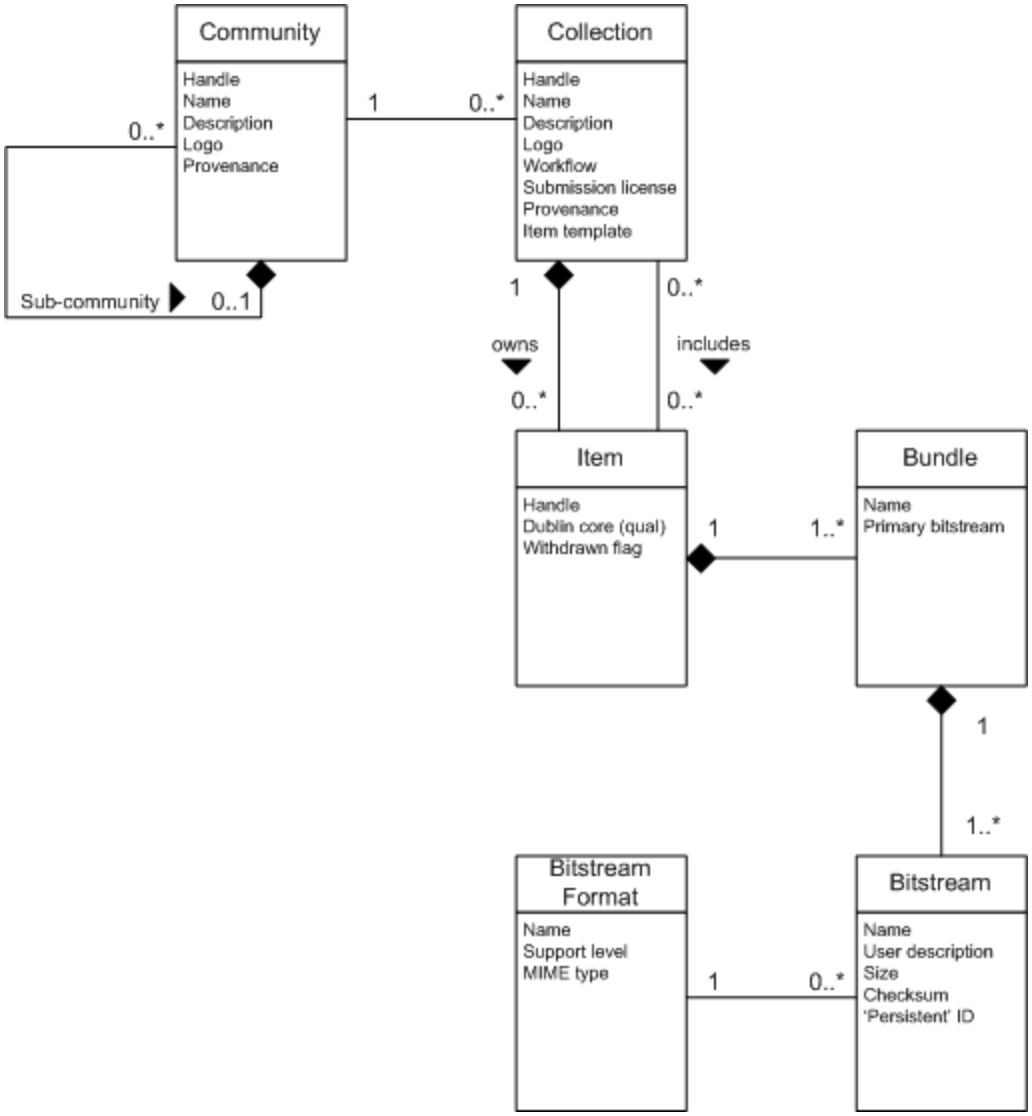
Digital Preservation

Checksum Checker

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

System Design

Data Model



Data Model Diagram

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

Communities contain *collections*, which are groupings of related content. A collection may only appear in one community at this time.

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 icenommons 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

Amazon S3 Support

DSpace offers two means for storing bitstreams. The first is in the file system on the server. The second is using Amazon S3. For more information, see Storage Layer

Technology Overview

DSpace open source software is free to use, and community supported.

DSpace consists of both a frontend (User Interface) and a backend (REST API & other machine interfaces). A brief overview of the technologies used for each is provided below.

- DSpace Frontend (UI) Technologies
- DSpace Backend (REST API) Technologies
- How the Frontend and Backend interact

DSpace Frontend (UI) Technologies

The DSpace Frontend provides the User Interface which allows people to interact with DSpace. It requires a DSpace backend, and cannot be run standalone.

The DSpace Frontend is built on the Angular platform, written in the Typescript language. It uses Bootstrap & HTML5 for theming/styling & strives for WCAG 2.1 AA alignment. The frontend also uses Angular Universal for "server-side rendering", which allows it to function even when Javascript is unavailable in the browser. For more information on Angular Universal, see the the Angular University guide.

More information on installing the DSpace Frontend can be found in the Installing DSpace guide.

DSpace Backend (REST API) Technologies

The DSpace Backend provides the REST API, which is required by the DSpace Frontend. It also provides additional machine interfaces for interacting with data in DSpace, such as OAI-PMH, SWORDv2 Server, SWORDv1 Server and various command-line (CLI) tools. The DSpace backend can be run standalone, but it doesn't provide a user friendly web interface (which is why the DSpace frontend is recommended).

The DSpace Backend is built on Spring Boot, written in Java. The REST API portion of the backend is built on Spring Technologies, including Spring REST, Spring HATEOAS, and aligns with Spring Data REST. The REST API uses the Spring Data REST Hal Browser as a basic web interface for exploring the REST API. All REST API responses are returned in JSON format.

The DSpace Backend requires a relational database (usually PostgreSQL), used to store all the metadata and relationships between objects. All files uploaded into DSpace are stored on the filesystem (any operating system is supported). Apache Solr is also required, and is used to index all objects for searching/browsing.

More information on installing the DSpace Backend can be found in the Installing DSpace guide. More information on the REST API specifically can be found in our REST Contract.

How the Frontend and Backend interact

Here is a high level overview of what happens when a user interacts with DSpace when the user interface is running in *production* mode:

1. *Initial static page via server-side rendering (SSR):* When a user initially visits any page in the DSpace user interface (UI), this triggers server-side rendering (SSR) via Angular Universal. This means that the UI (Javascript) application is run *on the server* by Node.js. The result is that a *static* HTML page is generated, which will be sent back to the user.
    a. This process of rendering the static HTML page will result in Node.js making requests to REST API to gather all the data necessary to build the *static* HTML page.
2. *Static page is dynamically replaced by UI application:* The user briefly sees the generated static HTML page *while the UI (Javascript) application is downloading to their browser* . This allows the user to immediately see the DSpace User Interface even before it becomes interactive. As soon as the UI application finishes downloading, it dynamically replaces that static HTML page, making the User Interface interactive to the user. (The time between the UI page appearing and becoming interactive is usually unnoticeable to a user.) This entire process is handled by Angular Universal.
3. *Interactions with the UI application send requests to the REST API (client-side rendering):* As soon as the UI becomes interactive, it runs entirely in the user's browser (as any other Javascript application). This means that when the user interacts with the application (by clicking links/buttons or typing in fields, etc), this will send requests from the user's browser to the REST API (backend). This is called client-side rendering (CSR) as all HTML is generated within the user's browser.
    a. At this point, every action in the User Interface will generate one or more requests to the REST API to gather necessary data. These requests are all visible in the user's browser (in the "Network" tab of the browser's "Developer tools").

Keep in mind, SSR can be potentially taxing for very large pages with a lot of objects or data display. This is because Node.js has to make requests to the REST API to gather all the data for the page before rendering the static HTML. Because of this, we do also document some Performance Tuning suggestions for the User Interface (e.g. there is an option to cache these SSR generated static pages in order to generate them less frequently).

Some bots and clients may use server-side rendering at all times

For bots or clients *without the ability to run Javascript*, every page request will trigger SSR (server-side rendering). This is because the static HTML page can never be dynamically replaced by the User Interface application (in step 2 above). However, this behavior is necessary to support Search Engine Optimization. Some search engine bots cannot run Javascript & therefore cannot index sites which do not generate static HTML pages.

Running the user interface in development mode disables SSR and may impact SEO

Running the user interface (frontend) in *development* mode will only utilize client-side rendering (CSR) (as described in step 3 above). This means server-side rendering (SSR) will never occur, and all HTML will be generated in the user's browser. The result is that bots or clients *without the ability to run Javascript* will be unable to interact with the site (which can negatively impact Search Engine Optimization)

Installing DSpace

- Installation Overview
- Installing the Backend (Server API)
  - Backend Requirements
  - Backend Installation
- Installing the Frontend (User Interface)
  - Frontend Requirements
  - Frontend Installation
- What Next?
- Common Installation Issues
  - Troubleshoot an error or find detailed error messages
  - User Interface never appears (no content appears) or "Proxy server received an invalid response"
  - User Interface partially load but then spins (never fully loads or some content doesn't load)
  - "500 Service Unavailable" from the User Interface
  - "No _links section found at..." error from User Interface
  - "RangeError: Maximum call stack size exceeded"
  - "XMLHttpRequest.. has been blocked by CORS policy" or "CORS error" or "Invalid CORS request"
  - Cannot login from the User Interface with a password that I know is valid
  - "403 Forbidden" error with a message that says "Access is denied. Invalid CSRF Token"
  - Using a Self-Signed SSL Certificate causes the Frontend to not be able to access the Backend
  - My REST API is running under HTTPS, but some of its "link" URLs are switching to HTTP
  - My User Interface's robots.txt has incorrect sitemap URLs
  - Cannot upload file from User Interface
  - Javascript heap out of memory
  - Solr responds with "Expected mime type application/octet-stream but got text/html" (404 Not Found)
  - Database errors occur when you run ant fresh_install

Installation Overview

Try out DSpace before you install

If you'd like to quickly try out DSpace 10 before a full installation, see Try out DSpace 10 for instructions on a quick install via Docker.

As of version 7 (and later), the DSpace application is split into a "frontend" (User Interface) and a "backend" (Server API). Most institutions will want to install BOTH. However, you can decide whether to run them on the same machine or separate machines.

- The DSpace Frontend consists of a User Interface built on Angular.io. It is a Node.js web application, i.e. once it is built/compiled, it only require Node.js to run. It cannot be run "standalone", as it *requires* a valid DSpace Backend to function. The frontend provides all user-facing functionality.
- The DSpace Backend consists of a Server API ("server" webapp), built on Spring Boot. It is a Java web application. It can be run standalone, however it has no user interface. The backend provides all machine-based interfaces, including the REST API, OAI-PMH, SWORD (v1 and v2) and RDF.

We recommend installing the Backend **first**, as the Frontend requires a valid Backend to run properly.

Installing the Backend (Server API)

Backend Requirements

- UNIX-like OS or Microsoft Windows
- Java JDK 21 (OpenJDK or Oracle JDK)
- Apache Maven 3.9.x or above (Java build tool)
  - Configuring a Maven Proxy
- Apache Ant 1.10.x or later (Java build tool)
- Relational Database (PostgreSQL)
  - PostgreSQL 14.x, 15.x, 16.x or 17.x
- Apache Solr 9.x (full-text index/search service)
- (Optional) Servlet Engine (Apache Tomcat 10.1.x, Jetty, Caucho Resin or equivalent)
- (Optional) IP to City Database for Location-based Statistics

UNIX-like OS or Microsoft Windows

- UNIX-like operating system (Linux, HP/UX, Mac OSX, etc.) : Many distributions of Linux/Unix come with some of the dependencies below pre-installed or easily installed via updates. You should consult your particular distribution's documentation or local system administrators to determine what is already available.
- Microsoft Windows: While DSpace can be run on Windows servers, most institutions tend to run it on a UNIX-like operating system.

Java JDK 21 (OpenJDK or Oracle JDK)

- OpenJDK download and installation instructions can be found here http://openjdk.java.net/install/. Most operating systems provide an easy path to install OpenJDK. Just be sure to install the full JDK (development kit), and not the JRE (which is often the default example).
- Oracle's Java can be downloaded from the following location: http://www.oracle.com/technetwork/java/javase/downloads/index.html. Make sure to download the appropriate version of the Java SE JDK.

Make sure to install the JDK and not just the JRE

DSpace requires the full JDK (Java Development Kit) be installed, rather than just the JRE (Java Runtime Environment). So, please be sure that you are installing the full JDK and not just the JRE.

Prior versions of Java are not supported

Older versions of Java are unsupported. This includes JDK v11-v20. You MUST be running JDK v21.

We highly recommend running only Java LTS (Long Term Support) releases in Production, as non-LTS releases may not receive ongoing security fixes. As of this DSpace release, JDK 21 and JDK 25 are the two most recent Java LTS releases. As soon as the next Java LTS release is available, we will analyze it for compatibility with this release of DSpace. For more information on Java releases, see the Java roadmaps for Oracle and/or OpenJDK.

Apache Maven 3.9.x or above (Java build tool)

We recommend using the most recent version of Maven that you can, as newer releases may include performance improvements and security updates. We recommend avoiding any that are "end of life" per https://maven.apache.org/docs/history.html

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 http://maven.apache.org/download.html It is also provided via many operating system package managers.

Configuring a Maven Proxy

You can configure a proxy to use for some or all of your HTTP requests in Maven. The username and password are only required if your proxy requires basic authentication (note that later releases may support storing your passwords in a secured keystore, in the meantime, 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>
```

Apache Ant 1.10.x or later (Java build tool)

Apache Ant is required for the second stage of the build process (deploying/installing the application). First, Maven is used to construct the installer (`[dspace-source]/dspace/target/dspace-installer`), after which Ant is used to install/deploy DSpace to the installation directory.

Ant can be downloaded from the following location: http://ant.apache.org It is also provided via many operating system package managers.

Relational Database (PostgreSQL)

PostgreSQL 14.x, 15.x, 16.x or 17.x

"pgcrypto" extension is no longer required for PostgreSQL, as UUID creation is supported out-of-the-box in Postgres 13 and later.

- PostgreSQL can be downloaded from http://www.postgresql.org/. It is also provided via many operating system package managers.
  - Make sure to select a version of PostgreSQL that is still under support from the PostgreSQL team.
  - If the version of Postgres provided by your package manager is outdated, you may wish to use one of the official PostgreSQL provided repositories:
    - Linux users can select their OS of choice for detailed instructions on using the official PostgreSQL apt or yum repository: http://www.postgresql.org/download/linux/

        - Windows users will need to use the windows installer: http://www.postgresql.org/download/windows/
        - Mac OSX users can choose their preferred installation method: http://www.postgresql.org/download/macosx/
- Unicode (specifically UTF-8) support must be enabled (but this is enabled by default).
- Once installed, you need to enable TCP/IP connections (DSpace uses JDBC):
    - In `postgresql.conf`: uncomment the line starting: `listen_addresses = 'localhost'`. This is the default, in recent PostgreSQL releases, but you should at least check it.
    - Then tighten up security a bit by editing `pg_hba.conf` and adding this line:

      ```
      host dspace dspace 127.0.0.1 255.255.255.255 md5
      ```

      This should appear *before* any lines matching `all` databases, because the first matching rule governs.
    - Then restart PostgreSQL.

Apache Solr 9.x (full-text index/search service)

Solr 8.x is "end of life". You should no longer be running DSpace with Apache Solr 8.x. If, for some reason, you are blocked from upgrading to Solr 9.x, it is possible to run DSpace with Solr 8.x provided that you undo the "solrconfig.xml" changes in this commit: https://github.com/DSpace/DSpace/pull/10627/commits/150d8f435718e57ef964e8b4929d697318a07784

Solr 9.8 and later require setting "solr.config.lib.enabled=true"

If you are using Solr 9.8 or later, you **must** specify "`-Dsolr.config.lib.enabled=true`" in your `SOLR_OPTS` environment variable, or on the command-line when you start Solr. For example:

```
export SOLR_OPTS="-Dsolr.config.lib.enabled=true"
```

This "solr.config.lib.enabled" setting allows for "solrconfig.xml" configuration files to still use "<lib>" tags to import plugins. This is **required** for DSpace, because several of our Solr cores (search, oai, etc) import plugins via these "<lib>" tags.

On macOS, Solr 9 may initially throw an IllegalStateException

On macOS, starting up Solr 9 may result in an exception that looks like this:

java.lang.IllegalStateException: java.security.AccessControlException: access denied ("java.io.FilePermission" "/private/var/folders/[some-path]")

If you encounter this, it's a sign that Solr doesn't have permission to read the specified file/directory. The fix is to start Solr with "`-Djava.security.manager=allow`" added to your `SOLR_OPTS` environment variable or specified on the command-line when you start Solr.

Make sure to install Solr with Authentication disabled (which is the default). DSpace does not yet support authentication to Solr (see https://github.com/DSpace/DSpace/issues/3169). Instead, we recommend placing Solr behind a firewall and/or ensuring port 8983 (which Solr runs on) is not available for public/anonymous access on the web. Solr only needs to be accessible to requests from the DSpace backend.

Solr can be obtained at the Apache Software Foundation site for Solr. You may wish to read portions of the quick-start tutorial to make yourself familiar with Solr's layout and operation. Unpack a Solr .tgz or .zip archive in a place where you keep software that is not handled by your operating system's package management tools, and arrange to have it running whenever DSpace is running. You should ensure that Solr's index directories will have plenty of room to grow. You should also ensure that port 8983 is not in use by something else, or configure Solr to use a different port.

If you are looking for a good place to put Solr, consider `/opt` or `/usr/local`. You can simply unpack Solr in one place and use it. Or you can configure Solr to keep its indexes elsewhere, if you need to – see the Solr documentation for how to do this.

It is not necessary to dedicate a Solr instance to DSpace, if you already have one and want to use it. Simply copy DSpace's cores to a place where they will be discovered by Solr. See below.

(Optional) Servlet Engine (Apache Tomcat 10.1.x, Jetty, Caucho Resin or equivalent)

Tomcat is optional depending on the installation approach you choose

As of DSpace 8, two deployment options exist for the backend:

1. The web application installation (traditional approach) still requires installing Tomcat and deploying the DSpace backend ("server") web application into your Tomcat installation. This approach is the same as DSpace 7 and below.
2. (NEW) A new Runnable JAR is available for the DSpace backend which embeds Tomcat within it and is fully executable on its own (see step 12: "Deploy web application" below for more details).

If you choose the first approach, Tomcat is required. If you choose the second, you no longer need to install Tomcat.

Older versions of Tomcat or Jetty are not supported

The DSpace backend can no longer be run on Tomcat 9 as it has been updated to Spring 6 / Spring Boot v3 to support Jakarta Enterprise Edition 9+.

If you are using a different servlet engine, you must ensure it is compatible with Jakarta EE 9+ (e.g. Jetty must be version 11 or later)

- **Apache Tomcat 10.1.x.** Tomcat can be downloaded from the following location: http://tomcat.apache.org. It is also provided via many operating system package managers.
    - *The Tomcat owner* (i.e. the user that Tomcat runs as) ***must have read/write access*** *to the DSpace installation directory* (i.e. `[dspace]`). There are a few common ways this may be achieved:
        - One option is to specifically give the Tomcat user (often named "tomcat") ownership of the [dspace] directories, for example:

```
# Change [dspace] and all subfolders to be owned by "tomcat"
chown -R tomcat:tomcat [dspace]
```

        - Another option is to have Tomcat itself *run as* a new user named "dspace" (see installation instructions below). Some operating systems make modifying the Tomcat "run as" user easily modifiable via an environment variable named TOMCAT_USER. This option may be more desirable if you have multiple Tomcat instances running, and you do not want all of them to run under the same Tomcat owner.
    - *On Debian systems*, you may also need to modify or override the "tomcat.service" file to specify the DSpace installation directory in the list of ReadWritePaths. For example:

```
# Replace [dspace] with the full path of your DSpace install
ReadWritePaths=[dspace]
```

    - You need to ensure that Tomcat a) has 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"*. One place where you can do this, is by creating a setenv.sh file within the /bin directory where the tomcat startup.sh script lives. If setenv.sh is present, it will be automatically loaded when using startup.sh.
    - **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" protocol="HTTP/1.1"
              minSpareThreads="25"
              redirectPort="8443"
              connectionTimeout="20000"
              maxParameterCount="1000"
              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*. You should set the URIEncoding even if you are running Tomcat behind a reverse proxy (Apache HTTPD, Nginx, etc.) via AJP.
- **Jetty 11+ or Caucho Resin**
    - NOTE: DSpace is not actively tested on these servlet engines. That said, DSpace *should* be able to run on a Tomcat-equivalent servlet Engine, such as Jetty (https://www.eclipse.org/jetty/) or Caucho Resin (http://www.caucho.com/). If you choose to use a different servlet container, please ensure that it supports Jakarta EE 9+ (e.g. Jetty must be version 11 or later)
    - Jetty and Resin are configured for correct handling of UTF-8 by default.

(Optional) IP to City Database for Location-based Statistics

Optionally, if you wish to record the geographic locations of clients in DSpace usage statistics records, you will need to install (and regularly update) one of the following:

- Either, a copy of MaxMind's GeoLite City database (in MMDB format)
    - NOTE: Installing MaxMind GeoLite2 is *free*. However, you **must** sign up for a (free) MaxMind account in order to obtain a license key to use the GeoLite2 database.
    - You may download GeoLite2 directly from MaxMind, or many Linux distributions provide the `geoipupdate` tool directly via their package manager. You will still need to configure your license key prior to usage.
    - Once the "GeoLite2-City.mmdb" database file is installed on your system, you will need to configure its location as the value of `usage-statistics.dbfile` in your `local.cfg` configuration file.
    - See the "Managing the City Database File" section of SOLR Statistics for more information about using a City Database with DSpace.
- Or, you can alternatively use/install DB-IP's City Lite database (in MMDB format)
    - This database is also free to use, but does **not** require an account to download.
    - Once the "dbip-city-lite.mmdb" database file is installed on your system, you will need to configure its location as the value of `usage-statistics.dbfile` in your `local.cfg` configuration file.
    - See the "Managing the City Database File" section of SOLR Statistics for more information about using a City Database with DSpace.

Backend Installation

1. Install all the Backend Requirements listed above.
2. **Create a DSpace operating system user *(optional)*** . As noted in the prerequisites above, Tomcat (or Jetty, etc) **must run as** an operating system user account that has full read/write access to the DSpace installation directory (i.e. `[dspace]`). Either you must ensure the Tomcat owner also owns `[dspace]`, OR you can create a new "dspace" user account, and ensure that Tomcat also runs as that account:

```
useradd -m dspace
```

The choice that makes the most sense for you will probably depend on how you installed your servlet container (Tomcat/Jetty/etc). If you installed it from source, you will need to create a user account to run it, and that account can be named anything, e.g. 'dspace'. If you used your operating system's package manager to install the container, then a user account should have been created as part of that process and it will be much easier to use that account than to try to change it.

3. **Download** the latest DSpace release from the DSpace GitHub Repository. You can choose to either download the zip or tar.gz file provided by GitHub, or you can use "git" to checkout the appropriate tag (e.g. `dspace-10.0`) or branch.
4. **Unpack the DSpace software**. After downloading the software, based on the compression file format, choose one of the following methods to unpack your software:
    a. **Zip file**. If you downloaded *dspace-10.0.zip* do the following:

    ```
    unzip dspace-10.0.zip
    ```

    b. **.gz file**. If you downloaded *dspace-9.0.tar.gz* do the following:

    ```
    gunzip -c dspace-10.0.tar.gz | tar -xf -
    ```

    For ease of reference, we will refer to the location of this unzipped version of the DSpace release as *[dspace-source]* in the remainder of these instructions. After unpacking the file, the user may wish to change the ownership of the dspace-10.x folder to the "dspace" user. (And you may need to change the group).
5. **Database Setup for PostgreSQL:**
    - Create a `dspace` database user (this user can have any name, but we'll assume you name it "dspace"). This is entirely separate from the `dspace` operating-system user created above:

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

    You will be prompted (twice) for a password for the new `dspace` user. Then you'll be prompted for the password of the PostgreSQL superuser (`postgres`).
    - Create a `dspace` database, owned by the `dspace` PostgreSQL user. Similar to the previous step, this can only be done by a "superuser" account in PostgreSQL (e.g. `postgres`):

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

    You will be prompted for the password of the PostgreSQL superuser (`postgres`).
6. **Initial Configuration (local.cfg):** Create your own `[dspace-source]/dspace/config/local.cfg` configuration file. You may wish to simply copy the provided `[dspace-source]/dspace/config/local.cfg.EXAMPLE`. This local.cfg file can be used to store *any* configuration changes that you wish to make which are local to your installation (see local.cfg configuration file documentation). ANY setting may be copied into this local.cfg file from the dspace.cfg or any other *.cfg file in order to override the default setting (see note below). For the initial installation of DSpace, there are some key settings you'll likely want to override. Those are provided in the `[dspace-source]/dspace/config/local.cfg.EXAMPLE`. (NOTE: Settings followed with an asterisk (*) are highly recommended, while all others are optional during initial installation and may be customized at a later time.)
    - `dspace.dir*` - must be set to the *[dspace]* (installation) directory (*NOTE: On Windows be sure to use forward slashes for the directory path!* For example: "`C:/dspace`" is a valid path for Windows.)
    - `dspace.server.url*` - complete URL of this DSpace backend (including port and any subpath). **Do not end with '/'**. For example: http://localhost:8080/server
    - `dspace.ui.url*` - complete URL of the DSpace frontend (including port and any subpath). REQUIRED for the REST API to fully trust requests from the DSpace frontend. **Do not end with '/'**. For example: http://localhost:4000
    - `dspace.name` - Human-readable, "proper" name of your server, e.g. "My Digital Library".
    - `solr.server*` - complete URL of the Solr server. DSpace makes use of Solr for indexing purposes. http://localhost:8983/solr unless you changed the port or installed Solr on some other host.
    - `default.language` - Default language for all metadata values (defaults to "en_US")
    - `db.url*` - The full JDBC URL to your database (examples are provided in the `local.cfg.EXAMPLE`)
    - `db.driver*` - Which database driver to use for PostgreSQL (default should be fine)
    - `db.dialect*` - Which database dialect to use for PostgreSQL (default should be fine)
    - `db.username*` - the database username used in the previous step.
    - `db.password*` - the database password used in the previous step.
    - `db.schema*` - the database schema to use (examples are provided in the local.cfg.EXAMPLE)
    - `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)

Your local.cfg file can override ANY settings from other *.cfg files in DSpace

The provided `local.cfg.EXAMPLE` only includes a small subset of the configuration settings available with DSpace. It provides a good starting point for your own `local.cfg` file.

However, you should be aware that ANY configuration can now be copied into your `local.cfg` to override the default settings. This includes ANY of the settings/configurations in:

- The primary dspace.cfg file (`[dspace]/config/dspace.cfg`)
- Any of the module configuration files (`[dspace]/config/modules/*.cfg` files)
- Any of the Spring Boot settings (`[dspace-src]/dspace-server-webapp/src/main/resources/application.properties`)

Individual settings may also be commented out or removed in your `local.cfg`, in order to re-enable default settings.

See the Configuration Reference section for more details.

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

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

(Assuming the *dspace* UNIX username.)

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

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

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

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

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

10. **Initialize your Database:** While this step is optional (as the DSpace database should auto-initialize itself on first startup), it's always good to verify one last time that your database connection is working properly. To initialize the database run:

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

    a. After running this script, it's a good idea to run "./dspace database info" to check that your database has been fully initialized. A fully initialized database should list the state of all migrations as either "Success" or "Out of Order". If any migrations have failed or are still listed as "Pending", then you need to check your "dspace.log" for possible "ERROR" messages. If any errors appeared, you will need to resolve them before continuing.

11. **Copy Solr cores:** DSpace installation creates a set of six empty Solr cores already configured.

    a. Copy them from `[dspace]`/solr to the place where your Solr instance will discover them. For example:

```
# [solr] is the location where Solr is installed.
# NOTE: On Debian systems the configsets may be under /var/solr/data/configsets
cp -R [dspace]/solr/* [solr]/server/solr/configsets

# Make sure everything is owned by the system user who owns Solr
# Usually this is a 'solr' user account
# See https://solr.apache.org/guide/solr/9_8/deployment-guide/taking-solr-to-production.
html#create-the-solr-user
chown -R solr:solr [solr]/server/solr/configsets
```

    b. Start (or re-start) Solr. For example:

```
[solr]/bin/solr restart
```

    c. You can check the status of Solr and your new DSpace cores by using its administrative web interface. Browse to `${solr.server}` (e.g. `http://localhost:8983/solr/`) to see if Solr is running well, then look at the cores by selecting (on the left) Core Admin or using the Core Selector drop list.

i. For example, to test that your "search" core is setup properly, try accessing the URL `${solr.server}/search/select`. It should run an empty query against the "search" core, returning an empty JSON result. If it returns an error, then that means your "search" core is missing or not installed properly.

12. **Deploy web application**
We have different possibilities in this case:
    - **Deploy web application to Tomcat (traditional installation):** The DSpace backend consists of a single "server" webapp (in `[dspace]/webapps/server`). You need to deploy this webapp into your Servlet Container (e.g. Tomcat). Generally, there are two options (or techniques) which you could use...either configure Tomcat to find the DSpace "server" webapp, or copy the "server" webapp into Tomcat's own webapps folder.
        - *Technique A.* Tell your Tomcat/Jetty/Resin installation where to find your DSpace web application(s). As an example, in the directory `[tomcat]/conf/Catalina/localhost` you could add files similar to the following (but replace `[dspace]`with your installation location):

          DEFINE A CONTEXT PATH FOR DSpace Server webapp: server.xml

          ```
          <?xml version='1.0'?>
          <Context
                  docBase="[dspace]/webapps/server"/>
          ```

          The name of the file (not including the suffix ".xml") will be the name of the context, so for example `server.xml` defines the context at `http://host:8080/server`. To define the *root context* (`http://host:8080/`), name that context's file `ROOT.xml`. Optionally, you can also choose to install the old, deprecated "rest" webapp if you
        - *Technique B.* Simple and complete. You copy only (or all) of the DSpace Web application(s) you wish to use from the [dspace]/webapps directory to the appropriate directory in your Tomcat/Jetty/Resin installation. For example:
          `cp -R [dspace]/webapps/* [tomcat]/webapps` (This will copy all the web applications to Tomcat).
          `cp -R [dspace]/webapps/server [tomcat]/webapps` (This will copy only the Server web application to Tomcat.)

          To define the *root context* (`http://host:8080/`), name that context's directory `ROOT`.
    - **Deploy Runnable JAR application (NEW) :** The DSpace backend now builds a Runnable JAR application made with `SpringBoot`. After building DSpace, a new "server-boot.jar" will be available at `[dspace]/webapps/server-boot.jar`. This JAR file contains the entire "server" webapp, embedded Tomcat, and the "dspace.dir" configuration made during the build phase. You can execute this JAR with the following command:

      Server-boot execution

      ```
      java -jar [dspace]/webapps/server-boot.jar
      ```

      By running it, the server will boot with the configuration that you've made during the build phase. There are optional parameters that you can use to override the build values:

        - `spring.config.location` - reference to the application.properties file to use

          ```
          --spring.config.location=file:///path/to/target/application.properties
          ```

        - `dspace.dir` - reference to the installation directory of the application, (default value in `application.properties`)

          ```
          --dspace.dir=/path/to/install/folder
          ```

        - `logging.config` - log configuration file of the project (default value in `application.properties`)

          ```
          --logging.config=file:///path/to/target/file/log2.xml
          ```

        - These are only the main ones, obviously , you can override every property that can be found inside the configuration files just by appending it as argument of the execution command, just like this: `--[prop]=[value]`. Or you may choose to use Environment Variable overriding as described in the Configuration Reference

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

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

14. **Initial Startup!** Now the moment of truth! Start up (or restart) Tomcat/Jetty/Resin.
    a. *REST API Interface* - (e.g.) http://dspace.example.edu:8080/server/
    b. *OAI-PMH Interface* - (e.g.) http://dspace.example.edu:8080/server/oai/request?verb=Identify
    c. For an example of what the default backend looks like, visit the Demo Backend: https://demo.dspace.org/server/

15. **Setup scheduled tasks for behind-the-scenes processes:** For all features of DSpace to work properly, there are some scheduled tasks you MUST setup to run on a regular basis. Some examples are tasks that help create thumbnails (for images), do full-text indexing (of textual content) and send out subscription emails. See the Scheduled Tasks via Cron for more details.
16. **Production Installation (adding HTTPS support)**: *Running the DSpace Backend on HTTP & port 8080 is only usable for local development environments (where you are running the UI and REST API from the same machine, and only accessing them via localhost URLs).* **If you want to run DSpace in Production, you MUST run the backend with HTTPS support** (otherwise logins will not work outside of your local domain).
    a. For HTTPS support, we recommend installing either Apache HTTPD or Nginx, configuring SSL at that level, and proxying all requests to your Tomcat installation (or Runnable JAR). Keep in mind, if you want to host both the DSpace Backend and Frontend on the same server, you can use one installation of Apache HTTPD or NGinx to manage HTTPS/SSL and proxy to both.
    b. *Apache HTTPD:* These instructions are specific to Apache HTTPD, but a similar setup can be achieved with NGinx (see below)
        i. Install Apache HTTPD, e.g. `sudo apt install apache2`
        ii. Install mod_headers, mod_proxy and mod_proxy_ajp (or mod_proxy_http) modules, e.g. `sudo a2enmod headers; sudo a2enmod proxy; sudo a2enmod proxy_ajp`
            1. Alternatively, you can choose to use mod_proxy_http to create an http proxy. A separate example is commented out below
        iii. For mod_proxy_ajp to communicate with Tomcat, you'll need to enable Tomcat's AJP connector in your Tomcat's server.xml:

```
<Connector protocol="AJP/1.3" port="8009" redirectPort="8443" URIEncoding="UTF-8" />
```

        iv. Restart Apache to enable these modules
        v. Obtain an SSL certificate for HTTPS support. If you don't have one yet, you can use Let's Encrypt (for free) using the "certbot" tool: https://certbot.eff.org/
        vi. Now, setup a new VirtualHost for your site (using HTTPS / port 443) which proxies all requests to Tomcat's AJP connector (running on port 8009)

```
<VirtualHost _default_:443>
    # Add your domain here. We've added "dspace.example.edu" as an example
    ServerName dspace.example.edu
    .. setup your host how you want, including log settings...     .. setup your host how
you want, including log settings...

    # Most installs will need these options enabled to ensure DSpace knows its hostname and
scheme (http or https)
    # Also required to ensure correct sitemap URLs appear in /robots.txt for User Interface.
    ProxyPreserveHost On
    RequestHeader set X-Forwarded-Proto https

    SSLEngine on
    SSLCertificateFile [full-path-to-PEM-cert]
    SSLCertificateKeyFile [full-path-to-cert-KEY]
    # LetsEncrypt certificates (and possibly others) may require a chain file be specified
    # in order for the UI / Node.js to validate the HTTPS connection.
    #SSLCertificateChainFile [full-path-to-chain-file]

    # Proxy all HTTPS requests to "/server" from Apache to Tomcat via AJP connector
    ProxyPass /server ajp://localhost:8009/server
    ProxyPassReverse /server ajp://localhost:8009/server

    # If you would rather use mod_proxy_http as an http proxy to port 8080
    # then use these settings instead
    #ProxyPass /server http://localhost:8080/server
    #ProxyPassReverse /server http://localhost:8080/server
</VirtualHost>
```

    c. *NGinx:* These instructions are specific to NGinx.
        i. Install/Setup NGinx
        ii. Sample NGinx "server block" configuration. Keep in mind we are only providing basic example settings.

```
# Setup HTTP to redirect to HTTPS
server {
  listen 80;
  # Add your domain here. We've added "dspace.example.edu" as an example
  server_name dspace.example.edu;
  rewrite ^ https://dspace.example.edu permanent;
}

# Setup HTTPS access
server {
  listen 443 ssl;
  # Add your domain here. We've added "dspace.example.edu" as an example
```

```
  server_name dspace.example.edu;

  # Add your SSL certificate/key path here
  # NOTE: For LetsEncrypt, the certificate should be the full certificate chain file
  ssl_certificate dspace.example.edu.crt (or PEM);
  ssl_certificate_key dspace.example.edu.key;

  # Proxy all HTTPS requests to "/server" from NGinx to Tomcat on port 8080
  location /server {
    proxy_set_header X-Forwarded-Proto https;
    proxy_set_header X-Forwarded-Host $host;
    proxy_pass http://localhost:8080/server;
  }
}
```

d. After switching to HTTPS, make sure to go back and update the URLs (primarily `dspace.server.url`) in your local.cfg to match the new URL of your backend (REST API). This will require briefly rebooting Tomcat.

Installing the Frontend (User Interface)

Frontend Requirements

- UNIX-like OS or Microsoft Windows
- Node.js (v20.19+, v22.x or v24.x)
- PM2 (or another Process Manager for Node.js apps) (optional, but recommended for Production)
- DSpace Backend (see above)

UNIX-like OS or Microsoft Windows

- UNIX-like operating system (Linux, HP/UX, Mac OSX, etc.) : Many distributions of Linux/Unix come with some of the dependencies below pre-installed or easily installed via updates. You should consult your particular distribution's documentation or local system administrators to determine what is already available.
- Microsoft Windows: While DSpace can be run on Windows servers, most institutions tend to run it on a UNIX-like operating system.

Node.js (v20.19+, v22.x or v24.x)

DSpace uses Angular 20 which requires Node v20.19.0 or later.

- Node.js can be found at https://nodejs.org/. It may be available through your Linux distribution's package manager. We recommend running a Long Term Support (LTS) version (even numbered releases). Non-LTS versions (odd numbered releases) are not recommended.
- Node.js is a Javascript runtime that also provides npm (Node Package Manager). It is used to both build and run the frontend.

PM2 (or another Process Manager for Node.js apps) (*optional, but recommended for Production*)

- In Production scenarios, we *highly recommend* starting/stopping the User Interface using a Node.js process manager. There are several available, but our current favorite is PM2. The rest of this installation guide assumes you are using PM2.
- PM2 is very easily installed via NPM

```
# You may need to run this command using "sudo" if you don't have proper privileges
npm install --global pm2
```

DSpace Backend (see above)

- The DSpace User Interface (Frontend) cannot function without an installed DSpace Backend. Follow the instructions above.
- The Frontend and Backend *do not need to be installed on the same machine/server*. They may be installed on separate machines as long as the two machines can connect to one another via HTTP or HTTPS.

Frontend Installation

1. **Download Code (to [dspace-angular]):** Download the latest dspace-angular release from the DSpace GitHub repository. You can choose to either download the zip or tar.gz file provided by GitHub, or you can use "git" to checkout the appropriate tag (e.g. `dspace-9.0`) or branch.
    a. NOTE: For the rest of these instructions, we'll refer to the source code location as `[dspace-angular].`
2. **Install Dependencies:** Install all required local dependencies by running the following from within the unzipped `[dspace-angular]` directory

```
# change directory to our repo
cd [dspace-angular]

# install the local dependencies
npm install
```

3. **Build/Compile:** Build the User Interface for Production. This builds source code (under `[dspace-angular]/src/`) to create a compiled version of the User Interface in the `[dspace-angular]/dist` folder. This `/dist` folder is what we will deploy & run to start the UI.

```
npm run build:prod
```

   a. You only need to rebuild the UI application if you change source code (under `[dspace-angular]/src/`). Simply changing the configurations (e.g. config.prod.yml, see below) do not require a rebuild, but only require restarting the UI.
4. **Deployment (to [dspace-ui-deploy]):** *(Only recommended for Production setups)* Choose/Create a directory on your server where you wish to run the compiled User Interface. We'll call this `[dspace-ui-deploy]`.

[dspace-ui-deploy] vs [dspace-angular]

`[dspace-angular]` is the directory where you've downloaded and built the UI source code (per the instructions above). For deployment /running the UI, we recommend creating an entirely separate `[dspace-ui-deploy]` directory. This keeps your running, production User Interface separate from your source code directory and also minimizes downtime when rebuilding your UI. You may even choose to deploy to a `[dspace-ui-deploy]` directory on a different server (and copy the `/dist` directory over via FTP or similar).

If you are installing the UI for the first time, or just want a simple setup, you can choose to have [dspace-ui-deploy] and [dspace-angular] be the *same directory*. This would mean you don't have to copy your /dist folder to another location. However, the downside is that your running site will become unresponsive whenever you do a re-build/re-compile (i.e. rerun "npm run build:prod") as this build process will first delete the `[dspace-angular]/dist` directory before rebuilding it.

   a. Copy the entire `[dspace-angular]/dist/` folder to this location. For example:

```
cp -r [dspace-angular]/dist [dspace-ui-deploy]
```

   b. WARNING: At this time, you MUST copy the entire "dist" folder and make sure NOT to rename it. Therefore, the directory structure should look like this:

Contents of [dspace-ui-deploy] folder

```
[dspace-ui-deploy]
    /dist
       /browser (compiled client-side code)
       /server  (compiled server-side code, including "main.js")
    /config     (Optionally created in the "Configuration" step below)
       /config.prod.yml (Optionally created in the "Configuration" step below)
```

   c. NOTE: the OS account which runs the UI via Node.js (see below) MUST have write privileges to the `[dspace-ui-deploy]` directory (because on startup, the runtime configuration is written to `[dspace-ui-deploy]/dist/browser/assets/config.json`)
5. **Configuration:** You have two options for User Interface Configuration, Environment Variables or YAML-based configuration (`config.prod.yml`). Choose one!

   a. *YAML configuration:* Create a "config.prod.yml" at `[dspace-ui-deploy]/config/config.prod.yml`. You may wish to use the `[dspace-angular]/config/config.example.yml` as a starting point. This `config.prod.yml` file can be used to override any of the default configurations listed in the `config.example.yml` (in that same directory). *At a minimum* this file MUST include a "rest" section (and may also include a "ui" section), similar to the following (keep in mind, you only need to include settings that you need to modify).

Example config.prod.yml

```
# The "ui" section defines where you want Node.js to run/respond. It often is a *localhost* (non-public) URL, especially if you are using a Proxy.
# However, the "baseUrl" setting will define the public URL of your DSpace instance. It should be set to the same value as "dspace.ui.url" on the backend.
# In this example, we are setting up our UI to just use localhost, port 4000.  But, the public URL is "https://dspace.example.edu"
# This is a common setup for when you want to use Apache or Nginx to handle HTTPS and proxy requests to Node on port 4000
ui:
  ssl: false
  host: localhost
  port: 4000
  nameSpace: /
  # This is the public URL where the site will respond (often defined in your Apache/Nginx proxy)
  # It MUST be kept in sync with the value of "dspace.ui.url" in the backend's local.cfg
  baseUrl: https://dspace.example.edu
```

```
# This example is valid if your Backend is publicly available at https://dspace.example.edu/server/
# The REST settings MUST correspond to the primary/public URL of the backend. Usually, this means
they must be kept in sync
# with the value of "dspace.server.url" in the backend's local.cfg
rest:
  ssl: true
  host: dspace.example.edu
  port: 443
  nameSpace: /server
```

b. *Environment variables:* Every configuration in the UI may be specified via an Environment Variable. See Configuration Override in the User Interface Configuration documentation for more details. For example, the below environment variables provide the same setup as the config.prod.yml example above.

Example Environment Variables

```
# All environment variables MUST
# (1) be prefixed with "DSPACE_"
# (2) use underscores as separators (no dots allowed), and
# (3) use all uppercase

# "ui" section
DSPACE_UI_SSL = false
DSPACE_UI_HOST = localhost
DSPACE_UI_PORT = 4000
DSPACE_UI_NAMESPACE = /
DSPACE_UI_BASEURL = https://dspace.example.edu

# "rest" section
DSPACE_REST_SSL = true
DSPACE_REST_HOST = dspace.example.edu
DSPACE_REST_PORT = 443
DSPACE_REST_NAMESPACE = /server
```

   i. NOTE: When using PM2, some may find it easier to use Environment variables, as it allows you to specify DSpace UI configs within your PM2 configuration. See PM2 instructions below.

c. Configuration Hints:
   i. See the User Interface Configuration documentation for a list of all available configurations.
   ii. In the "ui" section above, you may wish to start with "ssl: false" and "port: 4000" just to be certain that everything else is working properly **before** adding HTTPS support. KEEP IN MIND, we highly recommend always using HTTPS for Production. (See section on HTTPS below)
   iii. (Optionally) *Test the connection to your REST API from the UI from the command-line.* This is not required, but it can sometimes help you discover immediate configuration issues if the test fails.
      1. If you are using YAML configs, copy your config.prod.yml back into your source code folder at `[dspace-angular]/config/config.prod.yml`
      2. From `[dspace-angular]`, run `npm run test:rest` This script will attempt a basic Node.js connection to the REST API that is configured in your "config.prod.yml" file and validate the response.
      3. A successful connection should return a 200 Response and all JSON validation checks should return "true"
      4. If you receive a connection error or different response code, you MUST fix your REST API before the UI will be able to work. See also the "Common Installation Issues" below. If you receive an SSL error, see "Using a Self-Signed SSL Certificate causes the Frontend to not be able to access the Backend"
   iv. When using a subpath (nameSpace) in your UI server base URL (e.g. "http://localhost:4000/mysite/" instead of "http://localhost:4000/"), you must make sure that the URL **without** the subpath is added to the `rest.cors.allowed-origins` list in `[dspace]/config/modules/rest.cfg` or the `local.cfg` override. The default value used for this configuration assumes that Origin and DSpace URL are identical, but CORS origins do not contain a subpath.

6. **Start up the User Interface:** The compiled User Interface only requires Node.js to run. However, most users may want to use PM2 (or a similar Node.js process manager) in Production to provide easier logging and restart tools.
   a. *Quick Start:* To quickly startup / test the User Interface, you can just use Node.js. This is only recommended for quickly testing the UI is working, as no logs are available.

```
# You MUST start the UI from within the deployment directory
cd [dspace-ui-deploy]

# Run the "server/main.js" file to startup the User Interface
node ./dist/server/main.js

# Stop the UI by killing it via Ctrl+C
```

b. *Run via PM2:* Using PM2 (or a different Node.js process manager) is highly recommended for Production scenarios. Here's an example of a Production setup of PM2.
    i. First you need to create a PM2 JSON configuration file which will run the User Interface. This file can be named anything & placed where ever you like, but you may want to save it to your deployment directory (e.g. `[dspace-ui-deploy]/dspace-ui.json`).

dspace-ui.json

```
{
    "apps": [
        {
           "name": "dspace-ui",
           "cwd": "/full/path/to/dspace-ui-deploy",
           "script": "dist/server/main.js",
           "instances": "max",
           "exec_mode": "cluster",
           "env": {
              "NODE_ENV": "production"
           }
        }
    ]
}
```

1. NOTE: The "cwd" setting MUST correspond to your `[dspace-ui-deploy]` folder path.
2. NOTE #2: The "exec_mode" and "instances" settings are optional but highly recommended. Setting "exec_mode" to "cluster" enable's PM2's cluster mode. This will provide better performance in production as it allows PM2 to scale your site across multiple CPUs. The "instances" setting tells PM2 how many CPUs to scale across ("max" means all CPUs, but you can also specify a number.)
3. NOTE #3: If you wanted to configure your UI using Environment Variables, specify those Environment Variables under the "env" section. For example:

Configuration via Environment Variables

```
"env": {
   "NODE_ENV": "production",
   "DSPACE_REST_SSL": "true",
   "DSPACE_REST_HOST": "demo.dspace.org",
   "DSPACE_REST_PORT": "443",
   "DSPACE_REST_NAMESPACE": "/server"
}
```

4. NOTE #4: If you are using Windows, there are two other rules to keep in mind in this JSON configuration. First, *all paths must include double backslashes* (e.g. "C:\\dspace-ui-deploy"). Second, "cluster" mode is *required*. Here's an example configuration for Windows:

dspace-ui.json (for Windows)

```
{
    "apps": [
        {
           "name": "dspace-ui",
           "cwd": "C:\\full\\path\\to\\dspace-ui-deploy",
           "script": "dist\\server\\main.js",
           "instances": "max",
           "exec_mode": "cluster",
           "env": {
              "NODE_ENV": "production"
           }
        }
    ]
}
```

    ii. Now, start the application using PM2 using the configuration file you created in the previous step

```
# In this example, we are assuming the config is named "dspace-ui.json"
pm2 start dspace-ui.json
```

```
# To see the logs, you'd run
# pm2 logs

# To stop it, you'd run
# pm2 stop dspace-ui.json

# If you need to change your PM2 configs, delete the old config and restart
# pm2 delete dspace-ui.json
```

iii. For more PM2 commands see https://pm2.keymetrics.io/docs/usage/quick-start/
iv. HINT: You may also want to install/configure pm2-logrotate to ensure that PM2's log folder doesn't fill up over time.
v. *Did* PM2 *not work or throw an immediate error?* It's likely that something in your UI installation or configuration is incorrect. Check the PM2 logs ("pm2 logs") first for errors. If the problem is not obvious, try to see if you can run the UI using the "Quick Start" method (using just Node.js) instead. Once "Quick Start" is working, try PM2 again.
vi. *If neither PM2 nor the "Quick Start" method works for you:* then see the "User Interface never appears (no content appears)" section in the Commons Installation Issues below

7. **Test it out:** At this point, the User Interface should be available at the URL you configured!
   a. For an example of what the default frontend looks like, visit the Demo Frontend: https://demo.dspace.org/
   b. If the UI fails to start or throws errors, it's likely a configuration issue. See Commons Installation Issues below for common error messages you may see and how to resolve them.
   c. If you have an especially difficult issue to debug, you may wish to *stop* PM2. Instead, try running the UI via the "Quick Start" method (using just Node.js). This command might provide a more specific error message to you, if PM2 is not giving enough information back.
8. **Add HTTPS support:** For HTTPS (port 443) support, you have two options
   a. *(Recommended)* Install either Apache HTTPD or Nginx to act as a "reverse proxy" for the frontend (and backend). This allows you to manage HTTPS (SSL certificates) in either Apache HTTPD or Nginx, and proxy all requests to the frontend (running on port 4000) and backend (running on port 8080). This is our current recommended approach. These instructions are specific to Apache, but a similar setup can be achieved with Nginx.
      i. If you already have Apache / Nginx installed for the backend, you can use the same Apache / Nginx. You can also choose to install a separate one (either approach is fine).
         1. Install Apache HTTPD, e.g. `sudo apt install apache2`
         2. Install the mod_proxy and mod_proxy_http modules, e.g. `sudo a2enmod proxy; sudo a2enmod proxy_http`
         3. Restart Apache to enable
         4. Obtain an SSL certificate for HTTPS support. If you don't have one yet, you can use Let's Encrypt (for free) using the "certbot" tool: https://certbot.eff.org/
      ii. *Apache HTTPD sample configuration:*

         1. Now, setup (or update) the new VirtualHost for your UI site (preferably using HTTPS / port 443) which proxies all requests to PM2 running on port 4000.

```
<VirtualHost _default_:443>
    # Add your domain here. We've added "dspace.example.edu" as an example
    ServerName dspace.example.edu
    .. setup your host how you want, including log settings...

    # Most installs will need these options enabled to ensure DSpace knows its
hostname and scheme (http or https)
    # Also required to ensure correct sitemap URLs appear in /robots.txt for User
Interface.
    ProxyPreserveHost On
    RequestHeader set X-Forwarded-Proto https

    # These SSL settings are identical to those for the backend installation (see
above)
    # If you already have the backend running HTTPS, just add the new Proxy settings
below.
    SSLEngine on
    SSLCertificateFile [full-path-to-PEM-cert]
    SSLCertificateKeyFile [full-path-to-cert-KEY]
    # LetsEncrypt certificates (and possibly others) may require a chain file be
specified
    # in order for the UI / Node.js to validate the HTTPS connection.
    #SSLCertificateChainFile [full-path-to-chain-file]

    # These Proxy settings are for the backend. They are described in the backend
installation (see above)
    # If you already have the backend running HTTPS, just append the new Proxy
settings below.
    # Proxy all HTTPS requests to "/server" from Apache to Tomcat via AJP connector
    # (In this example: https://dspace.example.edu/server/ will display the REST API)
    ProxyPass /server ajp://localhost:8009/server
    ProxyPassReverse /server ajp://localhost:8009/server
```

```
    # [NEW FOR UI:] Proxy all HTTPS requests from Apache to PM2 on localhost, port
4000
    # NOTE that this proxy URL must match the "ui" settings in your config.prod.yml
    # (In this example: https://dspace.example.edu/ will display the User Interface)
    ProxyPass / http://localhost:4000/
    ProxyPassReverse / http://localhost:4000/
</VirtualHost>
```

iii. *NGinx sample configuration*
   1. Sample NGinx "server block" configuration. Keep in mind we are only providing basic example settings.

```
# Setup HTTPS access
server {
  listen 443 ssl;
  # Add your domain here. We've added "dspace.example.edu" as an example
  server_name dspace.example.edu;

  # Add your SSL certificate/key path here
  # NOTE: For LetsEncrypt, the certificate should be the full certificate chain file
  # These SSL settings are identical to those for the backend installation (see
above)
  # If you already have the backend running HTTPS, just add the new Proxy settings
below.
  ssl_certificate dspace.example.edu.crt (or PEM);
  ssl_certificate_key dspace.example.edu.key;

  # Proxy all HTTPS requests to "/server" from NGinx to Tomcat on port 8080
  # These Proxy settings are for the backend. They are described in the backend
installation (see above)
  location /server {
    proxy_set_header X-Forwarded-Proto https;
    proxy_set_header X-Forwarded-Host $host;
    proxy_pass http://localhost:8080/server;
  }

  # [NEW FOR UI:] Proxy all HTTPS requests from NGinx to PM2 on localhost, port 4000
  # NOTE that this proxy URL must match the "ui" settings in your config.prod.yml
  # (In this example: https://dspace.example.edu/ will display the User Interface)
  location / {
    proxy_set_header X-Forwarded-Proto https;
    proxy_set_header X-Forwarded-Host $host;
    proxy_pass http://localhost:4000/;
  }
}
```

iv. HINT#1: Because you are using a proxy for HTTPS support, in your User Interface Configuration, your "ui" settings will still have "ssl: false" and "port: 4000". This is perfectly OK!
v. HINT#2: to force the UI to connect to the backend using HTTPS, you should verify your "rest" settings in your User Interface Configuration **match** the "dspace.server.url" in your backend's "local.cfg" and both use the HTTPS URL. So, if your backend (REST API) is proxied to https://dspace.example.edu/server/, both those settings should specify that HTTPS URL.
vi. HINT#3: to force the backend to recognize the HTTPS UI, make sure to update your "dspace.ui.url" in your backend's "local.cfg" is updated to use the new HTTPS UI URL (e.g. https://dspace.example.edu).

b. *(Alternatively)* You can use the basic HTTPS support built into our UI and Node server. (This may currently be better for non-Production environments as it has not been well tested)
   i. Create a `[dspace-ui-deploy]/config/ssl/` folder and add a `key.pem` and `cert.pem` to that folder (they must have those exact names)
   ii. In your User Interface Configuration, go back and update the following:
      1. Set "ui > ssl" to true
      2. Update "ui > port" to be 443
         a. In order to run Node/PM2 on port 443, you also will likely need to provide node with special permissions, like in this example.
   iii. Restart the UI
   iv. Keep in mind, while this setup is simple, you may not have the same level of detailed, Production logs as you would with Apache HTTPD or Nginx

What Next?

After a successful installation, you may want to take a closer look at

- Performance Tuning DSpace: If you are noticing any slowness in your Production site, we have a guide for how you might speed things up.

- User Interface Customization: Documentation on customizing the User Interface with your own branding / theme(s)
- User Interface Configuration: Additional configurations available in the User Interface.
- Submission User Interface: Options to configure/customize the default Submission (deposit) process
- Configurable Workflow: Options to configure/customize the default Workflow approval process
- Scheduled Tasks via Cron : Several DSpace features **require** that a command-line script is run regularly via cron.
- Configuration Reference : Details on the configuration options available to the Backend
- Handle Server installation: Optionally, you may wish to enable persistent URLs for your DSpace site using CRNI's Handle.Net Registry
- Statistics and Metrics: Optionally, you may wish to configuration one (or more) Statistics options within DSpace, including Google Analytics and (internal) Solr Statistics
- Multilingual Support: Optionally, you may wish to enable multilingual support in your DSpace site.
- Using DSpace : Various other pages which describe usage and additional configurations related to other DSpace features.
- System Administration: Various other pages which describe additional backend installation options/configurations.

If you've run into installation problems, you may want to...

- Visit the Troubleshoot an error guide for tips on locating the cause of the error
- Review Commons Installation Issues (see below)
- Ask for Support via one of the support options documented on that page

Common Installation Issues

Troubleshoot an error or find detailed error messages

See the Troubleshoot an error guide, look for the section on "DSpace 7.x or 8.x". This will provide you hints on locating error messages both in the User Interface (frontend) and in the REST API (backend)

User Interface never appears (no content appears) or "Proxy server received an invalid response"

Chances are your User Interface (UI) is throwing a severe error or not starting properly. The best way to debug this issue would be to start the User Interface in development mode to see if it can give you a more descriptive error.

1. First, create a `[dspace-ui-deploy]/config/config.dev.yml` configuration file for development. This file supports the same configs as your existing config.prod.yml. So, you can copy over any settings you want to test out.
2. Start the UI in development mode (this doesn't require a proxy like Apache or Nginx)

   ```
   npm run start:dev
   ```

3. This will boot up the User Interface on whatever port you specified in "config.dev.yml"
4. At this point, attempt to access the UI from your web browser. Even if it isn't fully working, you should be able to still get more information from your browser's DevTools regarding the underlying error. See the Troubleshoot an error page, look for the section on "DSpace 7.x or 8.x". It has a guide for locating UI error messages in your browser's Developer Tools.

Once you've found the underlying error, it may be one of the "common installation issues" listed below.

User Interface partially load but then spins (never fully loads or some content doesn't load)

Chances are your User Interface (UI) is throwing an error or receiving an unexpected response from the REST API backend. Since the UI is Javascript based, it runs entirely in your browser. That means the error it's hitting is most easily viewed in your browser (and in fact the error may never appear in log files).

See the Troubleshoot an error page, look for the section on "DSpace 7.x or 8.x". It has a guide for locating UI error messages in your browser's Developer Tools.

"500 Service Unavailable" from the User Interface

This error is saying that the frontend is working, but it is unable to communicate with your backend. It's the same as the "No _links section found at..." error described in the next section. Please follow the troubleshooting details in that section.

"No _links section found at..." error from User Interface

When starting up the User Interface for the first time, you may see an error that looks similar to this...

```
No _links section found at [rest-api-url]
ERROR Error: undefined doesn't contain the link sites
     at MapSubscriber.project
```

This error means that the UI is unable to contact the REST API listed at `[rest-api-url]` and/or the response from that `[rest-api-url]` is unexpected (as it doesn't contain the "_links" to the endpoints available at that REST API). A valid DSpace `[rest-api-url]` will respond with JSON similar to our demo API at https://demo.dspace.org/server/api

First, *test the connection to your REST API from the UI from the command-line.*

```
# This script will attempt a basic Node.js connection to the REST API
# configured in your "[dspace-angular]/config/config.prod.yml" and
# validate the response.(NOTE: config.prod.yml MUST be copied to
# to [dspace-angular]/config/ for this script to find it!)
npm run test:rest
```

- A successful connection should return a 200 Response and all JSON validation checks should return "true".
- If you receive a connection error or different response code, you MUST fix your REST API before the UI will be able to work (see additional hints below for likely causes).

Usually, the core problem is caused by one of the following scenarios:

- *A possible configuration issue* in the frontend or backend.
  - Check the "rest" section of your `config.*.yml` configuration file for the User Interface. That configuration section defines which REST API the UI will attempt to use. If the settings do NOT map to a valid DSpace REST API, then you will see this "No _links section found.." error. Keep in mind, **the REST API must use HTTPS** (the only exception is if both the frontend and backend are running on "localhost"-based URLs)
  - Check the "dspace.ui.url" configuration of your backend & verify it corresponds to the public URL of the User Interface (i.e. the exact same URL you use in your browser)
  - Verify the backend "trusts" the frontend via the "rest.cors.allowed-origins" configuration (in rest.cfg or local.cfg). This setting must list all web-based clients which are trusted by the backend (REST API). By default, "dspace.ui.url" should be listed... but you should verify it has not been modified.
- *A possible SSL certificate issue*. This issue may also appear if the REST API's SSL Certificate is either *untrusted (by the frontend) or expired*.
  - If you are using a Let's Encrypt style certificate, you may need to modify your backend's Apache settings to also provide a Chain File as follows:

    ```
    # For example: /etc/letsencrypt/live/[domain]/cert.pem
    SSLCertificateFile [full-path-to-PEM-cert]
    # For example: /etc/letsencrypt/live/[domain]/privkey.pem
    SSLCertificateKeyFile [full-path-to-cert-KEY]
    # For example: /etc/letsencrypt/live/[domain]/chain.pem
    SSLCertificateChainFile [full-path-to-chain-file]
    ```

  - Per the Apache docs, you can also use the SSLCertificateFile setting to specify intermediate CA certificates along with the main cert.
  - For self-signed certs, see also "Using a Self-Signed SSL Certificate causes the Frontend to not be able to access the Backend" common issue listed below.
- *Something blocking access to the REST API.* This may be a proxy issue, a firewall issue, or something else generally blocking the port (e.g. port 443 for SSL).
  - Verify that you can access the REST API from the machine where Node.js is running (i.e. your UI is running). For example try a simple "wget" or "curl" to verify the REST API is returning expected JSON similar to our demo API at https://demo.dspace.org/server/api

    ```
    # Attempt to access the REST API via HTTPS from command-line on the machine where Node.js is
    running.
    # If this fails or throws a SSL cert error, you must fix it.
    wget https://[rest.host]/server/api
    ```

  - **In most production scenarios, your REST API should be publicly accessible on the web**, unless you are guaranteed that all your DSpace users will access the site behind a VPN or similar. So, this "No _links section found" error may also occur if you are accessing the UI from a client computer/web browser which is *unable to access the REST API.*

If none of the above suggestions helped, you may want to look closer at the request logs in your browser (using browser's Dev Tools) and server-side logs, to be sure that the requests from your UI are going where you expect, and see if they appear also on the backend. Tips for finding these logs can be found in the "DSpace 7.x or 8.x" section of our Troubleshoot an error guide.

"RangeError: Maximum call stack size exceeded"

When starting up the User Interface for the first time, you may see an error that looks similar to this...

```
ERROR RangeError: Maximum call stack size exceeded
```

This error means that the UI is trying to contact your REST API, but is having issues doing so (possibly because either a proxy or an HTTPHTTPS redirect is causing issues or a redirect loop).

Double check your "`dspace.server.url`" setting in your local.cfg on the backend. Is it the same URL you use in your browser to access the backend? Keep in mind the mode (http vs https), domain, port, and subpath(s) all must match, and it *must not end in a trailing slash.*

Also double check the "rest" section of your `config.*.yml` configuration file for the User Interface. Make sure it's also pointing to the exact same URL as that "`dspace.server.url`" setting. Again, check the mode, domain, port and paths all match exactly.

"XMLHttpRequest.. has been blocked by CORS policy" or "CORS error" or "Invalid CORS request"

If you are seeing a CORS error in your browser, this means that you are accessing the REST API via an "untrusted" client application. To fix this error, you must change your REST API / Backend configuration to trust the application.

- By default, the DSpace REST API / Backend will only trust the application at `dspace.ui.url`. Therefore, you should first verify that your `dspace.ui.url` setting (in your local.cfg) exactly matches the *primary URL* of your User Interface (i.e. the URL you see in the browser). This must be an exact match: mode (http vs https), domain, port, and subpath(s) all must match.
- If you need to trust *additional* client applications / URLs, those MUST be added to the `rest.cors.allowed-origins` configuration. See REST API for details on this configuration.
- Also, check your Tomcat (or servlet container) log files. If Tomcat throws a syntax or other major error, it may return an error response that triggers a CORS error. In this scenario, the CORS error is only a side effect of a larger error.

If you modify either of the above settings, you will need to restart Tomcat for the changes to take effect.

Cannot login from the User Interface with a password that I know is valid

If you cannot login via the user interface with a valid password, you should check to see what underlying error is being returned by the REST API. The easiest way to do this is by using your web browser's Dev Tools as described in our Troubleshoot an error guide (see the "Try this first" section for DSpace 7).

If the password is valid, more than likely you'll see the underlying error is "403 Forbidden" error with a message that says "Access is denied. Invalid CSRF Token" (see hints on solving this in the very next section)

"403 Forbidden" error with a message that says "Access is denied. Invalid CSRF Token"

First, double check that you are seeing that exact error message. A `403 Forbidden` error may be thrown in a variety of scenarios. For example, a 403 may be thrown if a page requires a login, if you have entered an invalid username or password, or even sometimes when there is a CORS error (see previous installation issue for how to solve that).

If you are seeing the message "Invalid CSRF Token" message (especially on every login), this is usually the result of a configuration / setup issue.

Here's some things you should double check:

1. If you site had been working, and this error seems random, it is possibly that `DSPACE-XSRF-COOKIE` cookie in your browser just got "out of sync" (this can occur if you are logging into the REST API and UI separately in the same browser).
    a. Logout and login & try the same action again. If it works this time, then that cookie was just "out of sync". If it fails a second time, then there is a likely configuration issue...see suggestions below.
2. **Make sure your backend is running HTTPS**! This is the most common cause of this error. The only scenario where you can run the backend in HTTP is when both the frontend & backend URLs are "localhost"-based URLs.
    a. The reason for this HTTPS requirement is that most modern browsers will automatically block cross-domain cookies when using HTTP. Cross-domain cookies are *required* for successful authentication. The only *exception* is when both the frontend and backend are using localhost URLs (as in that scenario the cookies no longer need to be sent cross-domain). A more technical description of this behavior is in the sub-bullets below.
        i. If the REST API Backend is running HTTP, then it will always send the required `DSPACE-XSRF-COOKIE` cookie with a value of `SameSite=Lax`. This setting means that the cookie will *not* be sent (by your browser) to any other domains. Effectively, this will block all logins from any domain that is not the same as the REST API (as this cookie will not be sent back to the REST API as required for CSRF validation). In other words, running the REST API on HTTP is only possible if the User Interface is running on the exact same domain. For example, running both on 'localhost' with HTTP is a common development setup, and this will work fine.
        ii. In order to allow for cross-domain logins, you MUST enable HTTPS on the REST API. This will result in the `DSPACE-XSRF-COOKIE` cookie being set to `SameSite=None; Secure`. This setting means the cookie will be sent cross domain, but only for HTTPS requests. It also allows the user interface (or other client applications) to be on any domain, provided that the domain is trusted by CORS (see `rest.cors.allowed-origins` setting in REST API)
3. Verify that your User Interface's "rest" section matches the value of "`dspace.server.url`" configuration on the Backend. This simply ensures your UI is sending requests to the correct REST API. Also pay close attention that both specify HTTPS when necessary (see previous bullet).
4. Verify that your "`dspace.server.url`" configuration on the Backend matches the primary URL of the REST API (i.e. the URL you see in the browser). This must be an exact match: mode (http vs https), domain, port, and subpath(s) all must match, and it *must not end in a trailing slash* (e.g. "https://demo.dspace.org/server" is valid, but "https://demo.dspace.org/server/" may cause problems).
5. Verify that your "`dspace.ui.url`" configuration on the Backend matches the primary URL of your User Interface (i.e. the URL you see in the browser). This must be an exact match: mode (http vs https), domain, port, and subpath(s) all must match, and it *must not end in a trailing slash* (e.g. "https://demo.dspace.org" is valid, but "https://demo.dspace.org/" may cause problems).
6. Verify that nothing (e.g. a proxy) is blocking Cookies and HTTP Headers from being passed between the UI and REST API. DSpace's CSRF protection relies on the client (User Interface) being able to return both a valid `DSPACE-XSRF-COOKIE` cookie and a matching `X-XSRF-TOKEN` header back to the REST API for validation. See our REST Contract for more details https://github.com/DSpace/RestContract/blob/main/csrf-tokens.md
7. If you are running a custom application, or accessing the REST API from the command-line (or other third party tool like Postman), you MUST ensure you are sending the CSRF token on every modifying request. See our REST Contract for more details https://github.com/DSpace/RestContract/blob/main/csrf-tokens.md

For additional information on how DSpace's CSRF Protection works, see our REST Contract at https://github.com/DSpace/RestContract/blob/main/csrf-tokens.md

Using a Self-Signed SSL Certificate causes the Frontend to not be able to access the Backend

If you setup the backend to use HTTPS with a self-signed SSL certificate, then Node.js (which the frontend runs on) may not "trust" that certificate by default. This will result in the Frontend not being able to make requests to the Backend.

One possible workaround (untested as of yet) is to try setting the NODE_EXTRA_CA_CERTS environment variable (which tells Node.js to trust additional CA certificates).

```
# May be necessary for self-signed certificates.
export NODE_EXTRA_CA_CERTS="/etc/ssl/my.dspace.pem"
```

Another option is to avoid using a self-signed SSL certificate. Instead, create a real, issued SSL certificate using something like Let's Encrypt (or similar free services)

My REST API is running under HTTPS, but some of its "link" URLs are switching to HTTP

This scenario may occur when you are running the REST API behind an HTTP proxy (e.g. Apache HTTPD's `mod_proxy_http`, Ngnix's `proxy_pass` or any other proxy that is forwarding from HTTPS to HTTP).

The fix is to ensure the DSpace REST API is sent the `X-Forwarded-Proto` header (by your proxying service), telling it that the forwarded protocol is HTTPS

```
X-Forwarded-Proto: https
```

In general, when running behind a proxy, the DSpace REST API depends on accurate X-Forwarded-* headers to be sent by that proxy.

My User Interface's robots.txt has incorrect sitemap URLs

This scenario may occur when you are running the User Interface behind an HTTP proxy (e.g. Apache HTTPD's `mod_proxy_http`, Ngnix's `proxy_pass` or any other proxy that is forwarding from HTTPS to HTTP).

The fix is to ensure the DSpace User Interface (frontend) is sent the correct `X-Forwarded-Proto` and `Host` (or X-Forwarded-Host) headers to tell it the correct hostname and scheme (HTTP or HTTPS)

Apache HTTPD example

```
ProxyPreserveHost on
RequestHeader set X-Forwarded-Proto https
```

Cannot upload file from User Interface

If everything seems to be working, but you cannot upload files, it's important to first check your logs for any possible backend errors. See the Troubleshoot an error page.

If you are running DSpace on a Debian-based system (e.g. Ubuntu), some users have reported that it's **required** grant "ReadWrite" access to Apache Tomcat (where the backend is running) via the service file (e.g. /lib/systemd/system/tomcat9.service). In the `[Service]` section you need to add something like this:

```
# Give Tomcat read/write on the DSpace installation
# Make sure to update the "/PATH/TO" to be the full path of your DSpace install
ReadWritePaths=/PATH/TO/dspace

# NOTE: If you don't want to give Tomcat read/write to all of DSpace,
# you could limit this further to just these folders
# dspace/assetstore
# dspace/solr
# dspace/log
```

Another area to check is your maximum upload file size. Spring Boot settings can be overridden in local.cfg, and/or you can also check your web server configuration for the directives "client_max_body_size" (Nginx) or "LimitRequestBody" (Apache HTTP) that specify the maximum allowable file size.

Javascript heap out of memory

On some versions of Node.js or some operating systems, sites have reported seeing a "Javascript heap out of memory" error when trying to run the User Interface *in development mode* (`npm run start:dev`). This does not seem to occur on every system, but the fix is always the same. You should ensure that Node.js is given *at least* 4GB of memory via the "NODE_OPTIONS" environment variable

```
# Set the "NODE_OPTIONS" environment variable on your system. This example will work for Linux/macOS
# Ensure the "max-old-space-size" is set to 4GB (4096MB) or greater.
export NODE_OPTIONS=--max-old-space-size=4096
```

NOTE: More discussion on this issue can be found in https://github.com/DSpace/dspace-angular/issues/2259 It appears to only occur on systems where the default memory allocated for Node isn't sufficient to build DSpace in development mode.

This same setting may also be used in production scenarios to give Node.js more memory to work with. See Performance Tuning DSpace for more details.

Solr responds with "Expected mime type application/octet-stream but got text/html" (404 Not Found)

This error occurs when Solr is either not initialized properly, or your DSpace backend is unable to find/communicate with Solr. Here's a few things you should double check:

1. Verify that Solr is running and/or check for errors in its logs. Try to restart it (usually via a command like `[solr]/bin/solr restart`), and verify it's accessible via wget or a web browser (usually at a URL like `http://localhost:8983/solr`)
2. Verify that your `solr.server` setting (in local.cfg) is correct for your Solr installation. This should correspond to the main URL of your Solr site (usually something like `http://localhost:8983/solr`). If you use `wget` or a browser from the machine running your DSpace backend, you should get a response from that URL (it should return the Solr Admin UI).
3. Verify that the required DSpace Solr cores have been properly installed/configured (per installation instructions above). When properly installed, you should be able to get a response from them. For example, the URL `${solr.server}/search/select` should run an empty query against the "search" core, returning an empty JSON result.
4. If Solr is running & you are sure `solr.server` is set properly, double check that nothing else could be blocking the DSpace backend from accessing Solr. For instance, if Solr is on a separate machine, verify that there is no firewall or proxy that could be blocking access between the DSpace backend machine and the Solr machine.

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.

Upgrading DSpace

What versions does this guide cover?

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

- *Upgrading ANY prior version (1.x.x, 3.x, 4.x, 5.x, 6.x, 7.x or 8.x) of DSpace to DSpace 8.x (latest version)*

For **major** upgrades, you may also want to consider Migrating DSpace to a new server. This guide provides a walkthrough of installing a fresh copy of latest DSpace, and migrating your existing production data into it.

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

- Releases - Provides links to release notes for all prior releases of DSpace
- Version History - Provides detailed listing of all changes in all prior releases of DSpace

Should I upgrade or migrate my current installation?

DSpace offers two approaches to getting your installation up-to-date.

- Upgrading to the latest release (covered by this guide). This provides a walkthrough of how to install the latest version of the code *over top* of your existing DSpace installation, in order to upgrade to the latest version.
- Migrating to the latest version. This separate guide provides a walkthrough of installing a new copy of DSpace and migrating your existing production data into it. This approach may be more useful if you wish to move your DSpace to a different server, or want to start "fresh" with the same data.

The approach you choose is up to you. Upgrading is often easiest for minor upgrades (e.g. 8.x latest 8.x). Migrating may be useful for major upgrades (e.g. 7.x 8.x), or if you need to move your DSpace installation.

Please refrain from customizing the DSpace database tables. It will complicate your next upgrade!

As DSpace automatically upgrades your database structure (using FlywayDB migrations), *we highly recommend AGAINST customizing the DSpace database tables/structure or backporting any features that change the DSpace tables/structure*. Doing so will often cause the automated database upgrade process to fail (and therefore will complicate your next upgrade).

If you must add features requiring new database tables/structure, we recommend creating new tables (instead of modifying existing ones), as that is usually much less disruptive to our automated database upgrade.

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 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 checkout your well tested and upgraded code.

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

1 Release Notes / Significant Changes
2 Upgrading the Backend (Server API)
  2.1 Backup your DSpace Backend
  2.2 Update Backend Prerequisite Software
  2.3 Upgrading the Backend Steps
3 Upgrading the Frontend (User Interface)
4 Troubleshooting Upgrade Issues
  4.1 See all "Common Installation Issues"
  4.2 Database migrate errors: "Migration V5.7_2017.04.11__DS-3563_Index_metadatavalue_resource_type_id_column.sql failed" or "Migration V5.7_2017.05.05__DS-3431.sql failed"
  4.3 Database migrate errors because of custom database tables/columns
  4.4 Running "Ignored" Flyway Migrations
  4.5 Manually updating the Metadata Registries

Release Notes / Significant Changes

DSpace 9.0 features some breaking changes which you may wish to be aware of before beginning your upgrade:

- **"Breaking Changes" section of Release Notes**
- Keep in mind, if you are "skipping over" a major verion (e.g. upgrading directly from 7.x to 10.x, skipping both the 8.x and 9.x releases), then you should also check the "Release Notes / Significant Changes" section of all major versions you are skipping over.

Upgrading the Backend (Server API)

Backup your DSpace Backend

Before you start your upgrade, it is strongly recommended that you create a backup of your DSpace content. 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 `[dspace]/config/spring/api/bitstore.xml`)
- 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.
- Statistics data: what to back up depends on what you were using before: the options are the default SOLR Statistics, or the legacy statistics. Legacy stats utilizes the dspace.log files, while SOLR Statistics stores data in `[dspace]/solr/statistics`. A simple copy of the logs or the Solr core directory tree should give you a point of recovery, should something go wrong in the update process. We can't stress this enough: your users depend on these statistics more than you realize. You need a backup.
- Authority data: stored in `[dspace]/solr/authority`. As with the statistics data, making a copy of the directory tree should enable recovery from errors.

Update Backend Prerequisite Software

DSpace 10.x requires the following updated versions of prerequisite software (when compared to DSpace 9.x).

- **Updated:** Java JDK 21 is required
- **Updated**: Apache Maven 3.9.x or later is required
- **Updated:** Node v20.19+ is required

Refer to the Backend Requirements section of "Installing DSpace" for more details around configuring and installing these prerequisites.

Upgrading the Backend Steps

Migration guide also available

If during the upgrade you are migrating your DSpace backend to a new server/machine, see Migrating DSpace to a new server instead. The migration process documented on that page can be used as an alternative to this upgrade procedure.

Additional steps when upgrading from DSpace 6.x or lower

If upgrading from 6.x or below, you will need to follow some additional steps. These steps have been moved to our guide for Upgrading from DSpace 6.x or lower.

1. **Download** the latest DSpace release from the DSpace GitHub Repository. You can choose to either download the zip or tar.gz file provided by GitHub, or you can use "git" to checkout the appropriate tag (e.g. `dspace-9.0`) or branch.
    a. Unpack it using "unzip" or "gunzip". If you have an older version of DSpace installed on this same server, you may wish to unpack it to a different location than that release. This will ensure no files are accidentally overwritten during the unpacking process, and allow you to compare configs side by side.
    b. 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.
2. **Build DSpace Backend.** Run the following commands to compile DSpace :

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

   The above command will re-compile the DSpace source code and build its "installer". You will find the result in `[dspace-source]/dspace/target/dspace-installer`
3. **Stop Tomcat (or servlet container).** Take down your servlet container.

    a. For Tomcat, use the `$CATALINA_HOME/shutdown.sh` script. (Many Unix-based installations will have a startup/shutdown script in the `/etc/init.d` or `/etc/rc.d` directories.)
4. **Update your DSpace Configurations.** Depending on the version of DSpace you are upgrading from, not all steps are required.
    a. If you haven't already, please review the Release Notes for details about new features or configuration changes. There may be new configuration you wish to configure, or similar.
    b. *Make sure your existing local.cfg is in the source directory* (e.g. `[dspace-source]/dspace/config/local.cfg`). That way your existing configuration gets reinstalled alongside the new version of DSpace.
    c. **If you are upgrading from DSpace 7.x**, *you will need to perform the following steps. If you are upgrading from 8.x or a prior version of 9.x, skip this and move along.*
        i. As of DSpace 8, the "db.dialect" configuration has changed from "org.hibernate.dialect.PostgreSQL94Dialect" to "org.dspace.util.DSpacePostgreSQLDialect". Therefore, MAKE SURE that your dspace.cfg or local.cfg has this setting:

```
db.dialect = org.dspace.util.DSpacePostgreSQLDialect
```

5. **Update DSpace Installation.** Update the DSpace installation directory with the new code and libraries. Issue the following commands:

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

6. **Upgrade your database**. The DSpace code will automatically upgrade your database (*from any prior version of DSpace*). By default, this database upgrade occurs automatically when you restart Tomcat (or your servlet container). However, if you have a large repository or are upgrading across multiple versions of DSpace at once, you may wish to manually perform the upgrade (as it could take some time, anywhere from 5-15 minutes for large sites).
    a. (Optional) If desired, you can optionally verify which migrations have not yet been run on your database. You can use this to double check that DSpace is recognizing your database version appropriately

    ```
    [dspace]/bin/dspace database info

    # The response will be a list all migrations which were previously run,
    # along with any which are "PENDING" or "IGNORED"
    # that need to be run to upgrade your database.
    ```

    b. (Optional) In some rare scenarios, if your database's "sequences" are outdated, inconsistent or incorrect, a database migration error may occur (in your DSpace logs). While this is seemingly a rare occurrence, you may choose to run the "update-sequences" command PRIOR to upgrading your database. If your database sequences are inconsistent or incorrect, this "update-sequences" command will auto-correct them (otherwise, it will do nothing).

    ```
    # This command only works if upgrading from DSpace 7.x or later
    [dspace]/bin/dspace database update-sequences

    # If upgrading from DSpace 6 or below, this script had to be run via psql from [dspace]/etc
    /postgres/update-sequences.sql
    # For example:
    # psql -U [database-user] -f [dspace]/etc/postgres/update-sequences.sql [database-name]

    # NOTE: It is important to run the "update-sequences" script which came with the OLDER version of
    DSpace (the version you are upgrading from)!  If you've misplaced
    # this older version of the script, you can download it from our codebase & run it via the "psql"
    command above.
    # DSpace 6.x version of "update-sequences.sql": https://github.com/DSpace/DSpace/blob/dspace-6_x
    /dspace/etc/postgres/update-sequences.sql
    # DSpace 5.x version of "update-sequences.sql": https://github.com/DSpace/DSpace/blob/dspace-5_x
    /dspace/etc/postgres/update-sequences.sql
    ```

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

    ```
    # If upgrading from DSpace 7.x, 8.x or 9.x
    [dspace]/bin/dspace database migrate ignored

    # If upgrading from an earlier version of 10.x
    [dspace]/bin/dspace database migrate
    ```

    d. *If the database upgrade process fails or throws errors,* then look at the "Troubleshooting Upgrade Issues" section below for possible tips /hints.
    e. More information on the "database" command can be found in Database Utilities documentation.

⚠ By default, your site will be automatically reindexed after a database upgrade

If any database migrations are run (even during minor release upgrades), then by default DSpace will automatically reindex all content in your site. This process is run automatically in order to ensure that any database-level changes are also immediately updated within the search/browse interfaces. See the notes below under "**Restart Tomcat (servlet container)**" for more information.

However, you may choose to **skip automatic reindexing.** Some sites choose to run the reindex process manually in order to better control when /how it runs.

To disable automatic reindexing, set `discovery.autoReindex = false` in `config/local.cfg` or `config/modules/discovery.cfg`.

As you have disabled automatic reindexing, make sure to manually reindex your site by running `[dspace]/bin/dspace index-discovery -b` *(This must be run after restarting Tomcat)*

*WARNING: It is not recommended to skip automatic reindexing, unless you will **manually reindex** at a later time, or have verified that a reindex is not necessary. Forgetting to reindex your site after an upgrade may result in unexpected errors or instabilties.*

7. **Deploy Server web application:** Two deployment options are now available for the DSpace backend.

a. *WAR Deployment via Tomcat* (traditional approach): In this approach, you *must* have Tomcat (or a different servlet container) installed locally. You will need to deploy the "server" webapp (in `[dspace]/webapps/server`) into your Servlet Container (e.g. Tomcat). Generally, there are two options (or techniques) which you could use...either configure Tomcat to find the DSpace "server" webapp, or copy the "server" webapp into Tomcat's own webapps folder. For more information & example commands, see the Installation Guide
b. *Runnable JAR deployment* (NEW in v8): In this approach, you can *remove* any existing Tomcat installation. Instead you would run the DSpace backend from the "server-boot" JAR as described in the Installation Guide.

Server-boot execution

```
java -jar [dspace]/webapps/server-boot.jar
```

8. **Update/Install the Solr cores and rebuild your indexes.** ***This may be done after starting the backend (e.g. via Tomcat), but is required for some features to function properly.***
   a. Copy the new, empty Solr cores to your new Solr instance.

```
cp -R [dspace]/solr/* [solr]/server/solr/configsets
chown -R solr:solr [solr]/server/solr/configsets
```

   b. Start Solr, or restart it if it is running, so that these new cores are loaded.

```
[solr]/bin/solr restart
```

   c. You can check the status of Solr and your new DSpace cores by using its administrative web interface. Browse to `${solr.server}` (e.g. `http://localhost:8983/solr/`) to see if Solr is running well, then look at the cores by selecting (on the left) Core Admin or using the Core Selector drop list.
      i. For example, to test that your "search" core is setup properly, try accessing the URL `${solr.server}/search/select`. It should run an empty query against the "search" core, returning an empty JSON result. If it returns an error, then that means your "search" core is missing or not installed properly.
9. **Restart Tomcat (servlet container) or Runnable JAR.** Now restart your servlet container (Tomcat/Jetty/Resin) or Runnable JAR and test out the upgrade.
   a. **Upgrade of database:** If you didn't manually upgrade your database in the previous step, then your database will be automatically upgraded to the latest version. This may take some time (seconds to minutes), depending on the size of your repository, etc. Check the DSpace log (`[dspace]/log/dspace.log.[date]`) for information on its status.
10. **Reindexing of all content for search/browse:** If your database was just upgraded (either manually or automatically), all the content in your DSpace will be automatically re-indexed for searching/browsing. As the process can take some time (minutes to hours, depending on the size of your repository), it is performed in the background; meanwhile, DSpace can be used as the index is gradually filled. *But, keep in mind that not all content will be visible until the indexing process is completed.* Again, check the DSpace log (`[dspace]/log/dspace.log.[date]`) for information on its status. *If you wish to skip automatic reindexing, please see the Note above under the "Upgrade your Database" step.*
    a. To reindex manually, just run:

```
[dspace]/bin/dspace index-discovery -b
```

    b. You may also wish to reindex OAI-PMH content at this time:

```
[dspace]/bin/dspace oai import
```

11. **Review / Update your scheduled tasks (e.g. cron jobs).** For all features of DSpace to work properly, there are some scheduled tasks you MUST setup to run on a regular basis. Some examples are tasks that help create thumbnails (for images), do full-text indexing (of textual content) and send out subscription emails. See the Scheduled Tasks via Cron for more details.
    a. *If upgrading from 7.4 (or earlier)* , you will want to make sure the new "subscription-send" task is added to your existing scheduled tasks (in cron or similar). This new task is in charge of sending Email Subscriptions for any users who have subscribed to updates. (NOTE: "subscription-send" *replaces* the older "sub-daily" task from 6.x or below). See the Scheduled Tasks via Cron for more details.
12. **Migrate DOI metadata.** In DSpace 10, the metadata field and format for DOIs has changed. You should strongly consider running the DOI migrator tool to ensure DOIs minted or imported before the upgrade are consistent with those that will be minted and imported after the update.
13. **Install or Upgrade the new User Interface (see below)**

Upgrading the Frontend (User Interface)

1. **Upgrade Node.js if necessary** based on the version of Node.js listed in the Installing the Frontend (User Interface) documentation
2. **Download the latest dspace-angular release from the DSpace GitHub repository**. You can choose to either download the zip or tar.gz file provided by GitHub, or you can use "git" to checkout the appropriate tag (e.g. `dspace-9.0`) or branch.
   a. If you've cloned or copied this code into your own GitHub or GitLab repository, you may wish to simply pull the latest tagged code into your codebase using Git. That will allow you to more easily address any "code conflicts" between your local changes and the new version of DSpace (if any are found).
   b. NOTE: For the rest of these instructions, we'll refer to the source code location as `[dspace-angular]`.

3. **Install any updated local dependencies** using NPM in the `[dspace-angular]` source code directory:

```
# change directory to our repo
cd [dspace-angular]

# Remove any cached dependencies from older DSpace releases
npm run clean

# install/update the local dependencies
npm install
```

4. **Build the latest User Interface code for Production**: This rebuilds the latest code into the `[dspace-angular]/dist` directory

```
npm run build:prod
```

5. **Review Configuration changes** to see if you wish to update any new configurations
    a. See the Release Notes, especially paying close attention to the "Breaking Changes" section
6. **Update your theme (if necessary)**, if you've created a custom theme in "src/themes" (or modified the existing "custom" or "dspace" themes in that location). Pay close attention to the following...
    a. *If you are upgrading from 7.0, 7.1 or 7.2,* a new "eager-theme.module.ts" and "lazy-theme.module.ts" has been added to both the "custom" and "dspace" themes to improve performance. Make sure to copy those to your custom theme. Additionally, this new "eager-theme.module.ts" for your theme MUST be imported/enabled in "src/themes/eager-themes.module.ts". For example, for a local theme under "src/theme/my-theme":

src/themes/eager-themes.module.ts

```
import { EagerThemeModule as MyThemeEagerThemeModule } from './my-theme/eager-theme.module';
...
@NgModule({
  imports: [
    MyThemeEagerThemeModule,
  ],
})
```

    b. *If you are upgrading from 7.x, you must migrate to standalone components.* To migrate your theme, you need to convert its components to the standalone architecture. To do so, simply follow the following steps:
        i. Fix any errors in all "<...>.module.ts" files in your theme folder. These errors may be mostly caused by importing old modules that are now deleted: simply delete any lines that refer to such modules;
        ii. Run the command "ng generate @angular/core:standalone --path src/themes/<theme-folder>" to migrate the theme components to the standalone architecture, replacing <theme-folder> as necessary. *WARNING: running the command while there are still errors on any <...>.module.ts will not produce the correct result, so make sure you have remedied those errors as mentioned in the previous step.*
    c. *If you are upgrading from 8.x, you must migrate to using Angular Control Flow syntax in all HTML templates (*.component.html file).* This means that all usages of "ngIf" and "ngFor" are now replaced with their equivalent "control flow" syntax (e.g. "@if" and "@for"). If your theme has any custom HTML templates, they may require updates. Angular provides an automated migration tool which can be used to perform this migration (this is the same tool that was used to migrate all DSpace code in #3997).
    d. *As of 9.0, all SASS/CSS styles use Bootstrap 5.* This means that some CSS and SASS styles have changed slightly. This may also impact your themes, as they may need to migrate Bootstrap 4 styles into Bootstrap 5 styles. See the Bootstrap 5 migration guide for details. For additional examples, see #3506 which updated DSpace to use Bootstrap 5.
    e. *As of 10.0,* theme configuration has been updated and simplified. See the "Upgrading from 9.x to 10.x" section of the User Interface Customization page.
    f. Additional minor changes may have been made. It's usually best to look for changes to whichever theme you started from. If you started your theme from the "custom" theme, look for any new changes made under "/src/themes/custom". If you started your theme from the "dspace" theme, look for any new changes made under "/src/themes/dspace".
        i. Using a tool like "git diff" from the commandline is often an easy way to see changes that occurred only in that directory.

```
# Example which will show all the changes to "src/themes/dspace" (and all subfolders)
# between dspace-8.1 (tag) and dspace-9.0 (tag)
git diff dspace-8.1 dspace-9.0 -- src/themes/dspace/
```

    g. For the "custom" theme, the largest changes are often:
        i. New themeable components (subdirectories) may be added under "src/themes/custom/app", allowing you the ability to now change the look & feel of those components.
        ii. The "src/themes/custom/theme.module.ts" file will likely have minor updates. This file registers any new themeable components (in the "`const DECLARATIONS`" section), and also registers new Modules, i.e. new UI features, (in the "`@NgModule`" "imports" section). Make sure those sections are updated in your copy of this file!
        iii. Sometimes, new styles may be added in the "styles" folder, or new imports to "styles/theme.scss"
        iv. If you have locally customized the styles or look & feel of any component, you should also verify that the component itself (in src/app) hasn't had updates.

h. For the "dspace" theme, the largest changes are often:
    i. Existing customized components (subdirectories) under "src/themes/dspace/app/" may have minor updates, if improvements were made to that component.
    ii. The "src/themes/custom/theme.module.ts" file will likely have minor updates. This file registers any new themeable components (in the "`const DECLARATIONS`" section), and also registers new Modules, i.e. new UI features, (in the "`@NgModule`" "imports" section). Make sure those sections are updated in your copy of this file!
    iii. Sometimes, new styles may be added in the "styles" folder, or new imports to "styles/theme.scss"
    iv. If you have locally customized the styles or look & feel of any additional component, you should also verify that the component itself (in src/app) hasn't had updates.

7. **Restart the User Interface.**
    a. If you are using PM2 as described in the Installing DSpace instructions, you'd stop it and then start it back up as follows

```
# Stop the running UI
pm2 stop dspace-ui.json

# If you had to update your PM2 configs, you may need to delete your old configuration from PM2
# pm2 delete dspace-ui.json

# Start it back up
pm2 start dspace-ui.json
```

    b. If you are using a different approach, you simply need to *stop the running UI*, and re-run:

```
# First stop the running UI process by killing it (or similar)

# You MUST restart the UI from within the deployment directory
# See Installation Instructions for more info on this directory
cd [dspace-ui-deploy]

# Then restart the UI via Node.js
node ./dist/server/main.js
```

8. **Verify the UI and REST API are both working properly.**
    a. If you hit errors, see the "Troubleshooting Upgrade Issues" section below. Additionally, check the "Common Installation Issues" section of the Installing DSpace documentation for other common misconfiguration or setup issues.

Troubleshooting Upgrade Issues

See all "Common Installation Issues"

At times the upgrade process may involve configuration changes which could result in one of our "Common Installation Issue" error messages. So, make sure to check that section of the Installing DSpace documentation, especially if you find your UI is no longer connecting to your REST API.

Database migrate errors: "Migration V5.7_2017.04.11__DS-3563_Index_metadatavalue_resource_type_id_column.sql failed" or " Migration V5.7_2017.05.05__DS-3431.sql failed"

If you are upgrading to DSpace 7.x and receive *either* of the two following errors after running "./dspace database migrate ignored":

```
Migration V5.7_2017.04.11__DS-3563_Index_metadatavalue_resource_type_id_column.sql failed

[or]

Migration V5.7_2017.05.05__DS-3431.sqln failed
```

This means your database never ran those older migrations during a past upgrade from 5.x6.x (or similar).

Luckily, though, these migrations are both obsolete in DSpace 7.x (and later). This means *you can skip these migration safely*.

As of DSpace 7.5, a new "./dspace database skip" command is provided to easily skip one (or both) of these failing migrations as follows:

```
# If you need to skip "V5.7_2017.04.11__DS-3563_Index_metadatavalue_resource_type_id_column.sql", run this:
./dspace database skip "5.7.2017.04.11"

# If you need to skip "V5.7_2017.05.05__DS-3431.sql", run this:
./dspace database skip "5.7.2017.05.05"
```

For more information on the "./dspace database skip" command see Database Utilities.

Database migrate errors because of custom database tables/columns

If you have modified your database structure directly (or installed custom plugins which do so), then it is possible a "./dspace database migrate" will fail if it encounters an unexpected database structure. In this scenario, you may need to do some manual migrations before the automatic migrations will succeed. The general process would be something like this:

1. Revert back to your current DSpace database
2. Manually upgrade just your database **past** the failing migration. For example, if you are current using DSpace 1.5 and the "V1.6" migration is failing, you may need to first manually upgrade your database to 1.6 compatibility. This may involve either referencing the upgrade documentation for that older version of DSpace, or running the appropriate SQL script from under `[dspace-src]/dspace-api/src/main/resources/org/dspace/storage/rdbms/sqlmigration/`)
3. Then, re-run the migration process from that point forward (i.e. re-run `./dspace database migrate`)

Running "Ignored" Flyway Migrations

During some upgrades, a Flyway database migration will be "ignored." One known instance of this is documented in https://github.com/DSpace/DSpace/issues/6762. If you are upgrading from DSpace 5.x to a later version of DSpace, the migration put in place to address https://github.com/DSpace/DSpace/pull/1128 will be "ignored" because it is not necessary. There is a special command you can run which will un-flag this migration as "ignored."

```
dspace database migrate ignored
```

Manually updating the Metadata Registries

The database migration (./dspace database migrate) should *automatically trigger* your metadata/file registries to be updated (based on the config files in `[dspace]/config/registries/`). However, if this update was NOT triggered, you can also manually run these registry updates (they will not harm existing registry contents) as follows:

```
cd [dspace]/bin/
./dspace registry-loader -metadata ../config/registries/dcterms-types.xml
./dspace registry-loader -metadata ../config/registries/dublin-core-types.xml
./dspace registry-loader -metadata ../config/registries/eperson-types.xml
./dspace registry-loader -metadata ../config/registries/local-types.xml
./dspace registry-loader -metadata ../config/registries/sword-metadata.xml
./dspace registry-loader -metadata ../config/registries/workflow-types.xml
```

Migrating DSpace to a new server

These instructions are meant as a general guideline to how you can migrate your DSpace site/data to a new server while also Upgrading DSpace to the latest release. Keep in mind that you MUST also review the Installing DSpace and Upgrading DSpace guides when performing a migration (e.g. you must ensure you have correct dependencies installed and you must ensure you perform all upgrade steps).

Overview of migration approaches

There are two main approaches to migrating your DSpace to a new server.

1. Install a fresh copy of DSpace & migrate the database/files into it - This is the approach documented on this page. It is the recommended approach as it ensures zero data loss. However, it does involve more steps to complete the migration.
2. Install a fresh copy of DSpace & use the AIP Backup and Restore - This is an alternative approach where you can use the AIP export tools to export AIPs from your old site, and then import them into the new site. While this also works, keep in mind that not all DSpace data can be exported to AIPs, so you will lose some data during this migration (namely any submissions not yet completed or still in workflow approval will be lost, see the AIP Backup and Restore documentation for more details on what data is currently missing from AIPs).

Step 1: Install a fresh copy of DSpace

On your new server, follow the Installing DSpace instructions and install a fresh (empty) copy of the latest version of DSpace. **BEFORE PROCEEDING**, ensure that this fresh copy of DSpace is correctly installed and shows no errors when you startup the site. (The site will obviously appear empty though, and that's OK)

You can also use this time to get your basic configurations setup properly for both the backend (local.cfg) and the frontend (config.prod.yml).

Step 2: Prepare your data to copy from the old DSpace to the new one

There are three main areas of data you need to migrate in order to ensure no data loss.

Perform these steps on the *old server*

1. First, you should STOP tomcat on the old server. These steps require the site to be down.
2. Update sequences *(optional)* - When migrating content, sometimes sites will find that database sequences will be outdated or incorrect. This can result in "duplicate key" errors during the database migration to the latest version. To avoid this, **before** you export your data, run this older copy of the "update-sequences" command on your database. This should ensure your database sequences are updated before you dump your data.

```
# If upgrading from DSpace 6 or below, run this on your old database
psql -U [database-user] -f [dspace]/etc/postgres/update-sequences.sql [database-name]
# e.g. psql -U dspace -f [dspace]/etc/postgres/update-sequences.sql dspace
```

    a. NOTE: It is important to run the "update-sequences" script which came with the OLDER version of DSpace (the version you are migrating from)! If you've misplaced this older version of the script, you can download it from our codebase & run it via the "psql" command above.
        i. DSpace 6.x version of "update-sequences.sql": https://github.com/DSpace/DSpace/blob/dspace-6_x/dspace/etc/postgres/update-sequences.sql
        ii. DSpace 5.x version of "update-sequences.sql": https://github.com/DSpace/DSpace/blob/dspace-5_x/dspace/etc/postgres/update-sequences.sql
3. The database data - Make sure to export the database data from your old DSpace site using a tool like "pg_dump" (for PostgresSQL). If you use "pg_dump", you'll end up with a large SQL file which contains all the data from your old database.

```
# Example of using pg_dump to export a database to an output file
pg_dump -U [db_username] [db_name] > [output_file.sql]
```

4. The "assetstore" folder - This folder is in your DSpace installation directory and it contains all the files stored in your DSpace. You will need **all the contents** of this folder (including all subdirectories), so you could choose to zip it up or you could copy it over directly.
5. The Solr data *(optional)* - Both DSpace authority and statistics are stored in Solr. If you want to keep these, you will want to export them from the old Solr and move them over. Use the "solr-export-statistics" tool provided with DSpace: see "Export SOLR Statistics" in the Solr Statistics Maintenance guide. *(Requires Solr to be running. Keep in mind, this may require you to start Tomcat back up if Solr is running in Tomcat.)*

Step 3: Copy over the prepared data and import it into the new DSpace

Copy the data you've prepared in Step 2 over to the new server.

Now, you'll import this data into your new installation of DSpace (created in Step 1).

Perform these steps on the *new server.*

1. First, you must STOP Tomcat on the new server.
2. The database data - Before you can import the data, you must delete the new, empty database.

a. Delete/Clean the new, empty database (created in step 1) as you will have empty tables created during the installation. The easiest way to achieve this is to run the "./dspace database clean" command. Keep in mind it requires temporarily enabling it via "db.cleanDisabled=false" in your local.cfg. (After the "clean" command succeeds, make sure to remove this configuration.)

```
# Delete everything in your database
# Requires temporarily setting "db.cleanDisabled=false" in your local.cfg
./dspace database clean
```

   i. Alternatively, PostgreSQL users could delete the entire database (using dropdb command, e.g. "dropdb -U [db_username] [db_name]") and recreate it based on the "Database Setup" instructions in Installing DSpace.

b. Import the database dump you created in Step 2 (above), which will recreate this database with all your old data in it. For Postgres, you can use the "psql" command.

```
# Example of using psql to import data from a SQL file into a database
psql -U [db_username] [db_name] < [output_file.sql]
```

(NOTICE the direction of the angle character... in this command you are telling Postgres to execute all the commands contained in your "output_file.sql", which will cause it to recreate all the database data in your new database.)

3. The "assetstore" folder - Delete the **empty** assetstore folder on the new server. Copy the entire assestore folder (and all subdirectories) from the old server to the new one. In the end, you should have a several subdirectory hierarchies (containing your files) under the `[dspace]/assetstore/` folder on the new server.
4. The Solr data (optional) - If you exported the statistics or authority data in Step 2, then you can import this data from the exported files using the "solr-import-statistics" tool provided with DSpace, see "Import SOLR Statistics" in the Solr Statistics Maintenance guide. *(Requires Solr to be running)*

Step 4: Update the database, Start DSpace & Reindex

Now that all the data is copied over, you need to ensure it's updated and reindexed properly (for the new version of DSpace).

Perform these steps on the *new server.*

1. Migrate/Upgrade the database to the latest version - Now that your old data is migrated, you MUST ensure it's using the latest database updates based on the new DSpace you've installed. Review the database steps in Upgrading DSpace and follow the instructions there.

```
# Migrate your old data to the latest DSpace version
# WARNING: You must review the Upgrading DSpace docs to see if there are any additional database steps
listed there!
./dspace database migrate ignored
```

   NOTE: You should check the logs (dspace.log) for errors. Additional steps may be documented in the Upgrading DSpace guide.
2. Start Tomcat. This will bring your new DSpace back up, with the migrated data in place. Check the backend logs (dspace.log and Tomcat log) to ensure no errors occur on startup.
3. Reindex all content - This will ensure all search/browse functionality works in the DSpace site. Optionally, if you use OAI-PMH, you will want to reindex content into there as well.

```
# Reindex all your content in DSpace
./dspace index-discovery -b

# (Optionally) also reindex everything into OAI-PMH endpoint
./dspace oai import
```

   NOTE: Until this command completes (it may take a while for large sites), you will not be able to fully browse/search the content from the User Interface. To check the progress of the reindex, check your dspace.log file.

Step 5: Review the Upgrade Instructions and final cleanup

If you've changed the version of DSpace you are running, you should review the Upgrading DSpace guide for instructions related to necessary configuration changes or other necessary updates. You should perform any upgrade steps that you have NOT yet performed above (keep in mind you already should have upgraded your database, and reindexed your content).

At this time, you also may wish to review your configurations on your *old* DSpace site, and see if there are any configurations that you wish to copy over into your new DSpace site. This step is optional, as you can also choose to start "fresh" with a new local.cfg file.

FINALLY, test the new site and verify that all the content, user accounts, etc. have moved over successfully. If you encounter any issues, see our Troubleshoot an error guide for hints/tips on finding the underlying error message & reporting it to Support lists/channels. Also make sure to check our list of Common Installation Issues in the Installing DSpace guide.

Upgrading from DSpace 6.x or lower

This page provides guidance for upgrading to the latest version of DSpace from an unsupported or older release, such as DSpace 1.x.x, 3.x, 4.x, 5.x or 6.x.

- Additional Upgrade Steps
  - Before Upgrade/Migration
    - Export your Solr Statistics from your existing DSpace site
  - Perform Upgrade/Migration
    - Choose your upgrade method
    - Update your DSpace configuration (if performing manual upgrade)
  - After Upgrade/Migration
    - Import your Solr Statistics from your existing DSpace site
    - Update your Handle Server Configuration

Additional Upgrade Steps

Before Upgrade/Migration

Export your Solr Statistics from your existing DSpace site

If you are using SOLR Statistics and wish to retain your statistical data in the latest version of DSpace, then you MUST export your old Solr statistics data from your existing site.

1. **From your old version of DSpace, dump your `authority` and `statistics` Solr cores.** (Only necessary if you want to keep both your authority records and/or SOLR Statistics)

   ```
   [dspace]/bin/dspace solr-export-statistics -i authority
   [dspace]/bin/dspace solr-export-statistics -i statistics
   ```

   The dumps will be written to the directory `[dspace]/solr-export`. *This may take a long time and require quite a lot of storage.* In particular, the statistics core is likely to be huge, perhaps double the size of the content of `solr/statistics/data`. You should ensure that you have sufficient free storage.

   This is not the same as the disaster-recovery backup that was done above. These dumps will be reloaded into new, reconfigured cores later. If you were **sharding** your statistics data, you will need to dump each shard separately. The index names for prior years will be `statistics-YYYY` (for example: `statistics-2017 statistics-2018` etc.) The current year's statistics shard is named `statistics` and you should dump that one too.
2. **Move your old Solr cores to a safe location** in case of trouble with the upgrade procedure. If you leave them in place, you will get a mixture of old and new files that the new Solr will refuse to load.

Perform Upgrade/Migration

Choose your upgrade method

Perform the DSpace upgrade or migration based on your preferred upgrade method. There are two main options:

1. *(RECOMMENDED)* Install the new version of DSpace and migrate your old data into it. This process is described in Migrating DSpace to a new server
2. Perform a manual upgrade of your site. This process is described in Upgrading DSpace

In most situations, you **may find it easier** to follow the steps in Migrating DSpace to a new server because it will allow you to work with a **fresh installation** of the new DSpace release before migrating your data into that installation. That process is often easier if you plan to upgrade from a very old DSpace release.

Update your DSpace configuration (if performing manual upgrade)

If you choose option two (the manual upgrade), please keep in mind that you will need to remove/modify some of your existing DSpace configuration. These are some of the configurations that used to exist in DSpace 6.x which are no longer valid in DSpace 7 or later:

1. **Review your customized configurations (recommended to be in local.cfg):** As mentioned above, we recommend any local configuration changes be placed in a local.cfg Configuration File. With any major upgrade some configurations may have changed. Therefore, it is recommended to review all configuration changes that exist in the `config` directory, and its subdirectories, concentrating on configurations your previously customized in your local.cfg. See also the Configuration Reference.
2. **Remove obsolete configurations.** With the removal of the JSPUI and XMLUI, a large number of server-side (backend) configurations were made obsolete and were therefore removed between the 6.x and 7.0 release. A full list can be found in the Release Notes.
3. **Remove BTE Spring configuration:** If it exists, remove the `[dspace]/config/spring/api/bte.xml` Spring Configuration. This file is no longer needed as the BTE framework was removed in favor of Live Import from external sources.
4. **Migrate or recreate your Submission configuration.** As of DSpace 7, the submission configuration has changed. The format of the "item-submission.xml" file has been updated, and the older "input-forms.xml" has been replaced by a new "submission-forms.xml". You can choose to

either start fresh with the new v7 configuration files, or you can use the steps below to migrate your old configurations into the new format. See the Submission User Interface for more information

a. First, create a temporary folder to copy your old v6 configurations into

```
# Example of creating a [dspace]/config/temp folder for this migration
# You must replace [dspace] with the full path of your DSpace 7 installation.
cd [dspace]/config
mkdir temp
```

b. Copy your old (v5 or v6) "item-submission.xml" and "input-forms.xml" into that temporary folder

c. Run the command-line migration script to migrate them to v7 configuration files

```
# This example uses [dspace] as a placeholder for all paths.
# Replace it with either the absolute or relative path of these files
[dspace]/bin/dspace submission-forms-migrate -s [dspace]/config/temp/item-submission.xml -f
[dspace]/config/temp/input-forms.xml
```

d. The result will be two files. These are valid v7 configurations based on your original submission configuration files.
    1. `[dspace]/config/item-submission.xml.migrated`
    2. `[dspace]/config/submission-forms.xml.migrated`

e. These "*.migrated" files have *no inline comments*, so you may want to edit them further before installing them (by removing the ".migrated" suffix). Alternatively, you may choose to copy sections of the *.migrated files into the default configurations in the `[dspace]/config/` folder, therefore retaining the inline comments in those default files.

5. **City IP Database file for Solr Statistics has been renamed.** The old `[dspace]/config/GeoLiteCity.dat` file is no longer maintained by its provider. You can delete it. The new file is named `GeoLite2-City.mmdb` by default. If you have configured a different name and/or location for this file, you should check the setting of `usage-statistics.dbfile` in `[dspace]/config/modules/usage-statistics.cfg` (and perhaps move your custom setting to `local.cfg`).
6. **tm-extractors media filtering (WordFilter) no longer exists:** the PoiWordFilter plugin now fulfills this function. If you still have WordFilter configured, remove from `dspace.cfg` and/or `local.cfg` all lines referencing `org.dspace.app.mediafilter.WordFilter` and uncomment all lines referencing `org.dspace.app.mediafilter.PoiWordFilter`.
7. **Re-configure Solr URLs:** change the value of `solr.server` to point at your new Solr external service. It will probably become something like s`olr.server = https://localhost:8983/solr`. Solr only needs to be accessible to the DSpace backend, and should not be publicly available on the web. It can either be run on localhost or via a hostname (if run on a separate server from the backend). Also review the values of
    a. `discovery.search.server`
    b. `oai.solr.url`
    c. `solr.authority.server`
    d. `solr-statistics.server`
8. **Sitemaps are now automatically generated/updated:** A new `sitemap.cron` setting exists in the dspace.cfg which controls when Sitemaps are generated. By default they are enabled to update once per day, for optimal SEO. See Search Engine Optimization docs for more detail
    a. Because of this change, if you had a system cron job which ran "`./dspace generate-sitemaps`", this system cron job can be removed in favor of the new `sitemap.cron` setting.

After Upgrade/Migration

Import your Solr Statistics from your existing DSpace site

If you are using SOLR Statistics and wish to retain your statistical data in the latest version of DSpace, then you MUST import your old Solr statistics data from your existing site. (This step is using the same data that you exported above)

1. **Reload Solr Statistics**
    a. Load `authority` and `statistics` from the CSV dumps that you made earlier in Step 2 above.

```
[dspace]/bin/dspace solr-import-statistics -i authority
[dspace]/bin/dspace solr-import-statistics -i statistics
```

This could take quite some time.

If you had sharded your statistics, you will need to load the dump of each shard separately into the "statistics" core. DSpace 7 does not support Solr shards at this time. Unfortunately, this will involve renaming all CSV export files to remove the year (e.g. rename "statistics-2012_export_2013-12_5.csv" to "statistics_export_2013-12_5.csv") and rerunning "[dspace]/bin/dspace solr-import-statistics -i statistics". More advice on this process can be found in this dspace-tech mailing list thread.

b. For the Statistics core(s) only, upgrade legacy DSpace Object Identifiers (pre-6.4 statistics) to UUID Identifiers.

```
[dspace]/bin/dspace solr-upgrade-statistics-6x -i statistics
```

Again If you had sharded your statistics, you will need to run this for each shard separately. See also SOLR Statistics Maintenance#UpgradeLegacyDSpaceObjectIdentifiers(pre-6xstatistics)toDSpace6xUUIDIdentifiers

2. **(Optional) Set up IP to City database for location-based statistics.** If you wish to (continue to) record the geographic origin of client activity, you will need to install (*and regularly update*) one of the following:
    a. Either, a copy of MaxMind's GeoLite City database (in MMDB format)
        i. NOTE: Installing MaxMind GeoLite2 is *free*. However, you **must** sign up for a (free) MaxMind account in order to obtain a license key to use the GeoLite2 database.
        ii. You may download GeoLite2 directly from MaxMind, or many Linux distributions provide the `geoipupdate` tool directly via their package manager. You will still need to configure your license key prior to usage.
        iii. Once the "GeoLite2-City.mmdb" database file is installed on your system, you will need to configure its location as the value of `usage-statistics.dbfile` in your `local.cfg` configuration file .
        iv. You can discard any old GeoLiteCity.dat database(s) found in the `config/` directory (if they exist).
        v. See the "Managing the City Database File" section of SOLR Statistics for more information about using a City Database with DSpace.
    b. Or, you can alternatively use/install DB-IP's City Lite database (in MMDB format)
        - This database is also free to use, but does **not** require an account to download.
        - Once the "dbip-city-lite.mmdb" database file is installed on your system, you will need to configure its location as the value of `usage-statistics.dbfile` in your `local.cfg` configuration file .
        - See the "Managing the City Database File" section of SOLR Statistics for more information about using a City Database with DSpace.

Update your Handle Server Configuration

Older versions of DSpace used an older Handle Server. If you are using the built-in Handle Server with hdl.handle.net URLs, then you'll need to add the follow to the end of the *server_config* section of your `[dspace]/handle-server/config.dct` file (the only new line is the "enable_txn_queue" line)

```
"case_sensitive" = "no"
"storage_type" = "CUSTOM"
"storage_class" = "org.dspace.handle.HandlePlugin"
"enable_txn_queue" = "no"
```

Alternatively, you could re-run the `./dspace make-handle-config` script, which is in charge of updating this `config.dct` file.

Using DSpace

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

Audit Trail feature

Error rendering macro 'toc'

null

Audit Overview

The Audit feature in DSpace offers a structured way to capture and review significant events across DSpace objects. It tracks changes to content, metadata, and other key operations, allowing administrators to:

- Monitor system activity for compliance purposes
- Troubleshoot issues
- Maintain a historical record of interactions

Functionality

At its core, the auditing system logs events that modify the state of DSpace objects, including:

- Creation
- Updates
- Deletions
- Removals

Each audit event captures detailed contextual information:

- **Event type**
- **Object affected**
- **User who performed the action**
- **Timestamp**
- (Optional) **metadata for additional insight**
- (Optional) **checksum for bitstreams**

Audit events are managed through a dedicated service layer for storing and retrieving records. A system listener observes internal operations and identifies which events should be logged, ensuring only meaningful actions are recorded.

For external access, a REST API exposes audit data, allowing system administrators to inspect or filter logs based on specific criteria.

Configuration and Access

Auditing can be toggled through a configuration setting:

```
audit.enabled = true
```

When enabled, the feature begins capturing events automatically. Additional properties:

- **`audit.item.in-workflow`**: Enables auditing of items in the workflow state
- **`audit.item.in-workspace`**: Enables auditing of items in the workspace

Note: Ensure the new audit core has been deployed to the SOLR service.

Administrators can view audit logs via the REST endpoint:

```
/api/system/auditevents
```

This endpoint supports filtering (e.g., by object UUID) and provides detailed information for each event. For example:

```
/api/system/auditevents/{id}
```

returns the full details of a specific event.

Audit Core management

Overview

The **Solr Core Export/Import Tool** enables the complete export of a SOLR core to files (in **CSV** or **JSON** format) and the subsequent import of those files into a target SOLR core.
It supports **multithreaded execution** and **automatic export splitting** by date intervals.

This tool is designed for **data migration**, **partial backups based on time ranges**, and **data transfers between environments**.
The export/import logic is cleanly separated from file writing and user logging mechanisms.

Key Features

- Full **export and import** of a SOLR core.
- **Output formats**: CSV and JSON.
- **Automatic splitting** of exports by date interval (`WEEK`, `MONTH`, or `YEAR`).
- **Configurable multithreading** for parallel export and import operations.
- **Robust design** with minimal dependencies:
    - Simple HTTP status code check for SOLR queries with date filters.

Command-Line Options

Option	Required	Description
`-m`, `--mode`		Execution mode. Allowed values: `export` or `import`.
`-c`, `--core`		SOLR core name — source core for export or target core for import.
`-d`, `--directory`		Directory where files are written (export) or read (import).
`-f`, `--format`		File format: `csv` or `json`. **Default**: `csv`.
`-t`, `--threads`		Number of worker threads to use. **Default**: `1`. **(single thread)**
`-s`, `--start-date`		Start date for export in format `yyyy-MM-dd`. **If not provided, the tool attempts to retrieve the earliest date from the core**.
`-e`, `--end-date`		End date for export in format `yyyy-MM-dd`. **If not provided, the tool attempts to retrieve the latest date from the core.**
`-i`, `--increment`		Date range increment for automatic export splitting: `WEEK`, `MONTH`, or `YEAR`. Default: `MONTH`.
`-h`, `--help`		Displays help information.

Typical Use Cases

- Migrating data between SOLR environments (e.g., test production).
- Creating partial backups limited to specific time periods.
- Rebuilding a core from exported data.
- Parallelizing import/export operations for large datasets.

Example Usage

Export example:

```
./dspace solr-core-management --mode export --core audit --directory /tmp/export --format csv --threads 4 --increment WEEK
```

Import example:

```
./dspace solr-core-management --mode import --core audit --directory /tmp/export --format csv --threads 2
```

Further Script details

please check the dedicated Solr Data Management page

Audit Logs

By default, when auditing is enabled, audit event logs are also written to configurable log files. This is managed through the `log4j2.xml` configuration file. The logging level for audit events can be set using the following property:

```
<Property name='loglevel.audit.events'>ALL</Property>
```

- **ALL** can be used to track all the Audit details
- **OFF** can be used to turn off the Audit Log trace

Administrators can adjust the logging behavior and destination by modifying the **log4j2.xml** file as needed.

Logging audit events to the **audit.log** file is not alternative to indexing data on SOLR. All the Audit events are stored on SOLR as well so data be easily accessible from the DSpace UI

Audit Logs Example

```
2025-10-07 11:29:51,709 INFO  org.dspace.app.audit.event @ AUDIT_EVENT eventType=MODIFY_METADATA
subjectUUID=bf61d08c-8521-43bc-81c6-54dbe53821b9 subjectType=ITEM objectUUID=null objectType=null
metadataField=dc_description_provenance value=Submitted by aaa aaa (aaa) on 2025-10-07T09:29:41Z

No. of bitstreams: 1

Screenshot from 2025-10-06 10-42-15.png: 67019 bytes, checksum: 189a3d28d4945e5cc5a5478c7a8f748b (MD5)
authority= confidence=-1 place=0 action=ADD checksum= datetime=1759829381380 epersonUUID=d31b5858-75e2-4ffa-
872d-b57bc21e1dfa

2025-10-07 11:29:51,712 INFO  org.dspace.app.audit.event @ AUDIT_EVENT eventType=MODIFY_METADATA
subjectUUID=bf61d08c-8521-43bc-81c6-54dbe53821b9 subjectType=ITEM objectUUID=null objectType=null
metadataField=dc_identifier_uri value=http://localhost:4000/handle/123456789/22 authority= confidence=-1
place=0 action=ADD checksum= datetime=1759829381422 epersonUUID=d31b5858-75e2-4ffa-872d-b57bc21e1dfa

2025-10-07 11:29:51,715 INFO  org.dspace.app.audit.event @ AUDIT_EVENT eventType=MODIFY_METADATA
subjectUUID=bf61d08c-8521-43bc-81c6-54dbe53821b9 subjectType=ITEM objectUUID=null objectType=null
metadataField=dc_date_accessioned value=2025-10-07T09:29:41Z authority= confidence=-1 place=0 action=ADD
checksum= datetime=1759829381431 epersonUUID=d31b5858-75e2-4ffa-872d-b57bc21e1dfa

2025-10-07 11:29:51,717 INFO  org.dspace.app.audit.event @ AUDIT_EVENT eventType=MODIFY_METADATA
subjectUUID=bf61d08c-8521-43bc-81c6-54dbe53821b9 subjectType=ITEM objectUUID=null objectType=null
metadataField=dc_date_issued value=2025 authority= confidence=-1 place=0 action=REMOVE checksum=
datetime=1759829381431 epersonUUID=d31b5858-75e2-4ffa-872d-b57bc21e1dfa

2025-10-07 11:29:51,720 INFO  org.dspace.app.audit.event @ AUDIT_EVENT eventType=MODIFY_METADATA
subjectUUID=bf61d08c-8521-43bc-81c6-54dbe53821b9 subjectType=ITEM objectUUID=null objectType=null
metadataField=dc_date_issued value=2025 authority= confidence=-1 place=0 action=ADD checksum=
datetime=1759829381431 epersonUUID=d31b5858-75e2-4ffa-872d-b57bc21e1dfa

2025-10-07 11:29:51,723 INFO  org.dspace.app.audit.event @ AUDIT_EVENT eventType=MODIFY_METADATA
subjectUUID=bf61d08c-8521-43bc-81c6-54dbe53821b9 subjectType=ITEM objectUUID=null objectType=null
metadataField=dc_description_provenance value=Made available in DSpace on 2025-10-07T09:29:41Z (GMT). No. of
bitstreams: 1

Screenshot from 2025-10-06 10-42-15.png: 67019 bytes, checksum: 189a3d28d4945e5cc5a5478c7a8f748b (MD5)

  Previous issue date: 2025 authority= confidence=-1 place=1 action=ADD checksum= datetime=1759829381431
epersonUUID=d31b5858-75e2-4ffa-872d-b57bc21e1dfa

2025-10-07 11:36:32,484 INFO  org.dspace.app.audit.event @ AUDIT_EVENT eventType=CREATE subjectUUID=4f3bbb96-
53f6-4bbf-bc77-dc7dfddd9a82 subjectType=BITSTREAM objectUUID=null objectType=null metadataField= value=
authority= confidence=null place=null action= checksum=38a6ff40a0cc4e9885f01a335039ea6f datetime=1759829792418
epersonUUID=d31b5858-75e2-4ffa-872d-b57bc21e1dfa

2025-10-07 11:36:32,488 INFO  org.dspace.app.audit.event @ AUDIT_EVENT eventType=CREATE subjectUUID=4f3bbb96-
53f6-4bbf-bc77-dc7dfddd9a82 subjectType=BITSTREAM objectUUID=bf61d08c-8521-43bc-81c6-54dbe53821b9
objectType=ITEM metadataField= value= authority= confidence=null place=null action=
checksum=38a6ff40a0cc4e9885f01a335039ea6f datetime=1759829792455 epersonUUID=d31b5858-75e2-4ffa-872d-
b57bc21e1dfa

2025-10-07 11:36:32,490 INFO  org.dspace.app.audit.event @ AUDIT_EVENT eventType=MODIFY_METADATA
subjectUUID=4f3bbb96-53f6-4bbf-bc77-dc7dfddd9a82 subjectType=BITSTREAM objectUUID=bf61d08c-8521-43bc-81c6-
54dbe53821b9 objectType=ITEM metadataField=dc_title value=Screenshot from 2025-10-03 10-37-04.png authority=
confidence=-1 place=0 action=ADD checksum= datetime=1759829792463 epersonUUID=d31b5858-75e2-4ffa-872d-
b57bc21e1dfa

2025-10-07 11:37:29,506 INFO  org.dspace.app.audit.event @ AUDIT_EVENT eventType=DELETE subjectUUID=bf61d08c-
8521-43bc-81c6-54dbe53821b9 subjectType=ITEM objectUUID=null objectType=null metadataField= value= authority=
confidence=null place=null action= checksum= datetime=1759829849388 epersonUUID=d31b5858-75e2-4ffa-872d-
b57bc21e1dfa
```

```
2025-10-07 11:37:29,510 INFO  org.dspace.app.audit.event @ AUDIT_EVENT eventType=DELETE subjectUUID=bbca23c1-
a8d9-492e-bdca-76e214f6ba36 subjectType=BUNDLE objectUUID=null objectType=null metadataField= value= authority=
confidence=null place=null action= checksum= datetime=1759829849394 epersonUUID=d31b5858-75e2-4ffa-872d-
b57bc21e1dfa

2025-10-07 11:37:29,514 INFO  org.dspace.app.audit.event @ AUDIT_EVENT eventType=DELETE subjectUUID=6e80f8e4-
84b5-4fe5-9b6d-dcfe7905262e subjectType=BITSTREAM objectUUID=null objectType=null metadataField= value=
authority= confidence=null place=null action= checksum= datetime=1759829849406 epersonUUID=d31b5858-75e2-4ffa-
872d-b57bc21e1dfa

2025-10-07 11:37:29,518 INFO  org.dspace.app.audit.event @ AUDIT_EVENT eventType=DELETE subjectUUID=6e80f8e4-
84b5-4fe5-9b6d-dcfe7905262e subjectType=BITSTREAM objectUUID=bf61d08c-8521-43bc-81c6-54dbe53821b9
objectType=ITEM metadataField= value= authority= confidence=null place=null action=
checksum=bb9bdc0b3349e4284e09149f943790b4 datetime=1759829849406 epersonUUID=d31b5858-75e2-4ffa-872d-
b57bc21e1dfa

2025-10-07 11:37:29,522 INFO  org.dspace.app.audit.event @ AUDIT_EVENT eventType=DELETE subjectUUID=41770bc0-
d7b2-4e0f-ae6b-3b4b560f5d96 subjectType=BUNDLE objectUUID=null objectType=null metadataField= value= authority=
confidence=null place=null action= checksum= datetime=1759829849417 epersonUUID=d31b5858-75e2-4ffa-872d-
b57bc21e1dfa

2025-10-07 11:37:29,526 INFO  org.dspace.app.audit.event @ AUDIT_EVENT eventType=DELETE subjectUUID=435d498d-
e1ee-4957-becb-8bba70ba5d31 subjectType=BITSTREAM objectUUID=null objectType=null metadataField= value=
authority= confidence=null place=null action= checksum= datetime=1759829849424 epersonUUID=d31b5858-75e2-4ffa-
872d-b57bc21e1dfa

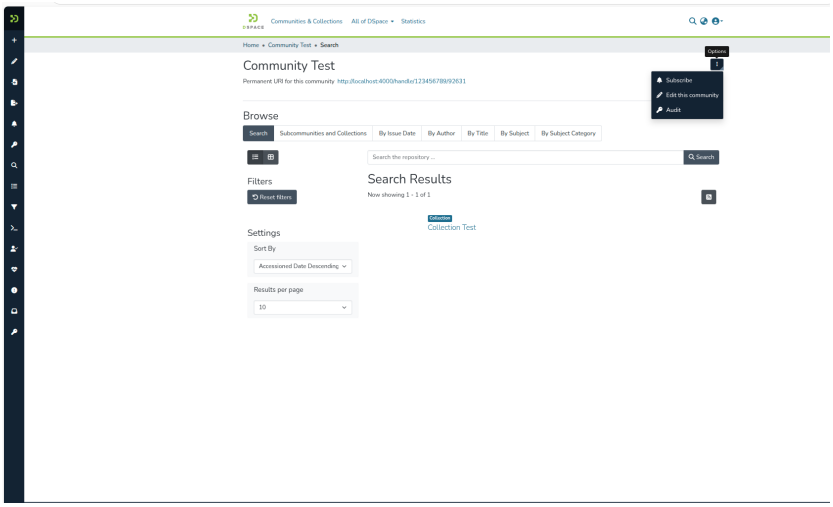
2025-10-07 11:37:29,530 INFO  org.dspace.app.audit.event @ AUDIT_EVENT eventType=DELETE subjectUUID=435d498d-
e1ee-4957-becb-8bba70ba5d31 subjectType=BITSTREAM objectUUID=bf61d08c-8521-43bc-81c6-54dbe53821b9
objectType=ITEM metadataField= value= authority= confidence=null place=null action=
checksum=189a3d28d4945e5cc5a5478c7a8f748b datetime=1759829849424 epersonUUID=d31b5858-75e2-4ffa-872d-
b57bc21e1dfa

2025-10-07 11:37:29,535 INFO  org.dspace.app.audit.event @ AUDIT_EVENT eventType=DELETE subjectUUID=4f3bbb96-
53f6-4bbf-bc77-dc7dfddd9a82 subjectType=BITSTREAM objectUUID=null objectType=null metadataField= value=
authority= confidence=null place=null action= checksum= datetime=1759829849437 epersonUUID=d31b5858-75e2-4ffa-
872d-b57bc21e1dfa

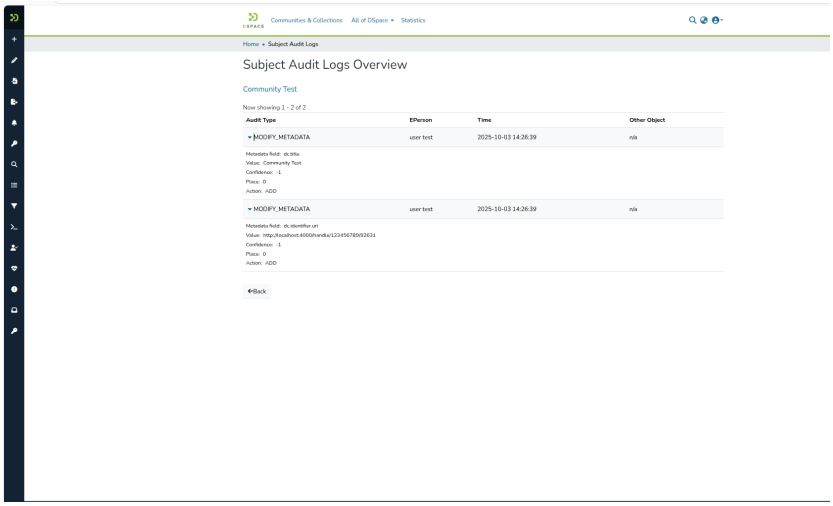
2025-10-07 11:37:29,538 INFO  org.dspace.app.audit.event @ AUDIT_EVENT eventType=DELETE subjectUUID=4f3bbb96-
53f6-4bbf-bc77-dc7dfddd9a82 subjectType=BITSTREAM objectUUID=bf61d08c-8521-43bc-81c6-54dbe53821b9
objectType=ITEM metadataField= value= authority= confidence=null place=null action=
checksum=38a6ff40a0cc4e9885f01a335039ea6f datetime=1759829849437 epersonUUID=d31b5858-75e2-4ffa-872d-
b57bc21e1dfa
```

Community/Collection Audit Example

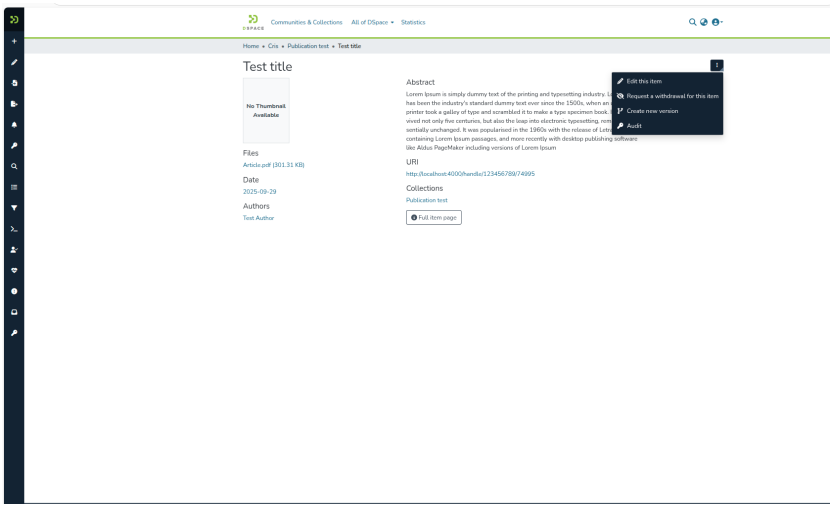
Community audit

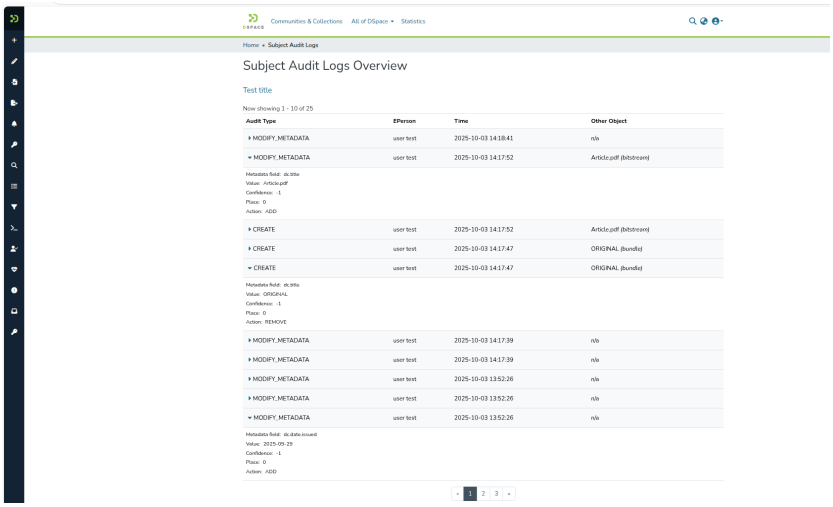


Collection Audit



Item Audit Example





Authentication and Authorization

- Allow Authors to download restricted bitstreams
- Authentication Plugins
- Bulk Access Management
- Embargo
- Managing User Accounts
- Request a Copy

Allow Authors to download restricted bitstreams

When a bitstream is embargoed or restricted in DSpace, standard ACL-based authorization denies download access to everyone without explicit READ permission. But authors who contributed to a publication reasonably expect to access their own work's files — even during embargo periods.

When the standard READ check fails, the system falls back to CRIS security policies configured in *bitstream-access-modes.xml*. These policies only grant additional access — they never restrict access that users already have through standard ACLs.

With the default configuration, if a Researcher Page/Profile is linked as a *dc.contributor.author* on a publication, the author can download any restricted files on that item.

Configuration

To enable the feature you should set the following property in *dspace/config/dspace.cfg*:

```
core.authorization.bitstream.author.bypass-restrictions = true
```

In the *bitstream-access-modes.xml* you should configure:

1. bitstreamAccessModesMap — Who gets access, per entity type
    a.
```
<!--Maps entity type names to lists of access mode beans:-->
<entry key="Publication">
    <list>
        <ref bean="submitterEditItemMode" />   <!-- item submitter -->
        <ref bean="customEditItemMode" />       <!-- authors/editors via metadata -->
    </list>
</entry>
```
    b. Only items with entity type "Publication" are evaluated. Other entity types (Person, Dataset, etc.) can be added with their own entries.
2. allowedBundlesForBitstreamAccess — Which bundles are accessible
    a.
```
<entry key-ref="submitterEditItemMode">
    <list><value>ORIGINAL</value></list>
</entry>
<entry key-ref="customEditItemMode">
    <list><value>ORIGINAL</value></list>
</entry>
```
    b. Both modes are restricted to the ORIGINAL bundle only — so authors can download the actual files but not LICENSE, THUMBNAIL, or TEXT bundles via this bypass.
3. The Access Mode Beans

```
<bean id="submitterEditItemMode" class="org.dspace.content.edit.EditItemMode">
    <property name="security">
        <value type="...CrisSecurity">SUBMITTER</value>
    </property>
</bean>

<bean id="customEditItemMode" class="org.dspace.content.edit.EditItemMode">
    <property name="security">
        <value type="...CrisSecurity">CUSTOM</value>
    </property>
    <property name="items">
        <list>
            <value>dc.contributor.author</value>
            <value>dc.contributor.editor</value>
        </list>
    </property>
</bean>
```

Authentication Plugins

1 Stackable Authentication Method(s)
- 1.1 Authentication by Password
  - 1.1.1 Enabling Authentication by Password
  - 1.1.2 Configuring Authentication by Password
- 1.2 Open ID Connect (OIDC) Authentication
  - 1.2.1 Enabling OIDC Authentication
  - 1.2.2 Configuring OIDC Authentication
    - 1.2.2.1 Sample/Test OIDC Configuration
- 1.3 SAML (Security Assertion Markup Language) Authentication
  - 1.3.1 Enabling SAML Authentication
  - 1.3.2 Configuring SAML Authentication
    - 1.3.2.1 Sample/Test SAML Configuration
- 1.4 Shibboleth Authentication
  - 1.4.1 Enabling Shibboleth Authentication
  - 1.4.2 Configuring Shibboleth Authentication
    - 1.4.2.1 Apache "mod_shib" Configuration (required)
    - 1.4.2.2 Sample shibboleth2.xml Configuration
    - 1.4.2.3 Sample attribute-map.xml Configuration (for samltest.id)
    - 1.4.2.4 DSpace Shibboleth Configuration Options
- 1.5 LDAP Authentication
  - 1.5.1 Introduction to LDAP specific terminology
  - 1.5.2 Enabling LDAP Authentication
  - 1.5.3 Configuring LDAP Authentication
  - 1.5.4 Debugging LDAP connection and configuration
  - 1.5.5 Enabling Hierarchical LDAP Authentication
  - 1.5.6 Configuring Hierarchical LDAP Authentication
- 1.6 ORCID Authentication
  - 1.6.1 Enabling ORCID Authentication
- 1.7 IP Authentication
  - 1.7.1 Enabling IP Authentication
  - 1.7.2 Configuring IP Authentication
- 1.8 Example of a Custom Authentication Method

Stackable Authentication Method(s)

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

Configuration File:	**`[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 (class: `org.dspace.authenticate.PasswordAuthentication`) (DEFAULT)
- Open ID Connect (OIDC) Authentication (class: `org.dspace.authenticate.OidcAuthentication`)
- Shibboleth Authentication (class: `org.dspace.authenticate.ShibAuthentication`)
- LDAP Authentication (class: `org.dspace.authenticate.LDAPAuthentication`)
- ORCID Authentication (class: `org.dspace.authenticate.OrcidAuthentication`)
- IP Address based Authentication (class: `org.dspace.authenticate.IPAuthentication`)
- X.509 Certificate Authentication (class: `org.dspace.authenticate.X509Authentication`)

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

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 sending login information to the "/api/authn/login" endpoint of the REST API, in order to obtain a JSON Web Token. This JSON Web token must be sent on every later request which requires authentication.
- 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:	`user.registration`
Example Value:	`user.registration = false`
Informational Note:	This option allows you to disable all self-registration. When set to "false", no one will be able to register new accounts with your system. Default is "true".
Property:	`user.forgot-password`
Example Value:	`user.forgot-password` = `false`
Informational Note:	This option allows you to disable the forgot password link inside the login page. When set to "`false`", no one will be able to reset the password related to its account. Default is "`true`".
Property:	`authentication-password.domain.valid`
Example Value:	`authentication-password.domain.valid = @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. (This setting only works when user.registration=true)
Property:	`authentication-password.login.specialgroup`
Example Value:	`authentication-password.login.specialgroup = My DSpace Group`
Informational Note:	This option allows you to automatically add all password authenticated user sessions to a specific DSpace Group (the group must exist in DSpace) for the remainder of their logged in session.
Property:	`authentication-password.digestAlgorithm`
Example Value:	`authentication-password.digestAlgorithm = SHA-512`
Informational Note:	This option specifies the hashing algorithm to be used in converting plain-text passwords to more secure password digests. The example value is the default. You may select any digest algorithm available through java.security.MessageDigest on your system. At least MD2, MD5, SHA-1, SHA-256, SHA-384, and SHA-512 should be available, but you may have installed others. Most sites will not need to adjust this.
Property:	`authentication-password.regex-validation.pattern`

Example Value:	`authentication-password.regex-validation.pattern = ^.{8\,}$`
Informational Note:	This option specifies a regular expression which all new passwords MUST validate against. By default, DSpace just requires a new password to be 8 or more characters (see above example value). However, sites can modify this regex in order to require more robust passwords of all users. One example of a complex rule is: `authentication-password.regex-validation.pattern = ^(?=.*?[a-z])(?=.*?[A-Z])(?=\\S*?[0-9])(?=\\S*?[!?$@#$%^&+=]).{8\,15}$` This example requires all users to adopt a more complex password: • `(?=.*?[a-z])` - the password must contain at least one lowercase character • `(?=.*?[A-Z])` - the password must contain at least one uppercase character • `(?=\\S*?[0-9])` - the password must contain at least one numeric character • `(?=\\S*?[!?$@#$%^&+=])` - the password must contain at least one of the following special character: !?$@#$%^&+= • `.{8\,15}` - the password must be at least 8 and at most 15 characters long (NOTE: the "\," is required to escape the comma, which is a special character)

Open ID Connect (OIDC) Authentication

Open ID Connect (OIDC) Authentication is only available in DSpace 7.2 or above.

Enabling OIDC Authentication

To enable OIDC Authentication, you must ensure the `org.dspace.authenticate.OidcAuthentication` 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.OidcAuthentication` (*NOTE: This setting may be repeated to support multiple AuthenticationMethods*) (WARNING: it's easy to miss, the "camel case" for `OidcAuthentication` might catch you off guard. It's important to *not* use `OIDCAuthentication` in this line, because that class does not exist. *Case matters*.

Configuring OIDC Authentication

OpenID Connect is an identity layer on top of the OAuth 2.0 protocol. It allows Clients to verify the identity of the End-User based on the authentication performed by an Authorization Server, as well as to obtain basic profile information about the End-User in an interoperable and REST-like manner. There are many server implementations of OpenID Connect, including Keycloak and AWS Cognito.

Configuration File:	**`[dspace]/config/modules/authentication-oidc.cfg`**
Property:	`authentication-oidc.auth-server-url`
Example Value:	`authentication-oidc.auth-server-url = https://auth.example.com`
Informational Note:	(Optional) The root URL of the OpenID Connect server. This is optional, as it's only used to fill out each of the "-endpoint" configs below (see below). So, for some setups, it may be easier to configure the "-endpoint" configs directly INSTEAD OF the "auth-server-url" and "auth-server-realm"
Property:	`authentication-oidc.auth-server-realm`
Example Value:	`authentication-oidc.auth-server-realm = dspace-realm`
Informational Note:	(Optional) The realm to authenticate against on the OpenID Connect server. This is optional, as it's only used to fill out each of the "-endpoint" configs below (see below). So, for some setups, it may be easier to configure the "-endpoint" configs directly INSTEAD OF the "auth-server-url" and "auth-server-realm"
Property:	`authentication-oidc.token-endpoint`
Example Value:	`authentication-oidc.token-endpoint = ${authentication-oidc.auth-server-url}/auth/realms/${authentication-oidc.auth-server-realm}/protocol/openid-connect/token`

Informational Note:	(Required) The URL of the OIDC Token endpoint. This defaults to using the configured "auth-server-url" and "auth-server-realm" to determine the likely OIDC path for this endpoint (see example above for the default value). However, if that default path is incorrect, you may choose to hardcode the correct URL in this field.
Property:	`authentication-oidc.authorize-endpoint`
Example Value:	`authentication-oidc.authorize-endpoint = ${authentication-oidc.auth-server-url}/auth/realms/${authentication-oidc.auth-server-realm}/protocol/openid-connect/auth`
Informational Note:	(Required) The URL of the OIDC Authorize endpoint. This defaults to using the configured "auth-server-url" and "auth-server-realm" to determine the likely OIDC path for this endpoint (see example above for the default value). However, if that default path is incorrect, you may choose to hardcode the correct URL in this field.
Property:	`authentication-oidc.user-info-endpoint`
Example Value:	`authentication-oidc.user-info-endpoint = ${authentication-oidc.auth-server-url}/auth/realms/${authentication-oidc.auth-server-realm}/protocol/openid-connect/userinfo`
Informational Note:	(Required) The URL of the OIDC Userinfo endpoint. This defaults to using the configured "auth-server-url" and "auth-server-realm" to determine the likely OIDC path for this endpoint (see example above for the default value). However, if that default path is incorrect, you may choose to hardcode the correct URL in this field.
Property:	`authentication-oidc.client-id`
Example Value:	`authentication-oidc.client-id = our-dspace`
Informational Note:	(Required) The registered OIDC client id for our DSpace server's use. No default value.
Property:	`authentication-oidc.client-secret`
Example Value:	`authentication-oidc.client-secret = some-sort-of-hash`
Informational Note:	(Required) The registered OIDC client secret for our DSpace server's use. No default value.
Property:	`authentication-oidc.redirect-url`
Example Value:	`authentication-oidc.redirect-url = ${dspace.server.url}/api/authn/oidc`
Informational Note:	The URL users will be redirected to after a successful login. The example above is the default value, and it usually does not need to be updated.
Property:	`authentication-oidc.scopes`
Example Value:	`authentication-oidc.scopes = openid,email,profile`
Informational Note:	The scopes to request from the OIDC server. The example above is the default value
Property:	`authentication-oidc.can-self-register`
Example Value:	`authentication-oidc.can-self-register = true`
Informational Note:	Specify if the user can self register using OIDC (true|false). If not specified, true is assumed. If this is set to false, then only users with an existing EPerson in DSpace will be able to authenticate through OIDC. When set to true, an EPerson will be automatically created for each person who successfully authenticates through OIDC.
Property:	`authentication-oidc.user-info.email`
Example Value:	`authentication-oidc.user-info.email = email`
Informational Note:	Specify the attribute present in the user info json related to the user's email. The default value is "email"
Property:	`authentication-oidc.user-info.first-name`
Example Value:	`authentication-oidc.user-info.first-name = given_name`
Informational Note:	Specify the attribute present in the user info json related to the user's first/given name. The default value is "given_name"
Property:	`authentication-oidc.user-info.last-name`
Example Value:	`authentication-oidc.user-info.last-name = family_name`
Informational Note:	Specify the attribute present in the user info json related to the user's last/family name. The default value is "family_name"

Sample/Test OIDC Configuration

One way to easily test OIDC Authentication is to use the PhantAuth test site at https://www.phantauth.net/. This site allows you to create a random OIDC client & a random OIDC user to authenticate as. So, it can be used to verify that DSpace's OIDC authentication is working in your system, but obviously is only meant for development/testing purposes.

To configure DSpace to use PhantAuth for authentication just requires the following updates to your local.cfg:

local.cfg updates for PhantAuth

```
# Enable OIDC
plugin.sequence.org.dspace.authenticate.AuthenticationMethod = org.dspace.authenticate.OidcAuthentication

# Settings for OIDC authentication
# Based on instructions at https://www.phantauth.net/doc/integration
authentication-oidc.authorize-endpoint = https://phantauth.net/auth/authorize
authentication-oidc.token-endpoint = https://phantauth.net/auth/token
authentication-oidc.user-info-endpoint = https://phantauth.net/auth/userinfo

# Obtain a random client-id and client-secret via https://phantauth.net/client
# Find the "client_id" and "client_secret" returned, and place those values in these next two configs.
authentication-oidc.client-id =
authentication-oidc.client-secret =

# Because PhantAuth uses random users, you MUST ensure self registration is enabled
# (This is the default setting though, which is why it's commented out)
# authentication-oidc.can-self-register = true
```

SAML (Security Assertion Markup Language) Authentication

SAML Authentication is only available in DSpace 9.0 or above.

Enabling SAML Authentication

To enable SAML Authentication, you must ensure the `org.dspace.authenticate.SamlAuthentication` 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.SamlAuthentication` (*NOTE: This setting may be repeated to support multiple AuthenticationMethods*) (WARNING: it's easy to miss, the "camel case" for `SamlAuthentication` might catch you off guard. It's important to *not* use `SAML Authentication` in this line, because that class does not exist. *Case matters*.

Configuring SAML Authentication

SAML (Security Assertion Markup Language) is an XML based standard for transferring identity information between entities, in particular an identity provider (IdP) and a service provider (SP). The former performs authentication and transmits user identity data to the latter, the system the user is attempting to access. The SAML Authenication plugin configures DSpace to operate as a SAML service provider interacting with an external identity provider. There are many SAML IdP server implementations, including open source and enterprise options from major cloud and SAAS providers.

Note: SAML configuration can be simple, but in some cases may require a lot of configuration. In addition to the settings summarized here if you are having trouble setting up a SAML intregration it is recommended you review the documentation provided in the source code. It may also be helpful to refer to the feature submission details: https://github.com/DSpace/DSpace/pull/9470

Configuration File:	**`[dspace]/config/modules/authentication-saml.cfg`**
Property:	`authentication-saml.relying-party-id`
Example Value:	`authentication-saml.relying-party-id = auth0`

Informational Note:	(Required) The ID of the SAML relying party we should use for authentication. This is an internal identifier used to reference configuration in saml-relying-party.cfg.
Property:	`authentication-saml.relying-party-url`
Example Value:	`authentication-saml.relying-party-url = ${dspace.server.url}/saml2`
Informational Note:	The base URL of all SAML relying party endpoints.
Property:	`authentication-saml.authenticate-endpoint`
Example Value:	`authentication-saml.authenticate-endpoint = ${authentication-saml.relying-party-url}/authenticate/${authentication-saml.relying-party-id}`
Informational Note:	The URL of the authenticate endpoint for the SAML relying party.
Property:	`authentication-saml.autoregister`
Example Value:	`authentication-saml.autoregister = true`
Informational Note:	Should we allow new users to be registered automatically?
Property:	`authentication-saml.attribute.relying-party-id`
Example Value:	`authentication-saml.attribute.relying-party-id = org.dspace.saml.RELYING_PARTY_ID`
Informational Note:	The request attribute that contains the ID of the SAML relying party that authenticated the user.
Property:	`authentication-saml.attribute.name-id`
Example Value:	`authentication-saml.attribute.name-id = org.dspace.saml.NAME_ID`
Informational Note:	The request attribute that contains the user's SAML name ID.
Property:	`authentication-saml.attribute.email`
Example Value:	`authentication-saml.attribute.email = org.dspace.saml.EMAIL`
Informational Note:	The request attribute that contains the user's email.
Property:	`authentication-saml.attribute.first-name`
Example Value:	`authentication-saml.attribute.first-name = org.dspace.saml.GIVEN_NAME`
Informational Note:	The request attribute that contains the user's first name.
Property:	`authentication-saml.attribute.last-name`
Example Value:	`authentication-saml.attribute.last-name = org.dspace.saml.SURNAME`
Informational Note:	The request attribute that contains the user's last name.
Property:	`authentication-saml.eperson.metadata`
Example Value:	`authentication-saml.eperson.metadata = org.dspace.saml.PHONE => phone`
Informational Note:	Additional attribute mappings. Multiple attributes may be stored for each user. The left side is the request attribute, and the right side is the ePerson metadata field to map the attribute to.
Property:	`authentication-saml.eperson.metadata.autocreate`
Example Value:	`authentication-saml.eperson.metadata.autocreate = false`
Informational Note:	If the ePerson metadata field is not found, should it be created automatically?

For the `saml-relying-party.cfg` settings below replace **`[RELYING-PARTY-ID]`** with the value of `authentication-saml.relying-party-id` from your `authentication-saml.cfg`

Configuration File:	**`[dspace]/config/modules/saml-relying-party.cfg`**
Property:	`saml-relying-party.[RELYING-PARTY-ID].asserting-party.metadata-uri`
Example Value:	`saml-relying-party.auth0.asserting-party.metadata-uri = https://dev-vynkcnqhac3c0s10.us.auth0.com/samlp/metadata/Vn8jWX0iFHtepmXi7rjZa9h5M1kqXNWY`
Informational Note:	The URI where metadata for the asserting party (IdP) can be retrieved. Ideally, this is the only asserting party configuration that is needed, because all of the other asserting party configuration properties will be set automatically from the metadata.
Property:	`saml-relying-party.[RELYING-PARTY-ID].asserting-party.entity-id`
Example Value:	`saml-relying-party.auth0.asserting-party.entity-id = urn:dev-vynkcnqhac3c0s10.us.auth0.com`
Informational Note:	(Optional) Only set manually if you cannot get metadata from `saml-relying-party.[RELYING-PARTY-ID].asserting-party.metadata-uri`
Property:	`saml-relying-party.[RELYING-PARTY-ID].asserting-party.single-sign-on.url`
Example Value:	`saml-relying-party.auth0.asserting-party.single-sign-on.url = https://dev-vynkcnqhac3c0s10.us.auth0.com/samlp/Vn8jWX0iFHtepmXi7rjZa9h5M1kqXNWY`
Informational Note:	(Optional) Only set manually if you cannot get metadata from `saml-relying-party.[RELYING-PARTY-ID].asserting-party.metadata-uri`
Property:	`saml-relying-party.[RELYING-PARTY-ID].asserting-party.single-sign-on.binding`
Example Value:	`saml-relying-party.auth0.asserting-party.single-sign-on.binding = POST`
Informational Note:	(Optional) Only set manually if you cannot get metadata from `saml-relying-party.[RELYING-PARTY-ID].asserting-party.metadata-uri`
Property:	`saml-relying-party.[RELYING-PARTY-ID].asserting-party.single-sign-on.sign-request`
Example Value:	`saml-relying-party.auth0.asserting-party.single-sign-on.sign-request = false`
Informational Note:	(Optional) Only set manually if you cannot get metadata from `saml-relying-party.[RELYING-PARTY-ID].asserting-party.metadata-uri`
Property:	`saml-relying-party.[RELYING-PARTY-ID].asserting-party.single-logout.url`
Example Value:	`saml-relying-party.auth0.asserting-party.single-logout.url = https://dev-vynkcnqhac3c0s10.us.auth0.com/samlp/Vn8jWX0iFHtepmXi7rjZa9h5M1kqXNWY/logout`
Informational Note:	(Optional) Only set manually if you cannot get metadata from `saml-relying-party.[RELYING-PARTY-ID].asserting-party.metadata-uri`
Property:	`saml-relying-party.[RELYING-PARTY-ID].asserting-party.single-logout.binding`
Example Value:	`saml-relying-party.auth0.asserting-party.single-logout.binding = POST`
Informational Note:	(Optional) Only set manually if you cannot get metadata from `saml-relying-party.[RELYING-PARTY-ID].asserting-party.metadata-uri`
Property:	`saml-relying-party.[RELYING-PARTY-ID].asserting-party.single-logout.response-url`
Example Value:	`saml-relying-party.auth0.asserting-party.single-logout.response-url =`
Informational Note:	**Note that single logout information can be configured for the asserting party, but SAML single logout is not currently supported by DSpace.**
Property:	`saml-relying-party.[RELYING-PARTY-ID].asserting-party.verification.credentials.0.certificate-location`
Example Value:	`saml-relying-party.auth0.asserting-party.verification.credentials.0.certificate-location = file://opt/dspace/cert/auth0-ap-certificate.crt`
Informational Note:	(Optional) If a verification certificate is configured, the certificate-location should contain the URL of an X509 certificate. This can be a file, http(s), or classpath URL.
Property:	`saml-relying-party.[RELYING-PARTY-ID].signing.credentials.0.private-key-location`

Example Value:	`saml-relying-party.auth0.signing.credentials.0.private-key-location = file:///opt/dspace/secrets/rp-private.key`
Informational Note:	Can be set to sign and/or encrypt/decrypt communications.
Property:	`saml-relying-party.[RELYING-PARTY-ID].signing.credentials.0.certificate-location`
Example Value:	`saml-relying-party.auth0.signing.credentials.0.certificate-location = file:///opt/dspace/cert/rp-certificate.crt`
Informational Note:	Can be set to sign and/or encrypt/decrypt communications.
Property:	`saml-relying-party.[RELYING-PARTY-ID].decryption.credentials.0.private-key-location`
Example Value:	`saml-relying-party.auth0.decryption.credentials.0.private-key-location = file:///opt/dspace/secrets/rp-private.key`
Informational Note:	Can be set to sign and/or encrypt/decrypt communications.
Property:	`saml-relying-party.[RELYING-PARTY-ID].decryption.credentials.0.private-key-location`
Example Value:	`saml-relying-party.auth0.decryption.credentials.0.private-key-location = file:///opt/dspace/secrets/rp-private.key`
Informational Note:	Can be set to sign and/or encrypt/decrypt communications.
Property:	`saml-relying-party.[RELYING-PARTY-ID].attributes`

Example Value:	`saml-relying-party.auth0.attributes = \`
	`  http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress => org.dspace.saml.EMAIL, \`
	`  http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname => org.dspace.saml.GIVEN_NAME, \`
	`  http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname => org.dspace.saml.SURNAME`

Informational Note:	Mapping of SAML assertion attributes to request attributes. The left side is the name of an attribute in the assertion received from the IdP. The right side is the name of the request attribute to map the SAML attribute to, before forwarding the request to the authentication endpoint. A mapping to `org.dspace.saml.EMAIL` must be supplied in order for the authentication endpoint to associate a SAML user to a DSpace user. Mappings for `org.dspace.saml.GIVEN_NAME` and `org.dspace.saml.SURNAME` are required for new users to be registered automatically.

Sample/Test SAML Configuration

One way to test SAML authentication is to sign up for a free account with a service like Auth0. They provide documentation on how to create a SAML Identity Provider. However, you can use any SAML IdP and there are many available.

To configure DSpace to use SAML for authentication (with auth0 as an example) requires this minimal configuration:

local.cfg updates for SAML

```
# Enable SAML
plugin.sequence.org.dspace.authenticate.AuthenticationMethod = org.dspace.authenticate.SAMLAuthentication
authentication-saml.relying-party-id = auth0 # refer to our IdP as "auth0" in our configuration

# Settings for SAML authentication (note the use of "auth0" in the replying-party configuation properties)
saml-relying-party.auth0.asserting-party.metadata-uri = https://dev-vynkcnqhac3c0s10.us.auth0.com/samlp/metadata/Vn8jWX0iFHtepmXi7rjZa9h5M1kqXNWY

saml-relying-party.auth0.attributes = \
  http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress => org.dspace.saml.EMAIL, \
  http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname => org.dspace.saml.GIVEN_NAME, \
  http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname => org.dspace.saml.SURNAME

# Add the IdP url to the list of allowed origins
rest.cors.allowed-origins = ${dspace.ui.url},https://dev-vynkcnqhac3c0s10.us.auth0.com
```

In general when setting up a SAML integration the flow will go something like:

1. The administrator of the IdP provides a URL to the metadata for the IdP.

2. Configure a new relying party in DSpace, setting `asserting-party.metadata-uri` to the IdP's metadata URL.
3. If encryption and/or signing are desired/required by the IdP, configure private keys and certificates into the relying party.
4. Ask the administrator of the IdP to release the user's email, first name, and last name as attributes, and ask what the names of those attributes will be. Configure the relying party's `attributes` property, to map those attribute names to `org.dspace.saml.EMAIL`, `org.dspace.saml.GIVEN_NAME`, and `org.dspace.saml.SURNAME`.
5. Start DSpace, and provide the relying party's metadata URL (`/server/saml2/service-provider-metadata/{id}`) to the administrator of the IdP, so they can use it to configure their side. Alternatively, retrieve the metadata, and provide it to the administrator of the IdP through another channel.
6. The administrator of the IdP says that the configuration is complete. Testing can commence.

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` (*NOTE: This setting may be repeated to support multiple AuthenticationMethods*)

Configuring Shibboleth Authentication

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

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

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

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

Note about Shibboleth Active vs Lazy Sessions:

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

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

Apache "mod_shib" Configuration (required)

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

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

- In Debian based environments, "mod_shib" tends to be in a package named something like "libapache2-mod-shib2"
- The Shibboleth setting "ShibUseHeaders" is no longer required to be set to "On", as DSpace will correctly utilize attributes instead of headers.

    - When "ShibUseHeaders" is set to "Off" (which is recommended in the mod_shib documentation), proper configuration of Apache to pass attributes to Tomcat (via either mod_jk or mod_proxy) can be a bit tricky, SWITCH has some great documentation on exactly what you need to do. We will eventually paraphrase/summarize this documentation here, but for now, the SWITCH page will have to do.
- When initially setting up Apache & mod_shib, https://samltest.id/ provides a great testing ground for your configurations. This site provides a sample/demo Shibboleth IdP (as well as a sample Shibboleth SP) which you can test against. It acts as a "sandbox" to get your configurations working properly, before you point DSpace at your production Shibboleth IdP.
- You also may wish to review the Shibboleth setup in our "dspace-shibboleth" Docker setup which the development team uses for testing (and it uses https://samltest.id as the IdP). It may provide you with good examples/hints on getting everything setup. However, keep in mind this code has not been tested in Production scenarios.

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

```
#### SAMPLE MOD_SHIB CONFIGURATION FOR APACHE2 (it may require local modifications based on your Apache setup)
####
# While this sample VirtualHost is for HTTPS requests (recommended for Shibboleth, obviously),
# you may also need/want to create one for HTTP (*:80)
<VirtualHost *:443>
   ...
   # PLEASE NOTE: We have omitted many Apache settings (ServerName, LogLevel, SSLCertificateFile, etc)
   # which you may need/want to add to your VirtualHost

   # As long as Shibboleth module is installed, enable all Shibboleth/mod_shib related settings
   <IfModule mod_shib>
       # Shibboleth recommends turning on UseCanonicalName
       # See "Prepping Apache" in https://wiki.shibboleth.net/confluence/display/SHIB2/NativeSPApacheConfig
       UseCanonicalName On

       # Most DSpace instances will want to use Shibboleth "Lazy Session", which ensures that users
       # can access DSpace without first authenticating via Shibboleth.
       # This section turns on Shibboleth "Lazy Session". Also ensures that once they have authenticated
       # (by accessing /Shibboleth.sso/Login path), then their Shib session is kept alive
       <Location />
         AuthType shibboleth
         ShibRequireSession Off
         require shibboleth
         # If your "shibboleth2.xml" file specifies an <ApplicationOverride> setting for your
         # DSpace Service Provider, then you may need to tell Apache which "id" to redirect Shib requests to.
         # Just uncomment this and change the value "my-dspace-id" to the associated @id attribute value.
         #ShibRequestSetting applicationId my-dspace-id
       </Location>

       # If a user attempts to access the DSpace shibboleth endpoint, force them to authenticate via Shib.
       <Location "/server/api/authn/shibboleth">
         Order deny,allow
         Allow from all
         AuthType shibboleth
         ShibRequireSession On
         # Please note that setting ShibUseHeaders to "On" is a potential security risk.
         # You may wish to set it to "Off". See the mod_shib docs for details about this setting:
         # https://wiki.shibboleth.net/confluence/display/SHIB2/NativeSPApacheConfig#NativeSPApacheConfig-
AuthConfigOptions
         # Here's a good guide to configuring Apache + Tomcat when this setting is "Off":
         # https://www.switch.ch/de/aai/support/serviceproviders/sp-access-rules.html#javaapplications
         ShibUseHeaders On
         Require shibboleth
       </Location>

       # If a user attempts to access the DSpace login endpoint, ensure Shibboleth is supported but other auth
methods can be too.
       <Location "/server/api/authn/login">
          Order deny,allow
          Allow from all
          AuthType shibboleth
          # For DSpace, this is required to be off otherwise the available auth methods will be not visible
          ShibRequireSession Off
          # Please note that setting ShibUseHeaders to "On" is a potential security risk.
          # You may wish to set it to "Off". See the mod_shib docs for details about this setting:
```

```
          # https://wiki.shibboleth.net/confluence/display/SHIB2/NativeSPApacheConfig#NativeSPApacheConfig-
AuthConfigOptions
          # Here's a good guide to configuring Apache + Tomcat when this setting is "Off":
          # https://www.switch.ch/de/aai/support/serviceproviders/sp-access-rules.html#javaapplications
          ShibUseHeaders On
       </Location>

       # Ensure /Shibboleth.sso path (in Apache) can be accessed
       # By default it may be inaccessible if your Apache security is tight.
       <Location "/Shibboleth.sso">
         Order deny,allow
         Allow from all
         # Also ensure Shibboleth/mod_shib responds to this path
         SetHandler shib
       </Location>

       # Finally, you may need to ensure requests to /Shibboleth.sso are NOT redirected
       # to Tomcat (as they need to be handled by mod_shib instead).
       # NOTE: THIS SETTING IS LIKELY ONLY NEEDED IF YOU ARE USING mod_proxy TO REDIRECT
       # ALL REQUESTS TO TOMCAT (e.g. ProxyPass /server ajp://localhost:8009/server)
       ProxyPass /Shibboleth.sso !
   </IfModule>

   ...


   # You will likely need Proxy settings to ensure Apache is proxying requests to Tomcat for the DSpace REST API
   # The below is just an example of proxying for REST API only. It requires installing & enabling "mod_proxy"
and "mod_proxy_ajp"
   ## Proxy / Forwarding Settings ##
   <Proxy *>
      AddDefaultCharset Off
      Order allow,deny
      Allow from all
   </Proxy>

   # Proxy all requests to /server to Tomcat via AJP
   ProxyPass /server ajp://localhost:8009/server
   ProxyPassReverse /server ajp://localhost:8009/server

   # Optionally, also proxy Angular UI (if on same server). This requires "mod_proxy_http"
   #ProxyPass / http://localhost:4000/
   #ProxyPassReverse / http://localhost:4000/
</VirtualHost>
```

Sample shibboleth2.xml Configuration

In addition, here's a sample "ApplicationOverride" configuration for "shibboleth2.xml". This particular "ApplicationOverride" is configured to use the Test IdP provided by https://samltest.id/ and is just meant as an example. In order to enable it for testing purposes, you **must** specify `ShibRequestSetting applicationId samltest` in your Apach `mod_shib` configuration (see above). An additional, more detailed example is provided in our "dspace-shibboleth" Docker configurations at https://github.com/DSpace/DSpace/blob/main/dspace/src/main/docker/dspace-shibboleth/shibboleth2.xml

```
       <!-- *** Sample Shibboleth Settings for  https://samltest.id/  ***     -->
       <!-- This provides a simple sample of how you could configure           -->
       <!-- shibboleth2.xml for DSpace sites.                                  -->
       <!-- TO ENABLE: You'd need to specify "applicationId" as "samltest" in  -->
       <!-- your mod_shib settings, e.g.                                       -->
       <!-- <Location />                                                       -->
       <!--     ...                                                            -->
       <!--     ShibRequestSetting applicationId samltest                      -->
       <!-- </Location>                                                        -->
       <ApplicationOverride id="samltest" entityID="http://[mydspace.edu]/shibboleth" REMOTE_USER="eppn
persistent-id targeted-id">

           <!-- We'll use a TEST IdP, hosted by the awesome https://samltest.id/ testing service. -->
           <!-- See also: https://wiki.shibboleth.net/confluence/display/SHIB2/NativeSPServiceSSO -->
           <!-- DSPACE 7 requires Shibboleth to use "SameSite=None" property for its Cookies -->
           <Sessions lifetime="28800" timeout="3600" checkAddress="false" relayState="ss:mem" handlerSSL="
```

```
true" cookieProps="; path=/; SameSite=None; secure; HttpOnly">
               <SSO entityID="https://samltest.id/saml/idp">
                 SAML2 SAML1
               </SSO>
            </Sessions>

            <!-- Loads and trusts a metadata file that describes the IdP and how to communicate with it. -->
            <!-- By default, metadata is retrieved from the TEST IdP at https://samltest.id/ -->
            <!-- and is cached in a local file named "samltest-metadata.xml". -->
            <!-- See also: https://wiki.shibboleth.net/confluence/display/SHIB2/NativeSPMetadataProvider -->
            <MetadataProvider type="XML" uri="https://samltest.id/saml/idp"
                          backingFilePath="samltest-metadata.xml" reloadInterval="180000"/>
        </ApplicationOverride>
```

Sample attribute-map.xml Configuration (for samltest.id)

In order to use the above example for https://samltest.id/, you may also need to modify your attribute-map.xml to support their attributes. Again, a more complete example is in our "dspace-shibboleth" Docker configurations at https://github.com/DSpace/DSpace/blob/main/dspace/src/main/docker/dspace-shibboleth/attribute-map.xml

```
<Attributes xmlns="urn:mace:shibboleth:2.0:attribute-map" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <!-- Custom Attributes specific to samltest.id -->
    <Attribute name="urn:oid:0.9.2342.19200300.100.1.1" id="uid"/>
    <Attribute name="urn:oid:0.9.2342.19200300.100.1.3" id="mail"/>
    <Attribute name="urn:oid:2.5.4.4" id="sn"/>
    <Attribute name="urn:oid:2.16.840.1.113730.3.1.241" id="displayName"/>
    <Attribute name="urn:oid:2.5.4.20" id="telephoneNumber"/>
    <Attribute name="urn:oid:2.5.4.42" id="givenName"/>
    <Attribute name="https://samltest.id/attributes/role" id="role"/>

    ...


    <!-- In addition to the attribute mapping, DSpace expects the following Shibboleth Headers to be set:
           - SHIB-NETID
           - SHIB-MAIL
           - SHIB-GIVENNAME
           - SHIB-SURNAME
         These are set by mapping the respective IdP attribute (left hand side) to the header attribute (right
hand side).
    -->
    <Attribute name="urn:oid:0.9.2342.19200300.100.1.1" id="SHIB-NETID"/>
    <Attribute name="urn:mace:dir:attribute-def:uid" id="SHIB-NETID"/>

    <Attribute name="urn:oid:0.9.2342.19200300.100.1.3" id="SHIB-MAIL"/>
    <Attribute name="urn:mace:dir:attribute-def:mail" id="SHIB-MAIL"/>

    <Attribute name="urn:oid:2.5.4.42" id="SHIB-GIVENNAME"/>
    <Attribute name="urn:mace:dir:attribute-def:givenName" id="SHIB-GIVENNAME"/>

    <Attribute name="urn:oid:2.5.4.4" id="SHIB-SURNAME"/>
    <Attribute name="urn:mace:dir:attribute-def:sn" id="SHIB-SURNAME"/>


</Attributes>
```

DSpace Shibboleth Configuration Options

Authentication Methods:

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

*1) NetID from Shibboleth Header (**best**)*

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

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

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

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

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

Identity Scheme Migration Strategies:

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

EPerson Metadata:

One of the primary benefits of using Shibboleth based authentication is receiving additional attributes about users such as their names, telephone numbers, and possibly their academic department or graduation semester if desired. DSpace treats the first and last name attributes differently because they (along with email address) are the three pieces of minimal information required to create a new user account. For both first and last name supply direct mappings to the Shibboleth headers. In 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 -shibboleth.role.<role-name>" which is a comma separated list of DSpace groups.

Having issues getting Safari working?

In addition to the below settings, you may need to ensure your Shibboleth IdP is *trusted* by the DSpace backend by adding it to your `rest.cors.allowed-origins` configuration. *This is required for Safari web browsers to work with DSpace's Shibboleth plugin.*

For example, if your IdP is https://samltest.id/, then you need to append that URL to the comma-separated list of "allowed-origins" like:

```
rest.cors.allowed-origins = ${dspace.ui.url}, https://samltest.id
```

More information on this configuration can be found in the REST API documentation.

Configuration File:	**`[dspace]/config/modules/authentication-shibboleth.cfg`**
Property:	`authentication-shibboleth.lazysession`
Example Value:	`authentication-shibboleth.lazysession = true`
Informational Note:	Whether to use lazy sessions or active sessions. For more DSpace instances, you will likely want to use lazy sessions. Active sessions will force **every** user to authenticate via Shibboleth before they can access your DSpace (essentially resulting in a "dark archive").
Property:	`authentication-shibboleth.lazysession.loginurl`
Example Value:	authentication-shibboleth.`lazysession.loginurl = /Shibboleth.sso/Login`
Informational Note:	The url to start a shibboleth session (only for lazy sessions). Generally this setting will be "/Shibboleth.sso/Login"
	`authentication-shibboleth.lazysession.secure`

Property:	
Example Value:	`authentication-shibboleth.lazysession.secure = true`
Informational Note:	Force HTTPS when authenticating (only for lazy sessions). Generally this is recommended to be "true".
Property:	`authentication-shibboleth.netid-header`
Example Value:	`authentication-shibboleth.netid-header = SHIB-NETID`
Informational Note:	The HTTP header where shibboleth will supply a user's NetID. This HTTP header should be specified as an Attribute within your Shibboleth "attribute-map.xml" configuration file.
Property:	`authentication-shibboleth.email-header`
Example Value:	`authentication-shibboleth.email-header = SHIB-MAIL`
Informational Note:	The HTTP header where the shibboleth will supply a user's email address. This HTTP header should be specified as an Attribute within your Shibboleth "attribute-map.xml" configuration file.
Property:	`authentication-shibboleth.email-use-tomcat-remote-user`
Example Value:	`authentication-shibboleth.email-use-tomcat-remote-user = false`
Informational Note:	Used when a netid or email headers are not available should Shibboleth authentication fall back to using Tomcat's remote user feature? Generally this is not recommended. See the "Authentication Methods" section above.
Property:	`authentication-shibboleth.reconvert.attributes`
Example Value	`authentication-shibboleth.reconvert.attributes = false`
Informational Note:	Shibboleth attributes are by default UTF-8 encoded. Some servlet container automatically converts the attributes from ISO-8859-1 (latin-1) to UTF-8. As the attributes already were UTF-8 encoded it may be necessary to reconvert them. If you set this property true, DSpace converts all shibboleth attributes retrieved from the servlet container from UTF-8 to ISO-8859-1 and uses the result as if it were UTF-8. This procedure restores the shibboleth attributes if the servlet container wrongly converted them from ISO-8859-1 to UTF-8. Set this true, if you notice character encoding problems within shibboleth attributes.
Property:	`authentication-shibboleth.autoregister`
Example Value:	`authentication-shibboleth.autoregister = true`
Informational Note:	Should we allow new users to be registered automatically?
Property:	`authentication-shibboleth.sword.compatibility`
Example Value:	`authentication-shibboleth.sword.compatibility = false`
Informational Note:	SWORD compatibility will allow this authentication method to work when using SWORD. SWORD 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. WARNING: If you enable this option while ALSO having "PasswordAuthentication" enabled, then you should ensure that "PasswordAuthentication" is listed prior to "ShibAuthentication" in your authentication.cfg file. Otherwise, ShibAuthentication will be used to authenticate all of your users INSTEAD OF PasswordAuthentication.
Property:	`authentication-shibboleth.firstname-header`
Example Value:	`authentication-shibboleth.firstname-header = SHIB_GIVENNAME`
Informational Note:	The HTTP header where the shibboleth will supply a user's given name. This HTTP header should be specified as an Attribute within your Shibboleth "attribute-map.xml" configuration file.
Property:	`authentication-shibboleth.lastname-header`
Example Value:	`authentication-shibboleth.lastname-header = SHIB_SN`
Informational Note:	The HTTP header where the shibboleth will supply a user's surname. This HTTP header should be specified as an Attribute within your Shibboleth "attribute-map.xml" configuration file.
Property:	`authentication-shibboleth.eperson.metadata`
Example Value:	

	`authentication-shibboleth.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-shibboleth.eperson.metadata.autocreate`
Example Value:	`authentication-shibboleth.eperson.metadata.autocreate = true`
Informational Note:	If the eperson metadata field is not found, should it be automatically created?
Property:	`authentication-shibboleth.role-header`
Example Value:	`authentication-shibboleth.role-header = SHIB_SCOPED_AFFILIATION`
Informational Note:	The Shibboleth header holding the user's Shibboleth roles. See the "Role-based Groups" section above for more info.
Property:	`authentication-shibboleth.role-header.ignore-scope`
Example Value:	`authentication-shibboleth.role-header.ignore-scope = true`
Informational Note:	Whether to ignore roles' scopes (everything after the @ sign for scoped attributes)
Property:	`authentication-shibboleth.role-header.ignore-value`
Example Value:	`authentication-shibboleth.role-header.ignore-value = false`
Informational Note:	Whether to ignore roles' values (everything before the @ sign for scoped attributes)
Property:	`authentication-shibboleth.role.[affiliation-attribute]`

Example Value:	`authentication-shibboleth.role.faculty = Faculty, Member`
	`authentication-shibboleth.role.staff = Staff, Member`
	`authentication-shibboleth.role.student = Students, Member`

Informational Note:	Mapping of affiliation values to DSpace groups. See the "Role-based Groups" section above for more info.
Property:	`authentication-shibboleth.default-roles`
Example Value:	`authentication-shibboleth.default-roles = GenericUser`
Informational Note:	These roles are assumed if no roles were sent by Shibboleth or there was no header with name matching the value of `authentication-shibboleth.role_header`. May be repeated to provide multiple default roles.

LDAP Authentication

Introduction to LDAP specific terminology

If you are unfamiliar with LDAP, the following introduction to some of its terminology might come in handy:

https://stackoverflow.com/questions/18756688/what-are-cn-ou-dc-in-an-ldap-search

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 below for more information about how to do this.

Ensure required commas are escaped in LDAP configuration

NOTE: As of DSpace 6, commas (,) are now a special character in the Configuration system. As some LDAP configuration may contain commas, you must be careful to escape any required commas by adding a backslash (\) before each comma, e.g. "\,". The configuration reference for authentication-ldap.cfg has been updated below with additional examples.

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

Configuration File:	**`[dspace]/config/modules/authentication-ldap.cfg`**
Property:	`authentication-ldap.enable`
Example Value:	`authentication-ldap.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:	`authentication-ldap.autoregister`
Example Value:	`authentication-ldap.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:	`authentication-ldap.provider_url`
Example Value:	`authentication-ldap.provider_url = ldap://ldap.myu.edu/o=myu.edu\,ou=mydept`
Informational Note:	This is the url to your institution's LDAP server. You may or may not need the /o=myu.edu part at the end. Your server may also require the ldaps:// protocol. (This field has no default value) **NOTE:** As of DSpace 6, commas (,) are now a special character in the Configuration system. Therefore, be careful to escape any required commas in this configuration by adding a backslash (\) before each comma, e.g. "\,"
Property:	`authentication-ldap.starttls`
Example Value:	`authentication-ldap.starttls = false`
Informational Note:	Should we issue StartTLS after establishing TCP connection in order to initiate an encrypted connection? Note: This (TLS) is different from LDAPS: • TLS is a tunnel for plain LDAP and is typically recognized on the same port (standard LDAP port: 389) • LDAPS is a separate protocol, deprecated in favor of the standard TLS method. (standard LDAPS port: 636)
Property:	`authentication-ldap.id_field`
Example Value:	`authentication-ldap.id_field = uid`
Explanation:	This is the unique identifier field in the LDAP directory where the username is stored. (This field has no default value)
Property:	`authentication-ldap.object_context`
Example Value:	`authentication-ldap.object_context = ou=people\,o=myu.edu`
Informational Note:	

	This is the LDAP object context to use when authenticating the user. By default, DSpace will use this value to create the user's DN in order to attempt to authenticate them. It is appended to the *id_field* and username. For example `uid=username\,ou=people\,o=myu.edu`. You will need to modify this to match your LDAP configuration. (This field has no default value) If your users do NOT all exist under a single "object_context" in LDAP, then you should ignore this setting and INSTEAD use the Hierarchical LDAP Authentication settings below (especially see "`search.user`" or "`search.anonymous`") **NOTE:** As of DSpace 6, commas (,) are now a special character in the Configuration system. Therefore, be careful to escape any required commas in this configuration by adding a backslash (\) before each comma, e.g. "\,"
Property:	`authentication-ldap.search_context`
Example Value:	`authentication-ldap.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=true`, when a user authenticates without an EPerson object we search the LDAP directory to get their name (`id_field`) and email address (`email_field`) 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. (This field has no default value, and it MUST be specified when either `search.anonymous=true` or `search.user` is specified) **NOTE:** As of DSpace 6, commas (,) are now a special character in the Configuration system. Therefore, be careful to escape any required commas in this configuration by adding a backslash (\) before each comma, e.g. "\,"
Property:	`authentication-ldap.email_field`
Example Value:	`authentication-ldap.email_field = mail`
Informational Note:	This is the LDAP object field where the user's email address is stored. "mail" is the most common for LDAP servers. (This field has no default value) If the "email_field" is unspecified, or the user has no email address in LDAP, his/her username (id_field value) will be saved as the email in DSpace (or appended to `netid_email_domain`, when specified)
Property:	`authentication-ldap.netid_email_domain`
Example Value:	`authentication-ldap.netid_email_domain = @example.com`
Informational Note:	If your LDAP server does not hold an email address for a user (i.e. no `email_field`), you can use the following field to specify your email domain. This value is appended to the netid (`id_field`) in order to make an email address (which is then stored in the DSpace EPerson). For example, a netid of 'user' and `netid_email_domain` as `@example.com` would set the email of the user to be user@example.com *Please note:* this field will only be used if "`email_field`" is unspecified OR the user in question has no email address stored in LDAP. If both "`email_field`" and "`netid_email_domain`" are unspecified, then the "`id_field`" will be used as the email address.
Property:	`authentication-ldap.surname_field`
Example Value:	`authentication-ldap.surname_field = sn`
Informational Note:	This is the LDAP object field where the user's last name is stored. "`sn`" is the most common for LDAP servers. If the field is not found the field will be left blank in the new eperson object. (This field has no default value)
Property:	`authentication-ldap.givenname_field`
Example Value:	`authentication-ldap.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. (This field has no default value)
Property:	`authentication-ldap.phone_field`
Example Value:	`authentication-ldap.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. (This field has no default value)
Property:	`authentication-ldap.login.specialgroup`
Example Value:	`authentication-ldap.login.specialgroup = group-name`
Informational Note:	If specified, all user sessions successfully logged in via LDAP will automatically become members of this DSpace Group (for the remainder of their current, logged in session). This DSpace Group **must** already exist (it will not be automatically created). This is useful if you want a DSpace Group made up of all internal authenticated users. This DSpace Group can then be used to bestow special permissions on any users who have authenticated via LDAP (e.g. you could allow anyone authenticated via LDAP to view special, on campus only collections or similar)

Property:	`login.groupmap.*`

Example Value:	`authentication-ldap.login.groupmap.1 = ou=Students:ALL_STUDENTS`
	`authentication-ldap.login.groupmap.2 = ou=Employees:ALL_EMPLOYEES`
	`authentication-ldap.login.groupmap.3 = ou=Faculty:ALL_FACULTY`

Informational Note:	The left part of the value (before the ":") must correspond to a portion of a user's DN (unless "`login.group.attribute`" is specified..please see below). The right part of the value corresponds to the name of an existing DSpace group.
	
	For example, if the authenticated user's DN in LDAP is in the following form:
	
	`cn=jdoe,OU=Students,OU=Users,dc=example,dc=edu`
	
	that user would get assigned to the ALL_STUDENTS DSpace group for the remainder of their current session.
	
	However, if that same user later graduates and is employed by the university, their DN in LDAP may change to:
	
	`cn=jdoe,OU=Employees,OU=Users,dc=example,dc=edu`
	
	Upon logging into DSpace after that DN change, the authenticated user would now be assigned to the ALL_EMPLOYEES DSpace group for the remainder of their current session.
	
	Note: This option can be used independently from the login.specialgroup option, which will put all LDAP users into a single DSpace group. Both options may be used together.

Property:	`authentication-ldap.login.groupmap.attribute`
Example Value:	`authentication-ldap.login.groupmap.attribute = group`
Informational Note:	The value of the "`authentication-ldap.login.groupmap.attribute`" should specify the name of a single LDAP attribute. If this property is uncommented, it changes the meaning of the left part of "`authentication-ldap.login.groupmap.*`" (see above) as follows: • If the authenticated user has this LDAP attribute, look up the value of this LDAP attribute in the left part (before the ":") of the `authentication-ldap.login.groupmap.*` value • If that LDAP value is found in any "`authentication-ldap.login.groupmap.*`" field, assign this authenticated user to the DSpace Group specified by the right part (after the ":") of the `authentication-ldap.login.groupmap.*` value. For example: • `authentication-ldap.login.groupmap.attribute = group` • `authentication-ldap.login.groupmap.1 = mathematics:Mathematics_Group` The above would ensure that any authenticated users where their LDAP "group" attribute equals "mathematics" would be added to the DSpace Group named "Mathematics_Group" for the remainder of their current session. However, if that same user logged in later with a new LDAP "group" value of "computer science", he/she would no longer be a member of the "Mathematics_Group" in DSpace.

Debugging LDAP connection and configuration

As every LDAP is different, configuring your DSpace to communicate with your LDAP can sometimes be a challenge. We recommend using third-party LDAP tools to test your LDAP connection / username / password, and perform sample searches to better understand what information is being returned from your local LDAP. This will help ensure that LDAP configuration goes more smoothly.

One example of such an LDAP tool is the `ldapsearch` commandline tool available in most Linux operating systems (e.g. in Debian / Ubuntu it's available in the "ldap-utils" package). Below are some example ldapsearch commands that can be used to determine (and/or debug) specific configurations in your `authentication-ldap.cfg`. In the below examples, we've used the names of specific DSpace configurations as placeholders (in square brackets).

```
# Basic anonymous connection (for VERBOSE, add -v)
ldapsearch -x -H [provider_url]

# Debug a connection error (add -d-1)
# If you are connecting to an LDAPS URL and see connection errors (e.g. "peer cert untrusted or revoked")
# then see below note about "SSL Connection Errors"
ldapsearch -x -H [provider_url] -d-1


# Attempt to connect to [provider_url] as [search.user] (will prompt for search.user's password)
# This doesn't actually perform a query, just ensures that authentication is working
# NOTE: "search.user" is USUALLY either the full user DN (e.g. "cn=dspaceadmin,ou=people,o=myu.edu")
# or "DOMAIN\USERNAME" (e.g. "MYU\DSpaceUser"). The latter is more likely with Windows Active Directory
ldapsearch -x -H [provider_url] -D [search.user] -W


# Attempt to list the first 100 users in a given [search_context], returning the "cn", "mail" and "sn" fields
```

```
for each
ldapsearch -x -H [provider_url] -D [search.user] -W -b [search_context] -z 100 cn mail sn


# Attempt to find the first 100 users whose [id_field] starts with the letter "t", returning the [id_field],
"cn", "mail" and "sn" fields for each
ldapsearch -x -H [provider_url] -D [search.user] -W -b [search_context] -z 100 -s sub "([id_field]=t*)"
[id_field] cn mail sn
```

SSL Connection Errors: If you are using ldapsearch with an LDAPS connection (secure connection), you may receive "peer cert untrusted or revoked" errors if the LDAP SSL certificate is self-signed. You can temporarily tell LDAP to accept any security certificate by setting `TLS_REQCERT allow` in your ldapsearch's `ldap.conf` file. *Be sure to remove this setting however after you are done testing!*

```
# FOR TESTING ONLY! This setting disables the check for a valid LDAP Server security certificate,
# which is considered a security issue for production LDAP setups. Setting this to "allow" tells
# the LDAP client to accept any security certificates that it cannot verify or validate.
TLS_REQCERT allow
```

More information on this SSL workaround can be found at:

- http://www.bind9.net/manual/openldap/2.3/tls.html
- http://muzso.hu/2012/03/29/how-to-configure-ssl-aka.-ldaps-for-libnss-ldap-auth-client-config-in-ubuntu

Enabling Hierarchical LDAP Authentication

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

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

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

Configuring Hierarchical LDAP Authentication

Hierarchical LDAP Authentication shares all the above standard LDAP configurations, but has some additional settings.

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

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

Informational Note:	This is the search scope value for the LDAP search during autoregistering (`autoregister=true`). This will depend on your LDAP server setup, and is only really necessary if your users are spread out across a hierarchical tree on your LDAP server. This value must be one of the following integers corresponding to the following values:
	`object scope : 0`
	`one level scope : 1`
	`subtree scope : 2`
	
	Please note that "`search_context`" in the LDAP configurations must also be specified.

Property:	`authentication-ldap.search.anonymous`
Example Value:	`authentication-ldap.search.anonymous = true`
Informational Note:	If true, DSpace will anonymously search LDAP (in the "`search_context`") for the DN of the user trying to login to DSpace. This setting is "false" by default. By default, DSpace will either use "`search.user`" to authenticate for the LDAP search (if search.user is specified), or will use the "`object_context`" value to create the user's DN.
Property:	`authentication-ldap.search.user` `authentication-ldap.search.password`
Example Value:	

	`authentication-ldap.search.user = cn=admin\,ou=people\,o=myu.edu` `authentication-ldap.search.password = password`
Informational Note:	The full DN and password of a user allowed to connect to the LDAP server and search (in the "`search_context`") for the DN of the user trying to login. By default, if unspecified, DSpace will either search LDAP anonymously for the user's DN (when `search.anonymous=true`), or will use the "`object_context`" value to create the user's DN. **NOTE:** As of DSpace 6, commas (,) are now a special character in the Configuration system. Therefore, be careful to escape any required commas in this configuration by adding a backslash (\) before each comma, e.g. "\,"

ORCID Authentication

Enabling ORCID Authentication

Enabling ORCID Authentication **requires** also enabling Configurable Entities and Researcher Profiles

To enable ORCID Authentication, see the documentation for enabling the ORCID Integration. You do not need to enable ORCID synchronization, but you currently must enable Researcher Profiles and Configurable Entities.

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:

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

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 session 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.

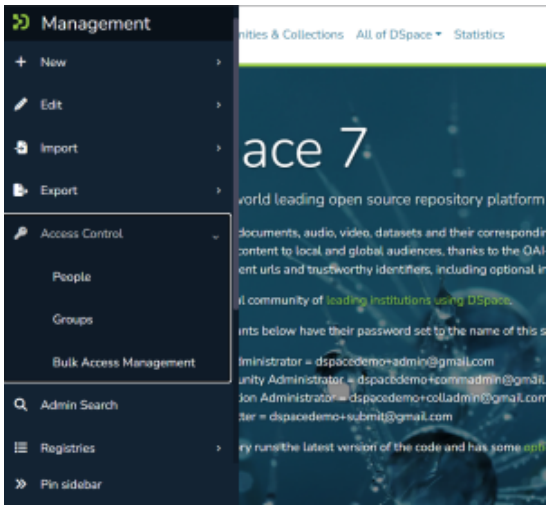
You can create your own custom authentication method and add it to the stack. Use the most similar existing method as a model, e.g. `org.dspace.authenticate.PasswordAuthentication` for an "explicit" method (with credentials entered interactively) or `org.dspace.authenticate.X509Authentication` for an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

Bulk Access Management

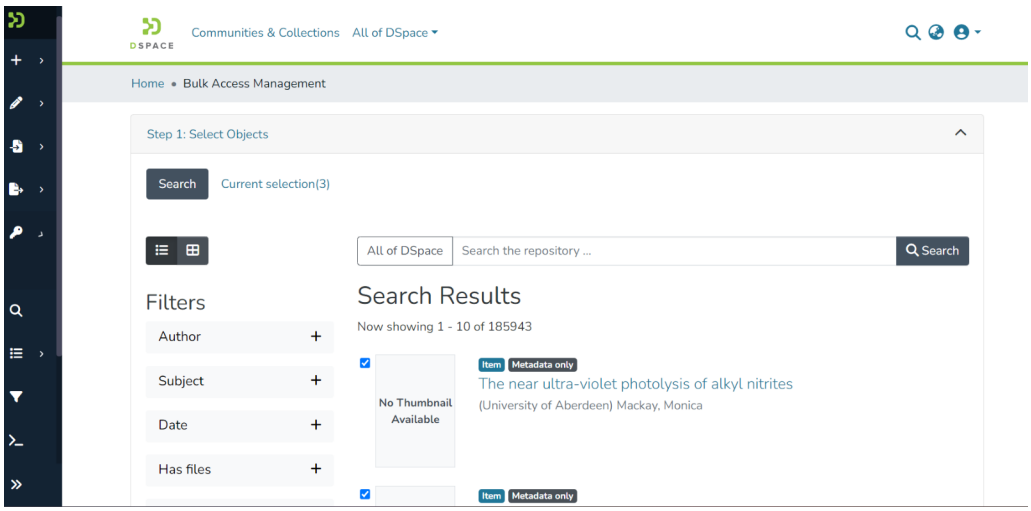
The Bulk Access Management will let administrators edit massively the access conditions of Metadata and Bitstreams on selected objects.

Usage:

- When logged as Administrator, it is possible to change the access condition to Metadata and Bitstreams on items by following the path: "Management > Access Control > Bulk Access Management".
- Community or Collection Administrators may also use this tool from the "Access Control" tab of the "Edit Community", "Edit Collection" or "Edit Item" page. In this scope, the tool will only perform changes within the selected Community/Collection/Item.



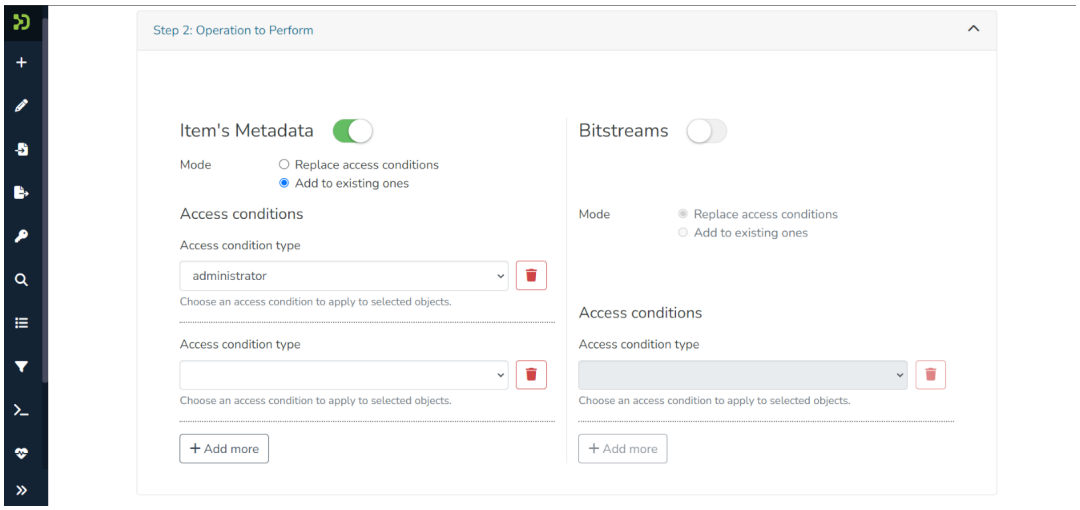
In the first box (Step 1) should be selected the objects on which access conditions will be changed.



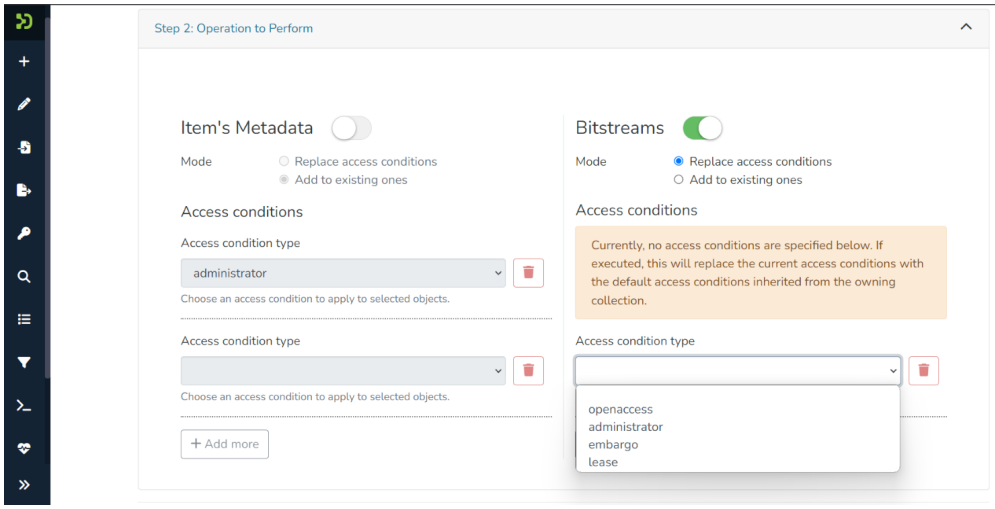
In the second box (Step 2) the administrator will choose if change the access conditions on Metadata, or on Bitstreams or both.

The actions that can be performed are:

- Replacing the existent access conditions
- Add new conditions to the existing ones



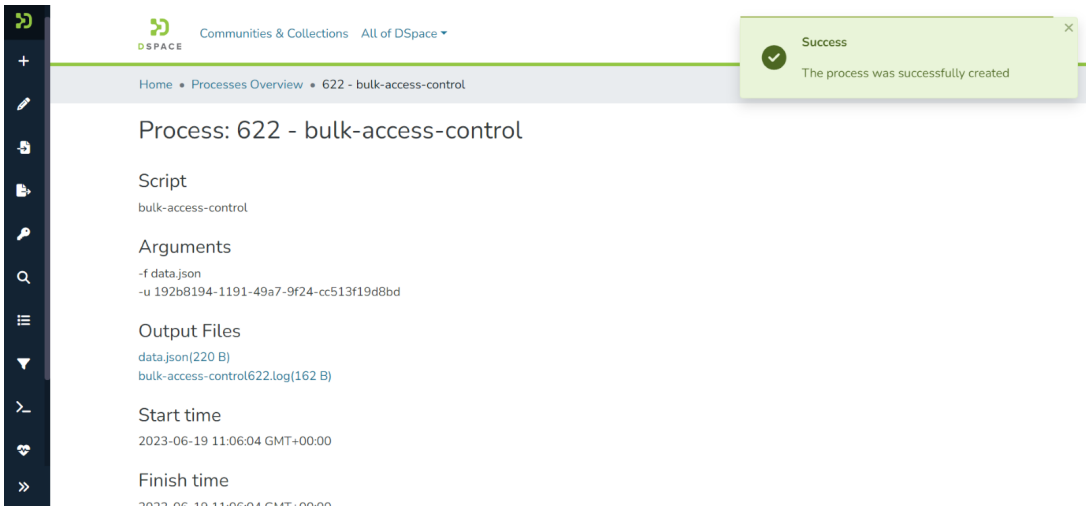
If there is no previous access condition defined, a warning box will appear.



The access conditions that can be chosen by default are:

- Openaccess
- Administrator
- Embargo
- Lease

When done, click Execute. The process will start. If it succeeded, the process page and a success message will show.



At the moment, DSpace supports a single feature configuration defined by the defaultBulkAccessConditionConfiguration, in which are specified the access conditions available for the Item's and the Bitstream's Metadata.

The access conditions listed in the dropdown menu are set by default as Openaccess, Administrator, Embargo, and Lease.

They can be edited by changing and adding options in the code:

```
<bean id="defaultBulkAccessConditionConfiguration"
        class="org.dspace.app.bulkaccesscontrol.model.BulkAccessConditionConfiguration">
      <property name="name" value="default"/>
      <property name="itemAccessConditionOptions">
          <list>
              <ref bean="openAccess"/>
              <ref bean="administrator"/>
              <ref bean="embargoed" />
              <ref bean="lease"/>
          </list>
      </property>
      <property name="bitstreamAccessConditionOptions">
          <list>
              <ref bean="openAccess"/>
              <ref bean="administrator"/>
              <ref bean="embargoed" />
              <ref bean="lease"/>
          </list>
      </property>
  </bean>

  <bean id="bulkAccessConditionConfigurationService"
        class="org.dspace.app.bulkaccesscontrol.service.BulkAccessConditionConfigurationService">
      <property name="bulkAccessConditionConfigurations">
          <list>
              <ref bean="defaultBulkAccessConditionConfiguration"/>
          </list>
      </property>
  </bean>
```

Please refer to this page for further information: https://github.com/DSpace/DSpace/blob/main/dspace/config/spring/api/access-conditions.xml#L78-L106

Embargo

1 What is an Embargo?
2 DSpace Embargo Functionality
    2.1 Managing Embargoes on existing Items
3 Configuring and using Embargo in DSpace Submission User Interface
    3.1 Enabling Item-level Embargo
    3.2 Configuring Embargo / Access Restriction options
    3.3 Private/Public (or Non-Discoverable/Discoverable) Item
    3.4 Pre-3.0 Embargo Migration Routine
4 Technical Specifications
    4.1 Introduction
    4.2 ResourcePolicy
    4.3 Item
    4.4 Item.inheritCollectionDefaultPolicies(Collection c)
    4.5 AuthorizeService
    4.6 Withdraw Item
    4.7 Reinstate Item
    4.8 Pre-DSpace 3.0 Embargo Compatibility
5 Creating Embargoes via Metadata
    5.1 Introduction
    5.2 Setting Embargo terms via metadata
        5.2.1 Terms assignment
        5.2.2 Terms interpretation/imposition
        5.2.3 Embargo period
    5.3 Configuration of metadata fields
    5.4 Operation
    5.5 Extending embargo functionality
        5.5.1 Setter
        5.5.2 Lifter

What is an Embargo?

An embargo is a temporary access restriction placed on metadata or bitstreams (i.e. files). Its scope or duration may vary, but the fact that it eventually expires is what distinguishes it from other content restrictions. For example, it is not unusual for content destined for DSpace to come with permanent restrictions on use or access based on license-driven or other IP-based requirements that limit access to institutionally affiliated users. Restrictions such as these are imposed and managed using standard administrative tools in DSpace, typically by attaching specific access policies (aka "resource policies") to Items, Collections, Bitstreams, etc.

Embargo functionality was originally introduced as part of DSpace 1.6, enabling embargoes on the level of items that applied to all bitstreams included in the item. Since DSpace 3.0, this functionality has been extended to the Submission User Interface, enabling embargoes on the level of individual bitstreams.

DSpace Embargo Functionality

Embargoes can be applied per *item (including metadata)* and per *bitstream (i.e. file)*. The *item* level embargo will be the default for every *bitstream*, although it could be customized at *bitstream* level.

When an embargo is set on either an item level or a bitstream level, a new ResourcePolicy (i.e. access policy) is added to the corresponding Item or Bitstream. **This ResourcePolicy will automatically control the lifting of the embargo (when the embargo date passes).** An embargo lift date is generally stored as the "start date" of such a policy. Essentially, this means that the access rights defined in the policy do not get applied until *after that date passes* (and prior to that date, the access rights will default to Admin only).

The scheduled, manual "embargo-lifter" commands (used prior to DSpace 3) are no longer necessary and not recommended to run.

Managing Embargoes on existing Items

Administrators are able to change the lift date of any embargo by editing the authorization policy (ResourcePolicy) on the object. These authorization policies can be managed from the Edit Item screen by clicking on "Authorizations".

- To add an embargo, edit the appropriate policy and set a "start date". To add an full Item embargo (including metadata), edit the Item policy. To embargo individual bitstreams, edit the appropriate Bitstream policy.
- To remove an embargo, edit the appropriate policy, and clear out the "start date".
- To change an embargo, edit the appropriate policy, and change the "start date" to a new date.

Changes to the embargo should take effect immediately. However, as Administrators have full access to embargoed items, you may need to log out first. After logging out, you will be subject to the embargo.

Configuring and using Embargo in DSpace Submission User Interface

Available in DSpace 7.2 and above

In DSpace 7.2 and above, both Item-level embargoes and bitstream (file) level embargoes are supported in the Submission user interface.
In DSpace 7.1 and 7.0, the Submission user interface only supported embargoes on specified bitstreams (files). However, item-level embargoes could be added after submissions were accepted using the "Manage Embargoes on existing Items" approach described above.

Enabling Item-level Embargo

While Bitstream-level embargoes are enabled by default, Item-level embargoes currently are not. However, enabling them is easy. Simply update your `item-submission.xml` to include this tag in your <submission-process>:

```
<submission-process name="traditional">
   ...

   <!-- This step enables embargoes and other access restrictions at the Item level -->
   <step id="itemAccessConditions"/>
</submission-process>
```

After making this update, you will need to restart your backend (REST API) for the changes to take effect.

Configuring Embargo / Access Restriction options

Starting in DSpace 7, embargo (and lease) settings are configurable via a Spring Bean configuration file `[dspace]/config/spring/api/access-conditions.xml`

For detailed information on configuring your Embargo options (and other related options like lease or restrict to a particular group of users), see the Submission User Interface documentation. Specifically these two sections:

- For Bitstream embargo / access options, see the section on "*Configuring the File Upload step*" of the Submission User Interface
- For Item embargo / access options, see the section on "*Configuring the Item Access Conditions step*" of the Submission User Interface

Private/Public (or Non-Discoverable/Discoverable) Item

It is also possible to adjust the Private/Public (or Non-Discoverable/Discoverable) state of an item after it has been archived in the repository. This can be achieved from either the "Admin Search" (/admin/search), or from the "Status" tab under "Edit Item".

Private (or non-Discoverable) items are not retrievable through the DSpace search, browse or Discovery indexes. However, they are accessible via a direct link. It is possible to create a publicly accessible, non-discoverable item...in which case it can only be shared via a direct link. But, once anyone has that link, it is available anonymously.

Therefore, an "Admin Search" option is provided, which allows you to search across all items, including private or withdrawn items. You can also filter your results to display only private items.

Pre-3.0 Embargo Migration Routine

If you have just upgraded from a DSpace 1.x.x version, any embargoes that are currently "in effect" will need to be migrated into ResourcePolicies. Prior to 3.0, embargoes in DSpace were managed entirely in metadata fields (and required running a scheduled "embargo-lifter" command). However, DSpace now stores all embargo information directly on ResourcePolicies (i.e. "access policies"). These ResourcePolicies automatically "lift" an embargo after the embargo date passes.

In order to migrate old embargoes into ResourcePolicies, a migration routine has been developed. **Please note that this migration routine should only need to be run ONCE** (immediately after an upgrade from 1.x.x to a more recent version of DSpace). After that point, any newly defined embargoes will automatically be stored on ResourcePolicies.

To execute it, run the following command:

```
[dspace]/bin/dspace migrate-embargo -a
```

Technical Specifications

Introduction

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

ResourcePolicy

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

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

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

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

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

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

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

Item

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

- isDiscoverable

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

Item.inheritCollectionDefaultPolicies(Collection c)

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

AuthorizeService

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

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

Withdraw Item

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

Reinstate Item

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

Pre-DSpace 3.0 Embargo Compatibility

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

Creating Embargoes via Metadata

Introduction

Prior to DSpace 3.0, all DSpace embargoes were stored as metadata. While embargoes are no longer stored permanently in metadata fields (they are now stored on ResourcePolicies, i.e. access policies), embargoes *can still be initialized via metadata fields.*

This ability to create/initialize embargoes via metadata is extremely powerful if you wish to submit embargoed content via electronic means (such as Importing Items via Simple Archive Format, SWORDv1, SWORDv2, etc).

Setting Embargo terms via metadata

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

"2020-09-12" - an absolute date (i.e. the date embargo will be lifted)
"6 months" - a time relative to when the item is accessioned
"forever" - an indefinite, or open-ended embargo
"local only until 2015" - both a time and an exception (public has no access until 2015, local users OK immediately)
"Nature Publishing Group standard" - look-up to a policy somewhere (typically 6 months)

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

Terms assignment

The first step in placing an embargo on an item is to attach (assign) "terms" to it. If these terms are missing, no embargo will be imposed. As we will see below, terms are carried in a configurable DSpace metadata field, so assigning terms just means assigning a value to a metadata field. This can be done in a web submission user interface form, in a SWORD deposit package, a batch import, etc. - anywhere metadata is passed to DSpace. The terms are not immediately acted upon, and may be revised, corrected, removed, etc, up until the next stage of the life-cycle. Thus a submitter could enter one value, and a collection editor replace it, and only the last value will be used. Since metadata fields are multivalued, theoretically there can be multiple terms values, but in the default implementation only one is recognized.

Terms interpretation/imposition

In DSpace terminology, when an Item has exited the last of any workflow steps (or if none have been defined for it), it is said to be "installed" into the repository. At this precise time, the interpretation of the terms occurs, and a computed "lift date" is assigned, and recorded as part of the ResourcePolicy (aka policy) of the Item. Once the lift date has been assigned to the ResourcePolicy, the metadata field which defined the embargo is **cleared**. From that point forward, all embargo information is controlled/defined by the ResourcePolicy.

It is important to understand that this interpretation happens only once, (just like the installation). Therefore, **updating/changing an embargo cannot be done via metadata fields.** Instead, all embargo updates must be made to the ResourcePolicies themselves (e.g. ResourcePolicies can be managed from the Admin UI in the Edit Item screens).

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 the embargo date passes. Once the embargo date passes, the policy restrictions are automatically lifted. An embargo lift date is generally stored as the "start date" of a policy. Essentially, this means that the policy does not get applied until after that date passes (and prior to that date, the object defaults to Admin only access).

Administrators are able to change the lift date of the embargo by editing the policy (ResourcePolicy). These policies can be managed from the Edit Item screens.

Configuration of metadata fields

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

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

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

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

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

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

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

which you may change to suit linguistic or other preference.

You are free to use existing metadata fields, or create new fields. If you choose the latter, you must understand that the embargo system does **not** create or configure these fields: i.e. you must follow all the standard documented procedures for actually creating them (i.e. adding them to the metadata registry, or to display templates, etc) - this does not happen automatically. Likewise, if you want the field for "terms" to appear in submission screens and workflows, you must follow the documented procedure for configurable submission (basically, this means adding the field to submission-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. Use a local metadata schema. Breaking compliance with the standard Dublin Core in the default metadata registry can create a problem for the portability of data to/from of your repository.
2. If using existing metadata fields, avoid any that are automatically managed by DSpace. For example, fields like "date.issued" or "date.accessioned" are normally automatically assigned, and thus must not be recruited for embargo use.
3. Do not place the field for "lift date" in submission screens. This can potentially confuse submitters because they may feel that they can directly assign values to it. As noted in the life-cycle above, this is erroneous: the lift date gets assigned by the embargo system based on the terms. Any pre-existing value will be over-written. But see next recommendation for an exception.
4. As the life-cycle discussion above makes clear, after the terms are applied, that field is no longer actionable in the embargo system. Conversely, the "lift date" field is not actionable **until** the application. Thus you may want to consider configuring both the "terms" and "lift date" to use the same metadata field. In this way, during workflow you would see only the terms, and after item installation, only the lift date. If you wish the metadata to retain the terms for any reason, use 2 distinct fields instead.

Operation

After the fields defined for terms and lift date have been assigned in dspace.cfg, and created and configured wherever they will be used, you can begin to embargo items simply by entering data (dates, if using the default setter) in the terms field. They will automatically be embargoed as they exit workflow, and that the computed lift date will be stored on the ResourcePolicy

Extending embargo functionality

The embargo system supplies a default "interpreter/imposition" class (the "Setter") .

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

DEPRECATED: The Lifter is no longer used in the DSpace API, and is not recommended to utilize. Embargo lift dates are now stored on ResourcePolicies and, as such, are "lifted" automatically when the embargo date passes. Manually running a "lifter" may bypass this automatic functionality and result in unexpected results.

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

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

Pre-3.0 Embargo Lifter Commands

DEPRECATED - Not recommended to use

The old "embargo-lifter" command is no longer necessary to run. All Embargoes in DSpace are now stored on ResourcePolicies and are lifted automatically after the lift date passed. See Embargo documentation for more information.

Continuing to run the "embargo-lifter" is not recommended and this feature will be removed entirely in a future DSpace release.

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

Command used:	`[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
```

Managing User Accounts

- From the browser
- From the command line
  - The user command
    - To create a new user account:
    - To list accounts:
    - To modify an account:
    - To delete an account:
  - The Groomer
    - Find accounts with unsalted passwords
    - Find (and perhaps delete) disused accounts
- Cryptographic properties

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

From the browser

- Login as an Administrator
- Sidemenu "Access Control" "People"
- Browse or search for the account you wish to modify or delete.

To modify user permissions / group memberships:

- Login as an Administrator
- Sidemenu "Access Control" "Groups"
- Edit the Group
- Search for the EPerson & add/remove them from that group.

To debug issues for a specific user, it's possible to login as (or "impersonate") that user account

- On the backend, first you MUST enable the "assumelogin" feature. This feature is disabled by default. Update this setting in your local.cfg or dspace.cfg

```
# Required to use "Impersonate EPerson" feature
# When enabled, a full Administrator can impersonate any other non-Administrative user
webui.user.assumelogin = true
```

- Then, from the user interface, login as an Administrator
- Sidemenu "Access Control" "People"
- Browse or search for the account you wish to login as
- Edit that User, and click the "Impersonate EPerson" button.
- You are now logged in as that user. You'll see an Impersonate icon/button in the header.
- You are able to temporarily manage any activities as that user.
- Once your are done, click the "Stop impersonating EPerson".
- Optionally, you may wish to disable this feature again in your local.cfg by setting the above configuration to "false" or commenting it out.

From the command line

The `user` command

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

To create a new user account:

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

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

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

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

To list accounts:

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

This simply lists some characteristics of each EPerson.

short	long	meaning
-L	--list	required

To modify an account:

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

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

To delete an account:

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

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

The Groomer

This tool inspects all user accounts for several conditions.

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

Find accounts with unsalted passwords

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

Discovering accounts with unsalted password hashes

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

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

Find (and perhaps delete) disused accounts

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

Discovering disused accounts

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

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

You can also have the tool delete matching accounts:

Deleting disused accounts

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

Cryptographic properties

The cryptographic properties used for generating the salted hashes, to ensure encryption at rest for user passwords, can be found and adjusted in:

https://github.com/DSpace/DSpace/blob/main/dspace-api/src/main/java/org/dspace/eperson/PasswordHash.java

Email Subscriptions

Introduction
Adding new subscriptions
Managing your subscriptions
Enable sending out emails

Introduction

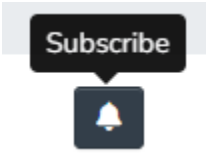
This feature is available in 7.5 or later.

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

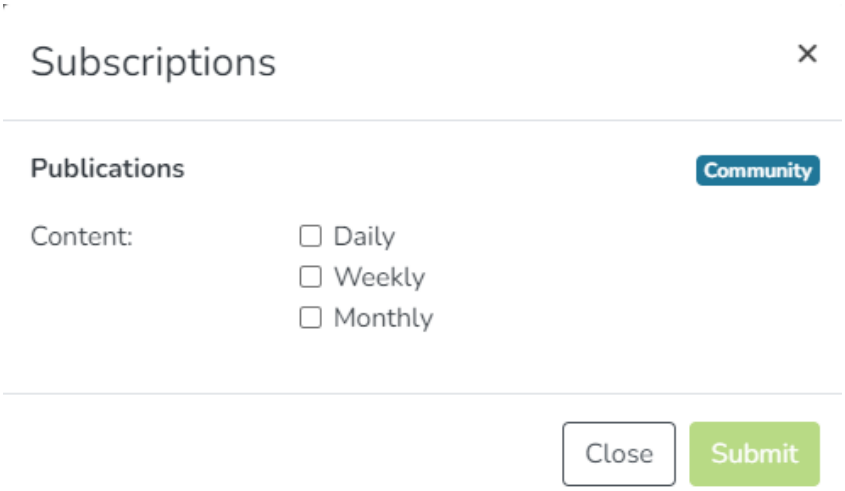
Adding new subscriptions

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

In the User interface, browse to the Community or Collection you wish to subscribe to, and click on the Subscribe button.



After clicking that button, you'll see a popup window which allows you to select the frequency of subscription you'd like.

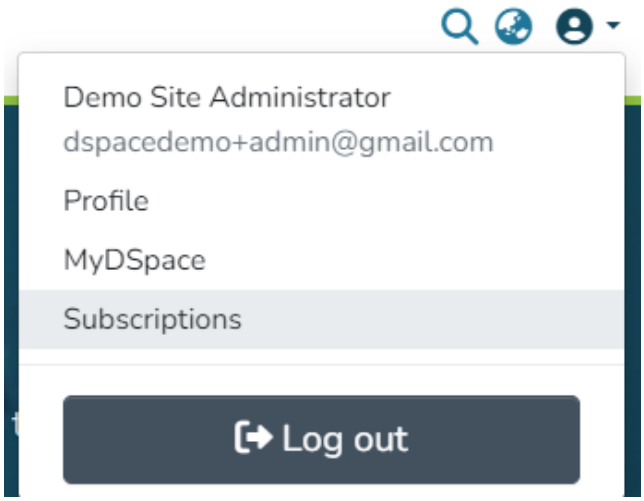


Currently, three frequency options are available:

- Daily: Receive a daily email of Items under the Community/Collection which have been updated in the last day.
- Weekly: Receive a weekly email of Items under the Community/Collection which have been updated in the last week.
- Monthly: Receive a monthly email of Items under the Community/Collection which have been updated in the last month.

Managing your subscriptions

To manage your subscriptions, visit your "Subscriptions" page under your user profile:



From this page, you are able to see all your current Community/Collection subscriptions. You can choose to edit or delete any in the list.

Enable sending out emails

Run "subscription-send" to enable

⚠ NOTE: Until you enable the "subscription-send" script, users will not receive the email updates for their subscriptions. It is HIGHLY RECOMMENDED to enable this script via Scheduled Tasks via Cron. See sample settings on that page.

To send out the subscription emails you MUST invoke the **subscription-send** script from the DSpace command-line or Processes UI. It is advised to setup this script as a Scheduled Tasks via Cron. See sample settings on that page.

```
# Example of running subscription-send to send out all "Daily" subscription emails
./dspace subscription-send -f D
```

This script requires the "-f" (--frequency) parameter with a value of "D" (Daily), "W" (Weekly), or "M" (Monthly). Keep in mind, you will want to schedule it to run on a Daily, Weekly and Monthly basis to send the appropriate emails.

Request a Copy

Introduction
Requesting a copy using the User Interface
(Optional) Requesting a copy with Help Desk workflow
Email templates
Configuration parameters
Protecting with Captcha
Selecting Request a Copy strategy
    Configure who gets request via a metadata field
    Configure all requests to go to a helpdesk email
    Configure all requests to go to the administrators of a Collection
    Combine multiple strategies
    Configure who gets request via a metadata field defined within the primary bitstream metadata submission form

Introduction

Supported in 7.1 or above

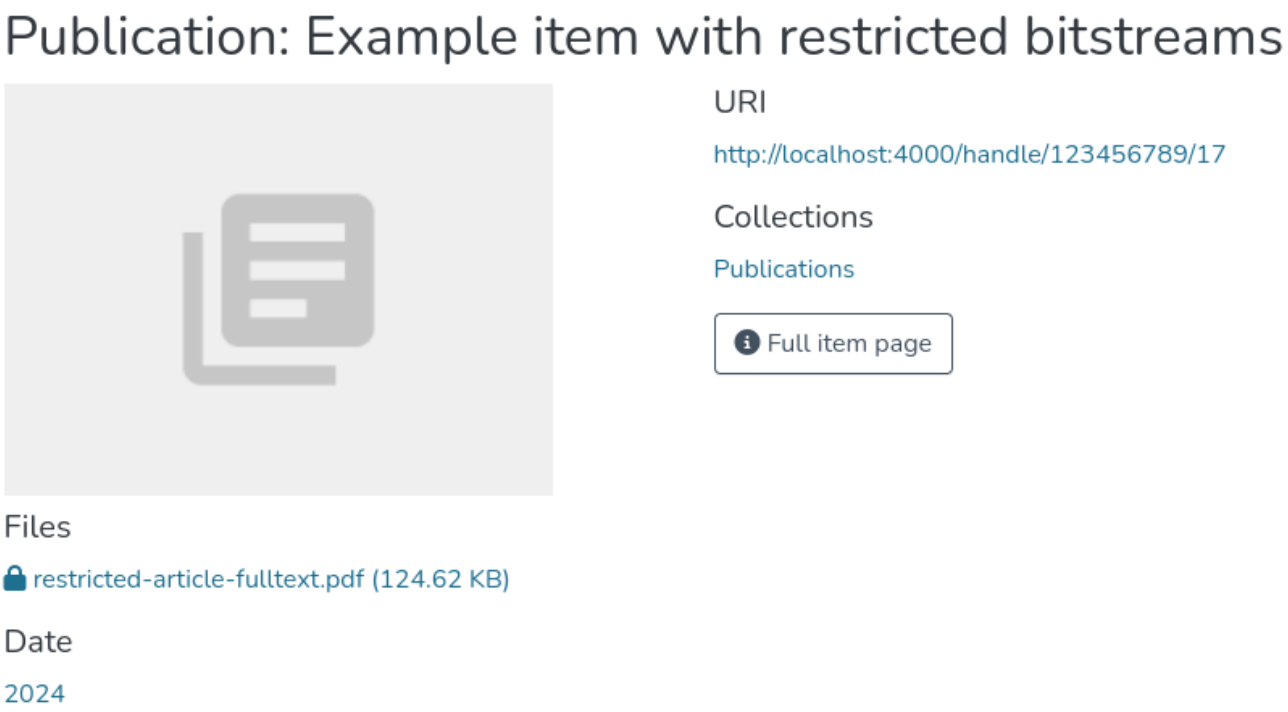
Request a Copy was not available in DSpace 7.0. It was restored in DSpace 7.1. See DSpace Release 7.0 Status

Starting with DSpace 9.0, it is possible to send requested item/bitstream access as a secure link instead of attaching to an email.

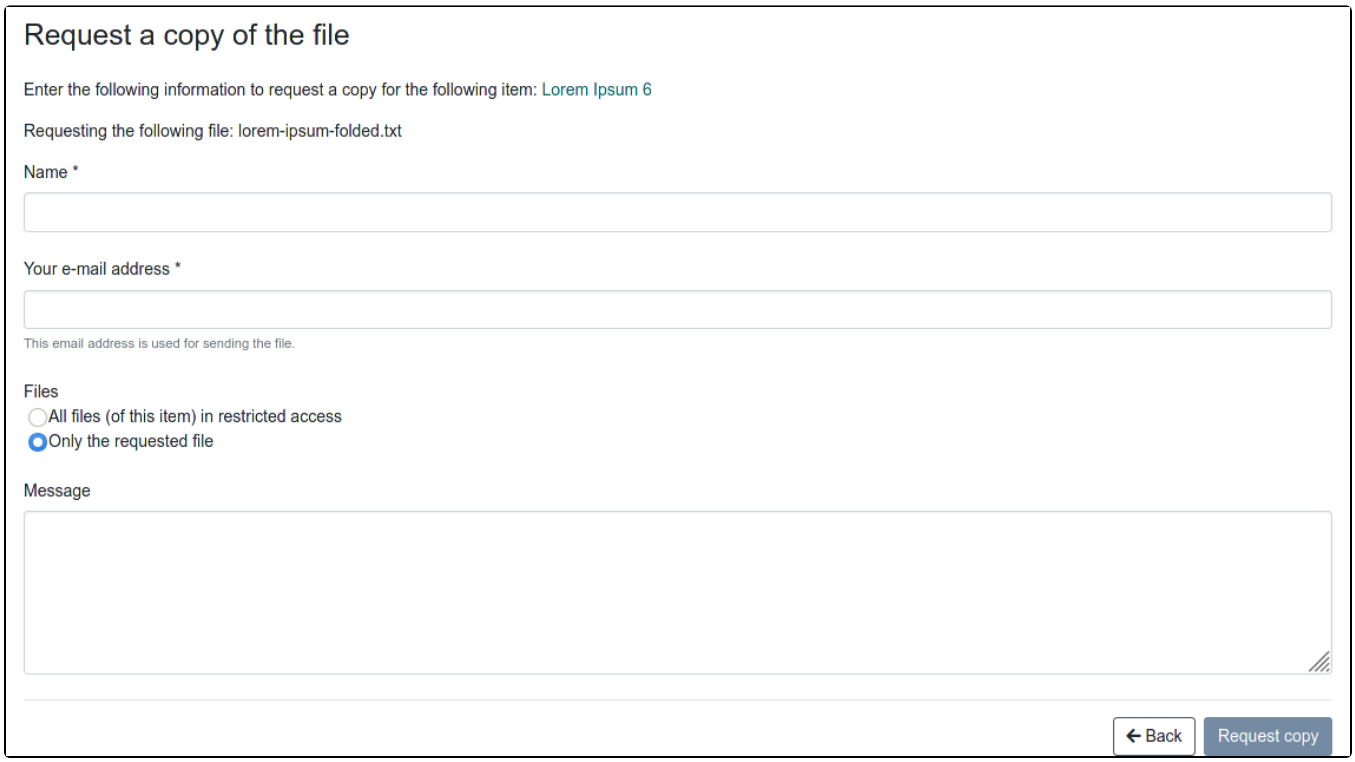
The request a copy functionality was added to DSpace as a measure to facilitate access in those cases when uploaded content can not be openly shared with the entire world immediately after submission into DSpace. It gives users an efficient way to request access to the original submitter of the item,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

Requesting a copy using the User Interface

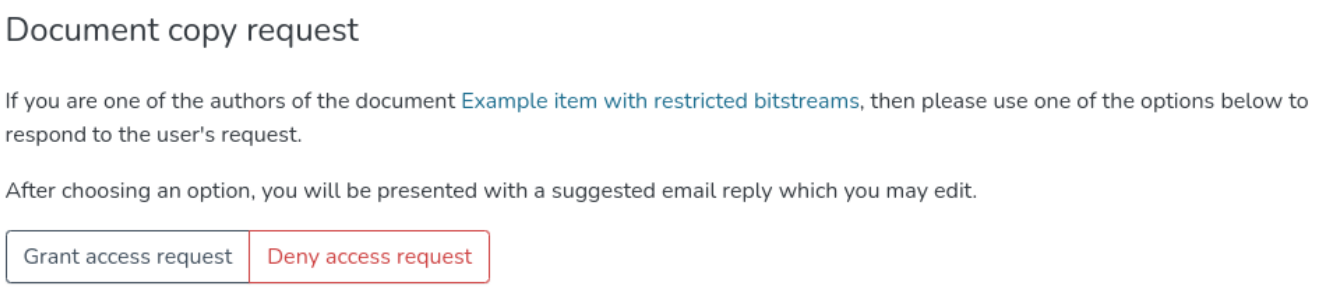
Users can request a copy by clicking the file thumbnail or the name of a file that the user is restricted from viewing.



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



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

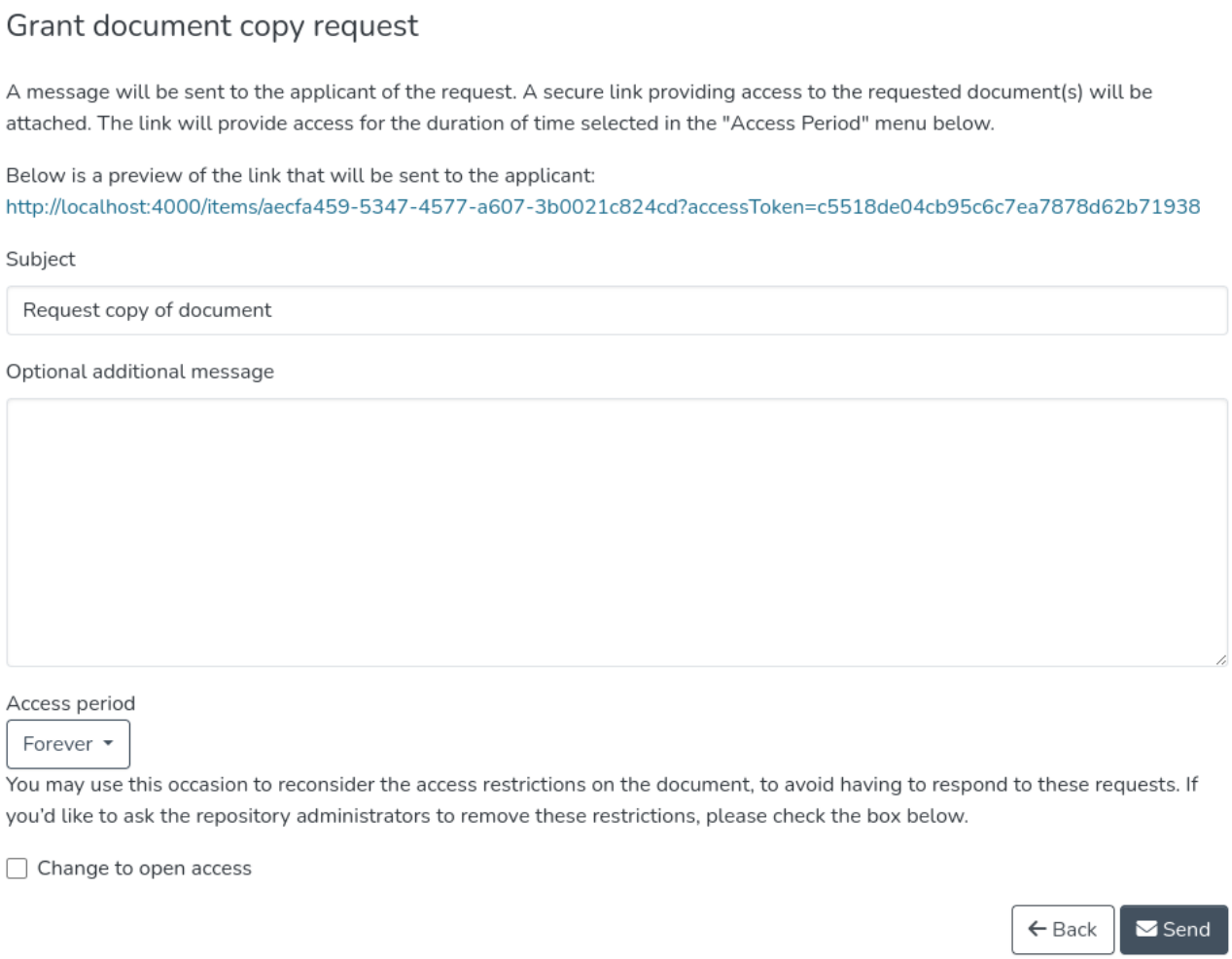


Each of these buttons registers the choice of the submitter, displaying the following form in which an additional reason for granting or rejecting the access can be added.

If sending links for files (above a certain size, or for all files) is enabled, a preview of the access token will be displayed

If sending links for files (above a certain size, or for all files) is enabled, an access period dropdown will also be shown on this form. If a period is selected, the end date of the access will be calculated from the time of approval.

If all requested bitstreams are to be sent only by email, the access period dropdown and link preview will not be displayed.



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

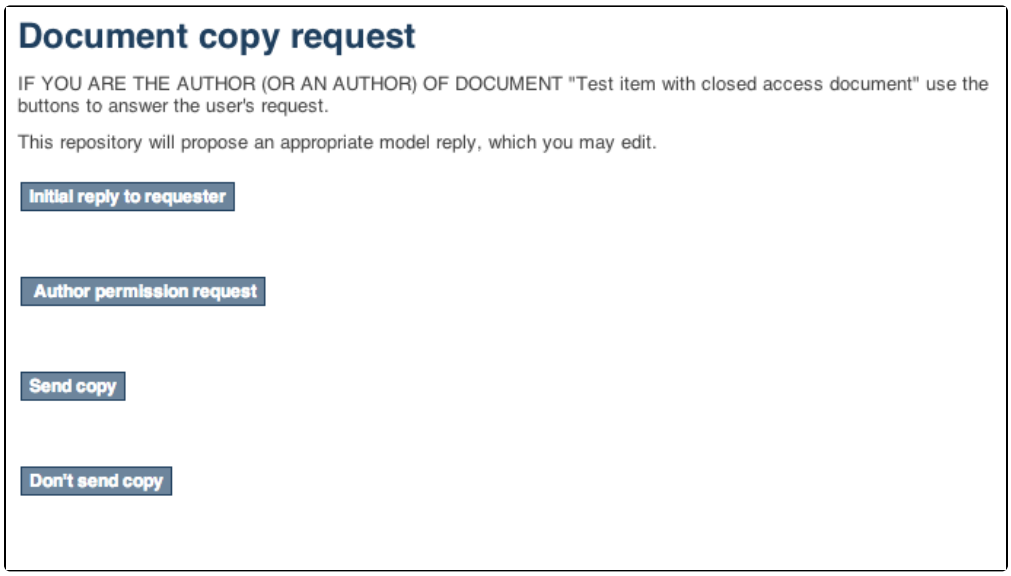
While responding positively to a request for copy, the person who approved may also ask the repository administrator to alter the access rights of the item, allowing unrestricted open access to everyone, by checking "Change to open access".

(Optional) Requesting a copy with Help Desk workflow

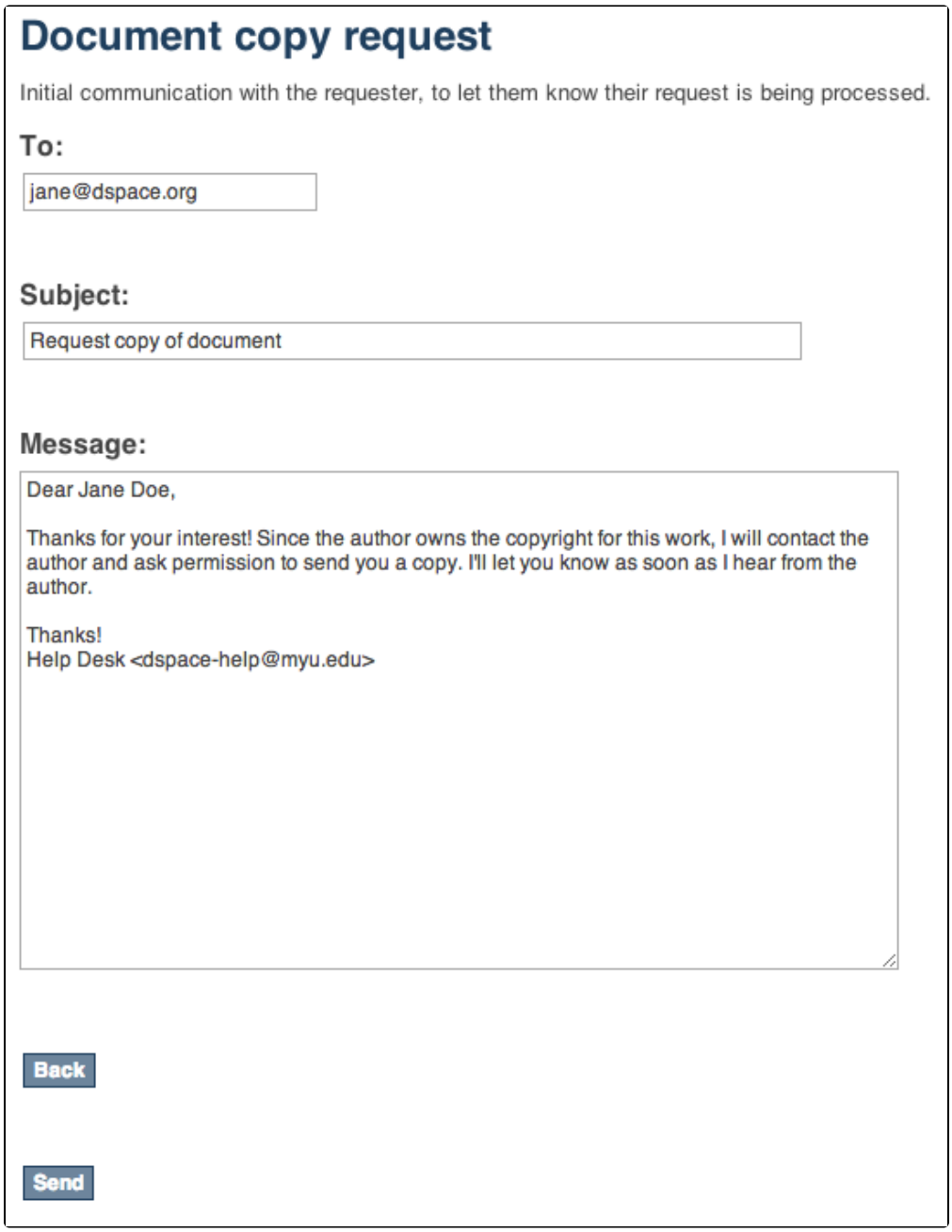
Available in 7.5 or later. However, in 7.5, users approving/rejecting these requests via the HelpDesk workflow **must** first authenticate. This is a known bug as described in https://github.com/DSpace/DSpace/issues/8636

As of 7.6, the HelpDesk workflow can be performed without requiring authentication (issue #8636 has been fixed)

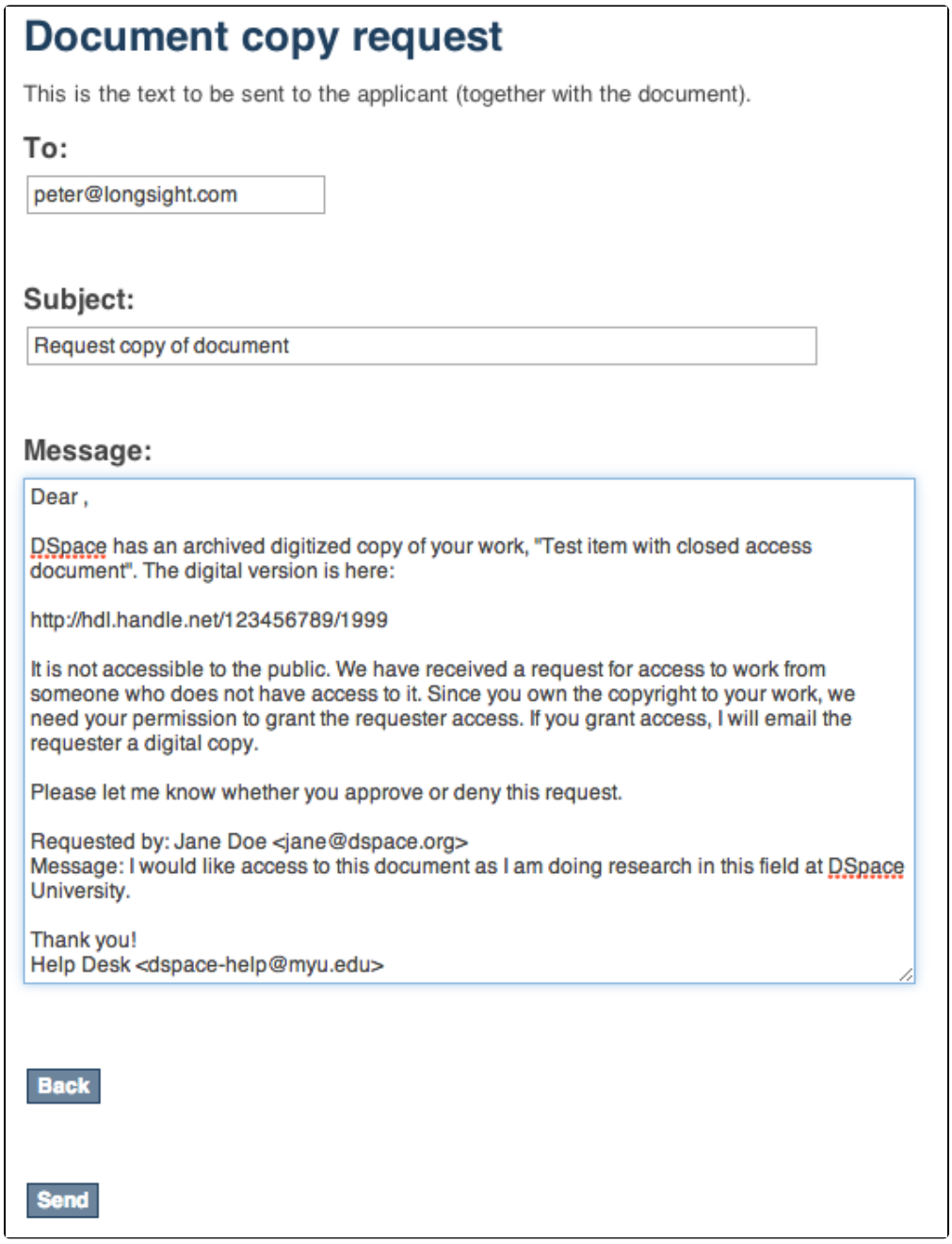
(Optional) Request Item with HelpDesk intermediary, is steered towards having your Repository Support staff act as a helpdesk that receives all incoming RequestItem requests, and then processes them. This adds the options of "Initial Reply to Requestor" to let the requestor know that their request is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



Initial Reply to Requester



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



Email templates

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

request_item.admin	template for the message that will be sent to the administrator of the repository, after the original submitter requests to have the permissions changed for this item.
request_item.author	template for the message that will be sent to the original submitter of an item with the request for copy.
request_item.granted	template for the message, with attachment, that will be sent to the requester on approval
request_item. granted_token	template for the message, with secure link and expiry date (if set), that will be sent to the requester on approval
request_item. rejected	template for the message that will be sent to the requester on rejection (unless configuration is set to silently reject requests)

Configuration parameters

Request a copy is enabled by default.

Note: As of Dspace 9.0, these properties have been moved from dspace.cfg to their own module in modules/requestitem.cfg

These configuration parameters in requestitem.cfg relate to Request a Copy:

Property:	`request.item.type`
Example Value	`request.item.type = all`
Informational Note	This parameter manages who can file a request for an item. The parameter is optional. When it is empty or commented out, request a copy is disabled across the entire repository. When set to **all**, any user can file a request for a copy. When set to **logged**, only registered users can file a request for copy.
Property:	`mail.helpdesk`
Example Value	`mail.helpdesk = foo@bar.com`
Informational Note	The email address assigned to this parameter will receive the emails both for granting or rejecting request a copy requests, as well as requests to change item policies. This parameter is optional. If it is empty or commented out, it will default to `mail.admin`. *WARNING:* This setting is only utilized if the `RequestItemHelpdeskStrategy` bean is enabled in `[dspace]/config/spring/api/requestitem.xml` (see below)
Property:	`request.item.helpdesk.override`
Example Value	`request.item.helpdesk.override = true`
Informational Note	Should all Request Copy emails go to the `mail.helpdesk` instead of the item submitter? Default is `false`, which sends Item Requests to the item submitter. *WARNING:* This setting is only utilized if the `RequestItemHelpdeskStrategy` bean is enabled in `[dspace]/config/spring/api/requestitem.xml` (see below)
Property:	request.item.reject.mail
Example Value	`request.item.reject.mail = true`
Informational Note	Should a rejection of a copy request send an email back to the requester? Defaults to "true", which means a rejection email is sent back. Setting it to "false" results in a silent rejection.
Property:	request.item.grant.bundles
Example Value	`request.item.grant.bundles = ORIGINAL`
Informational Note	Bundles to use when granting access and inspecting file size, if "all files" is indicated. This supports multiple values. To specify multiple bundles, separate them with a commao r (preferred) repeat the configuration property with additional values. Default: ORIGINAL
Property:	request.item.grant.link

Example Value	`request.item.grant.link = true`
Informational Note	Prior to DSpace 9.0, all granted requests would send the file as an attachment. If a secure web link to the bitstream(s) should be sent when files are over a certain file size threshold, enable the following property Default: true
Property:	request.item.grant.link.filesize
Example Value	`request.item.grant.link.filesize = 20`
Informational Note	If request.item.grant.link is enabled, you can specify a minimum file size (in megabytes) to use as a threshold, or else the default attachment functionality will be used instead. If 'all files' was indicated in the request, this threshold will be activated if any of the files meet the configured size minimum. To send links instead of attachments for all files, set this property to 0. Default: 20
Property:	request.item.grant.link.period
Example Values	`request.item.grant.link.period = FOREVER` `request.item.grant.link.period = +1DAY`
Informational Note	Valid access periods, in seconds, to allow the approver to select when using links. These are presented to users with friendly labels via i18n keys in the angular frontend. These should be in a format like +<n><unit>, where <n> is the number of units and <unit> is one of "DAYS", MONTHS", "YEARS". (or singular if n=1). The default is FOREVER, which will set a very distant expiry date for permanent access. The first access period in the list will be the default option in the grant form dropdown. To configure displayed labels, edit the angular .json5 i18n assets to include keys beginning with "grant-request-copy.access-period." and ending in the period, e.g.: "grant-request-copy.access-period.+1DAY": "1 day", "grant-request-copy.access-period.+7DAYS": "1 week", "grant-request-copy.access-period.+1MONTH": "1 month", "grant-request-copy.access-period.+3MONTHS": "3 months", "grant-request-copy.access-period.FOREVER": "Forever",
Property:	request.item.grant.link.dateformat
Example Value	`request.item.grant.link.dateformat = yyyy-MM-dd`
Informational Note	Date format to use for the access expiry date in the grant email
Property:	request.item.create.captcha
Example Value	`request.item.create.captcha = false`
Informational Note	Require a captcha for item request creation. Note, this requires altcha.cfg to be configured with a valid HMAC key Default: false

Protecting with Captcha

To see more about configuring and using ALTCHA, see: CAPTCHA Verification

Selecting Request a Copy strategy

The process that DSpace uses to determine who is the recipient of the Item Request is configurable in this Spring file: `[dspace]/config/spring/api/requestitem.xml`

New in DSpace 7

The strategy is selected using a Spring `<alias alias='org.dspace.app.requestitem.RequestItemAuthorExtractor' name='theStrategyClass'/>`. Previously this was done by moving the `RequestItemAuthorExtractor id` to the selected `<bean>` definition.

New in DSpace 7.6

The strategy is selected by configuring it into the `<bean/>` for `RequestItemMetadataStrategy` as a constructor argument.

New in DSpace 10.0

New Primary Bitstream strategy - `RequestPrimaryBitstreamMetadataStrategy` - is available for requesting a copy when defining the email address at the Bitstream metadata level

Configure who gets request via a metadata field

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

Syntax for 7.6 or later

```
<!-- This alias defines that you want to use the RequestItemMetadataStrategy (this is enabled by default) -->
<!-- This bean specifies that you want to use the RequestItemMetadataStrategy (this is enabled by default) -->
<bean class="org.dspace.app.requestitem.RequestItemEmailNotifier" lazy-init='false'>
        <description>This sends various emails between the requestor and the grantor.</description>

        <!-- Modify the "ref" here to point at the "RequestItemHelpdeskStrategy" -->
        <constructor-arg index='0'
                         ref='org.dspace.app.requestitem.RequestItemMetadataStrategy'/>
</bean>

<!-- This bean allows you to specify which metadata field is used (if any) -->
<bean class="org.dspace.app.requestitem.RequestItemMetadataStrategy"
      id="org.dspace.app.requestitem.RequestItemMetadataStrategy">
  <!--
  Uncomment these properties if you want lookup in metadata the email and the name of the author to contact for
request copy.
  If you don't configure that or if the requested item doesn't have these metadata the submitter data are used
as fail over

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

  -->
</bean>
```

Syntax for 7.5 or earlier

```
<!-- This alias defines that you want to use the RequestItemMetadataStrategy (this is enabled by default) -->
<alias alias='org.dspace.app.requestitem.RequestItemAuthorExtractor'
       name='org.dspace.app.requestitem.RequestItemMetadataStrategy'/>

<!-- This bean allows you to specify which metadata field is used (if any) -->
<bean class="org.dspace.app.requestitem.RequestItemMetadataStrategy"
      id="org.dspace.app.requestitem.RequestItemMetadataStrategy"
      autowireCandidate="true">
  <!--
  Uncomment these properties if you want lookup in metadata the email and the name of the author to contact for
request copy.
  If you don't configure that or if the requested item doesn't have these metadata the submitter data are used
as fail over

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

  -->
</bean>
```

Configure this as follows:

1. Create a metadata field which you'd like to use to store this email address (and optionally a second metadata field for the full name).
    a. Hint: You may wish to add this metadata field to your "metadata.hide.*" settings in local.cfg in order to ensure this metadata field is hidden from normal users & is only visible to Administrative users. That way this email address will NOT appear in Item display pages (except to Administrators)
2. Uncomment the "emailMetadata" setting above, and configure it's "value" to use the new metadata field.
3. Edit the Item(s) which you wish to use this field. Add the new metadata field to those items, given it a value of the email address which will *receive* the request for copy. By default, if an Item does NOT have this metadata field, the request for copy will still go to the Item's submitter.

Configure all requests to go to a helpdesk email

Prior to 7.6, all users who wish to respond to a request to the helpdesk email *must first login* to DSpace. See https://github.com/DSpace/DSpace/issues/8636 This was fixed in 7.6.

Another common request strategy is the use a single Helpdesk email address to receive all of these requests (see corresponding helpdesk configs in dspace.cfg above). If you wish to use the Helpdesk Strategy, you must replace the references to the default `RequestItemMetadataStrategy`, bean with the `RequestItemHelpdeskStrategy` bean:

Syntax for 7.6 or later

```
<!-- To change the settings, you need to modify the constructor-arg (see below) to use the
"RequestItemEmailNotifier" bean.-->
<bean class="org.dspace.app.requestitem.RequestItemEmailNotifier" lazy-init='false'>
        <description>This sends various emails between the requestor and the grantor.</description>

        <!-- Modify the "ref" here to point at the "RequestItemHelpdeskStrategy" -->
        <constructor-arg index='0'
                         ref='org.dspace.app.requestitem.RequestItemHelpdeskStrategy'/>
</bean>
```

Syntax for 7.5 or earlier

```
<!-- Change this alias to use "RequestItemHelpdeskStrategy" bean-->
<alias alias='org.dspace.app.requestitem.RequestItemAuthorExtractor'
       name='org.dspace.app.requestitem.RequestItemHelpdeskStrategy'/>

<!-- Ensure the bean is uncommented (it should be by default) -->
<bean class="org.dspace.app.requestitem.RequestItemHelpdeskStrategy"
        id="org.dspace.app.requestitem.RequestItemHelpdeskStrategy"
        autowireCandidate="true"/>
```

Configure all requests to go to the administrators of a Collection

This strategy sends mail to all of the members of the administrators group for the collection which owns the item.

Syntax for 7.6 or later

```
<!-- To change the settings, you need to modify the constructor-arg (see below) to use the
"CollectionAdministratorsRequestItemStrategy" bean.-->
<bean class="org.dspace.app.requestitem.RequestItemEmailNotifier" lazy-init='false'>
        <description>This sends various emails between the requestor and the grantor.</description>

        <!-- Modify the "ref" here to point at the "CollectionAdministratorsRequestItemStrategy" -->
        <constructor-arg index='0'
                         ref='org.dspace.app.requestitem.CollectionAdministratorsRequestItemStrategy'/>
</bean>
```

Syntax for 7.5 or earlier

```
<!-- Change this alias to use "CollectionAdministratorsRequestItemStrategy" bean-->
<alias alias='org.dspace.app.requestitem.RequestItemAuthorExtractor'
       name='org.dspace.app.requestitem.CollectionAdministratorsRequestItemStrategy'/>

<!-- Ensure the bean is uncommented (it should be by default) -->
<bean class='org.dspace.app.requestitem.CollectionAdministratorsRequestItemStrategy'
      id='org.dspace.app.requestitem.CollectionAdministratorsRequestItemStrategy'
      autowireCandidate='true'/>
```

Combine multiple strategies

This strategy combines the results of other strategies into a single list of email recipients. Pass the strategy a single constructor argument, being a list of the strategy beans whose output should be combined.

In the following example, email will be sent to the address(es) found in the configured metadata fields (or to the submitter if none), *and* to the owning collection's administrators.

Syntax for 7.6 or later

```
<!-- To change the settings, you need to modify the constructor-arg (see below) to use the
"CombiningRequestItemStrategy" bean.-->
<bean class="org.dspace.app.requestitem.RequestItemEmailNotifier" lazy-init='false'>
        <description>This sends various emails between the requestor and the grantor.</description>

        <!-- Modify the "ref" here to point at the "CombiningRequestItemStrategy" -->
        <constructor-arg index='0'
                         ref='org.dspace.app.requestitem.CombiningRequestItemStrategy'/>
</bean>

<!-- This bean is where you can combine multiple strategies by referencing them in the <list> below -->
<bean class='org.dspace.app.requestitem.CombiningRequestItemStrategy'
      id='org.dspace.app.requestitem.CombiningRequestItemStrategy'>
    <constructor-arg>
        <description>A list of references to RequestItemAuthorExtractor beans</description>
        <list>
            <ref bean='org.dspace.app.requestitem.RequestItemMetadataStrategy'/>
            <ref bean='org.dspace.app.requestitem.CollectionAdministratorsRequestItemStrategy'/>
        </list>
    </constructor-arg>
</bean>
```

Syntax for 7.5 or earlier

```
<!-- Change this alias to use "CombiningRequestItemStrategy" bean-->
<alias alias='org.dspace.app.requestitem.RequestItemAuthorExtractor'
       name='org.dspace.app.requestitem.CombiningRequestItemStrategy'/>

<!-- This bean is where you can combine multiple strategies by referencing them in the <list> below -->
<bean class='org.dspace.app.requestitem.CombiningRequestItemStrategy'
      id='org.dspace.app.requestitem.CombiningRequestItemStrategy'
      autowireCandidate='true'>
    <constructor-arg>
        <description>A list of references to RequestItemAuthorExtractor beans</description>
        <list>
            <ref bean='org.dspace.app.requestitem.RequestItemMetadataStrategy'/>
            <ref bean='org.dspace.app.requestitem.CollectionAdministratorsRequestItemStrategy'/>
        </list>
    </constructor-arg>
</bean>
```

Configure who gets request via a metadata field defined within the primary bitstream metadata submission form

One could configure a new form field at your **submission-forms.xml** file:

For 10.0 or later

```
<form name="bitstream-metadata">
...
            <row>
                <field>
                    <dc-schema>person</dc-schema>
                    <dc-element>email</dc-element>
                    <dc-qualifier></dc-qualifier>
                    <repeatable>false</repeatable>
                    <label>Person's email for request a copy</label>
                    <input-type>onebox</input-type>
                    <hint></hint>
                    <required></required>
                </field>
            </row>
...
</form>
```

And also change the default dspace/config/spring/api/requestitem.xml configuration, to support this feature (or you can uncomment the existing configuration):

For 10.0 or later

```
<bean class="org.dspace.app.requestitem.RequestPrimaryBitstreamMetadataStrategy"
          id="org.dspace.app.requestitem.RequestPrimaryBitstreamMetadataStrategy">
        <description>
            Get recipients from a bitstream metadata field.
        </description>
        <!--
            Uncomment these properties if you want lookup in bitstream's metadata
            the email and the name of the author for the contact for request copy.
            If you don't configure that or if the bitstream doesn't have
            these metadata, the submitter data are used as a fail over.
        -->
        <!--
            <property name="emailMetadata"
                      value="schema.element.qualifier" />
            <property name="fullNameMetadata"
                      value="schema.element.qualifier" />  -->
</bean>
```

you can also combine this strategy with others by using the `CombiningRequestItemStrategy`.

Authority framework functionalities

- How to manage relationships between items
  - ItemAuthority
    - Features
    - Configuration
    - Entity Type Mappings
    - Frontend Auto-Fill Configuration
    - Custom query filters for ItemAuthority query
  - External Authority Integration
    - ORCID Registry Lookup
    - Orcid expanded search
      - Authority and extra informations
  - Vocabulary Multilingualism

The Authority Framework enables real-time entity lookup, confidence-based linking, and automatic graph maintenance through server-side consumers.

On the UI, it introduces a new way to link authority controlled metadata to the related item, generating a link in search results or item page, this connection is handled in submission.

When the relationship is managed internally, i.e between dspace items the uuid of the linked item is stored as authority value in the metadata holding the relationship from the source item. For instance, given a publication Item A the metadata dc.contributor.author will have the value “Mario Rossi” and the authority of such metadata value will be the UUID of a Person Item B that represents the person “Mario Rossi”.

In order to use the **Authority Framework** functionality the Virtual Metadata feature should be disabled with the following property:

```
relationship.enable-virtual-metadata = false
```

Be aware that currently there is no way to go back and forth from Virtual Metadata Relations to Authority Framework Relations.

How to manage relationships between items

Relationships are always implicitly bidirectional with one preferred side that will be used to actually store the data. The side holding the relationship is usually identified as the one that changes more frequently compared to the other or the one that usually is created later. For instance to store the “author” relationship between publication - person it is better to use metadata in the publication record so that when new publications are added to the system for an existing person only the new publication record is touched and there is no need to edit also the person record.

The primary class for retrieving authority choices from DSpace items is the **ItemAuthority**.

ItemAuthority are configured using the *authority.cfg* file; a configuration example is shown below:

```
plugin.named.org.dspace.content.authority.ChoiceAuthority = \
 org.dspace.content.authority.ItemAuthority = OrgUnitAuthority

choices.plugin.person.affiliation.name = OrgUnitAuthority
choices.presentation.person.affiliation.name = suggest
authority.controlled.person.affiliation.name = true

cris.ItemAuthority.OrgUnitAuthority.entityType = OrgUnit
```

From this configuration we can understand that the metadata *person.affiliation.name* is linked to an *OrgUnitAuthority* and to a Person type entity.



ItemAuthority

It provides Solr-based searching for authority records.

Features

1. Solr-based search - Queries Solr Authority core
2. Dynamic scoping - Filters by entity type via entityType config
3. Frontend auto-fill - Supports data- prefixed metadata for auto-population
4. Pluggable filtering - Custom authority filters via CustomAuthorityFilter

Configuration

```
# Plugin Registration
choices.plugin.{metadata_field} = {authority_class}

# Presentation Style
choices.plugin.{metadata_field}.presentation = autocomplete|lookup|select

# Controlled Fields
authority.controlled.{metadata_field} = true|false

# Entity Type Mapping
cris.ItemAuthority.{metadata_field}.entityType = {entity_type}
```

Entity Type Mappings

```
cris.ItemAuthority.{metadata_field}.entityType = {EntityType}
```

Frontend Auto-Fill Configuration

Through a rest configuration, it is also possible to activate an "auto-fill" functionality, which will populate the submission fields based on the selected value (value must be authority controlled).

Example of config in *cris-authority-metadatagenerator.xml*:

```
<bean class="org.dspace.content.authority.ItemSimpleAuthorityMetadataGenerator">
    <property name="authorityName" value="AuthorAuthority"/>
    <property name="keyId" value="person_identifier_orcid"/>
    <property name="useForDisplay" value="true"/>
    <property name="useAsData" value="true"/>
    <property name="schema" value="person"/>
    <property name="element" value="identifier"/>
    <property name="qualifier" value="orcid"/>
    <property name="singleResultOnAggregate" value="true"/>
</bean>
```

- `authorityName` : is the name of the authority that will provide the metadata enrichment
- `keyId`: is the name of the metadata that will be exposed in the other-information property for the authority entry. If `useAsData` is set to true this metadata will be enriched in the submission form where the authority entry is used
- `useForDisplay` : if this flag is set to true the value of the metadata set in the `keyId` is also displayed as additional information when the authority entry is shown as result in the submission form (if missing default value is true)
- `useAsData` : if this flag is set to true the value of the metadata set in the `keyId` is used for the enrichment, so it's required Metadata enrichment functionality (if missing default value is true)
- `schema` : the schema of the metadata belonging to the authority entry which is exposed and used for the enrichment
- `element` : the element of the metadata belonging to the authority entry which is exposed and used for the enrichment
- `qualifier` : the qualifier of the metadata belonging to the authority entry which is exposed and used for the enrichment
- `singleResultOnAggregate` : if set to `true`, the search results are squashed by placing the various values in the resulting string when they are different from each other. Otherwise, the different results are listed. It is used with `true` only with search queries that give unique and precise results (such as queries based on ORCID value).

Considering the previous configuration if the `AuthorAuthority` is used in a Publication entity, when an authority entry is selected the metadata `oairecerif.author.affiliation` present in the Publication is enriched with the value of the metadata `person.affiliation.name` present in the item which the authority entry refer to.

Note : Every metadata defined as source for the authority enrichment should be present even in the property `discovery.index.projection` of the discovery.cfg file, e.g. :

```
discovery.index.projection=person.affiliation.name
```

After changing the `discovery.index.projection` property a **reindex is required.**

Custom query filters for ItemAuthority query

Standard solr query that performs the lookup of items during ItemAuthority (`org.dspace.content.authority.ItemAuthority`) `getMatches()` method can be extended with custom query filters.

To facilitate this, standard abstract class `org.dspace.content.authority.CustomAuthorityFilter` is provided. To define custom filter, this class needs to be extended and instrumented as Spring Bean.

Extending classes must implement `appliesTo()` method, that checks if custom filter queries need to be applied, according to its own custom internal logic.

Two ready-to-be configured extensions are provided by classes `org.dspace.content.authority.EntityTypeAuthorityFilter` and `org.dspace.content.authority.AuthorityNameAuthorityFilte` that allow users to define some static queries which don't need to use particular runtime logic.

`org.dspace.content.authority.EntityTypeAuthorityFilter` provides a way to filter out the elements according to their Entity type. The supported entities need to be listed using the `supportedEntities` field, while the filtering parameters are specified with the `customQueries` field.

`org.dspace.content.authority.AuthorityNameAuthorityFilter` provides a way to filter out the elements according to their Authority name. The supported entities need to be listed using the `supportedEntities` field, while the filtering parameters are specified with the `customQueries` field.

`customQueries` , `supportedAuthorities` and `supportedEntities` fields can be set directly while configuring the Spring bean.

Following snippet is an example of filtering by Entity type:

```
        <bean class="org.dspace.content.authority.EntityTypeAuthorityFilter">
                <property name="supportedEntities">
                        <util:list>
                                <value>EntityOrg</value>
                                <value>EntityPerson</value>
                        </util:list>
                </property>
                <constructor-arg name="customQueries">
                        <util:list>
                                <value>dc.type:mytype</value>
                        </util:list>
                </constructor-arg>
        </bean>
```

Following snippet is an example of filtering by Authority name:

```
        <bean class="org.dspace.content.authority.AuthorityNameAuthorityFilter">
                <property name="supportedAuthorities">
                        <util:list>
                                <value>EntityOrgAuthority</value>
                                <value>EntityPersonAuthority</value>
                        </util:list>
                </property>
                <constructor-arg name="customQueries">
                        <util:list>
                                <value>dc.type:mytype</value>
~/Projects/4science/dspace-cris/source-rest task/main-cris/DSC-2688*
            </util:list>
                </constructor-arg>
        </bean>
```

In order to enable one of these custom filters, the corresponding Spring configuration xml needs to be put in any spring configuration file under the following path `config/spring/api`.

External Authority Integration

ORCID Registry Lookup

For metadata controlled by an authority related to Person item, it is possible to configure the org.dspace.content.authority.**OrcidAuthority** which allows, in addition to searching for matches between persons stored in DSpace-CRIS, also to search for profiles on the ORCID register. The OrcidAuthority is therefore a class that extends the org.dspace.content.authority.**ItemAuthority** and appends the profiles identified on the ORCID register to the results identified by the ItemAuthority.

To set which metadata should be associated with this authority it is necessary to modify the file **authority.cfg**, as for the other authorities. By default, all the authorities linked to entities of type Person are configured with the OrcidAuthority.

Orcid expanded search

To identify the profiles on the ORCID register that match a specific string (the name of the person linked to the profile), the OrcidAuthority uses the **expand ed search endpoint** made available by ORCID. For more information about the expanded search click here.

Given an input “XYZ”, the query sent to the search endpoint is the following:

```
given-names:XYZ+OR+family-name:XYZ+OR+other-names:XYZ
```

If the input name has spaces or commas then the query shown above is duplicated for each section of the name, concatenating them with AND. For example, a search for "Smith, John" produces the following query:

```
(given-names:Smith+OR+family-name:Smith+OR+other-names:Smith) AND (given-names:John+OR+family-name:
John+OR+other-names:John)
```

The request that is made is paged on the basis of the pagination set and taking into account the number of results resulting from the research of the ItemAuthority.

If the ORCID API keys are configured then the search is performed on the API endpoint, first obtaining a read public access-token; if instead the API keys are not configured then the public endpoint is contacted.

Authority and extra informations

For each profile identified by the search described above, a choice is constructed with the following attributes:

- **authority** the ORCID iD of the profile with the prefix configured through the xxx property.
  Example: will be referenced::ORCID::0000-0000-0123-4567
- **label** the given and family name of the profile
- **value** the given and family name of the profile
- **extras** the ORCID iD (data-person_identifier_orcid) and, if present, the list of all the institutions linked to the profile (institution-affiliation-name).

Vocabulary Multilingualism

In addition, DSpace supports multilingualism for authority-controlled vocabularies. To achieve this, a vocabulary version for each language should be enabled by the property: `webui.supported.locales`

For instance if we have

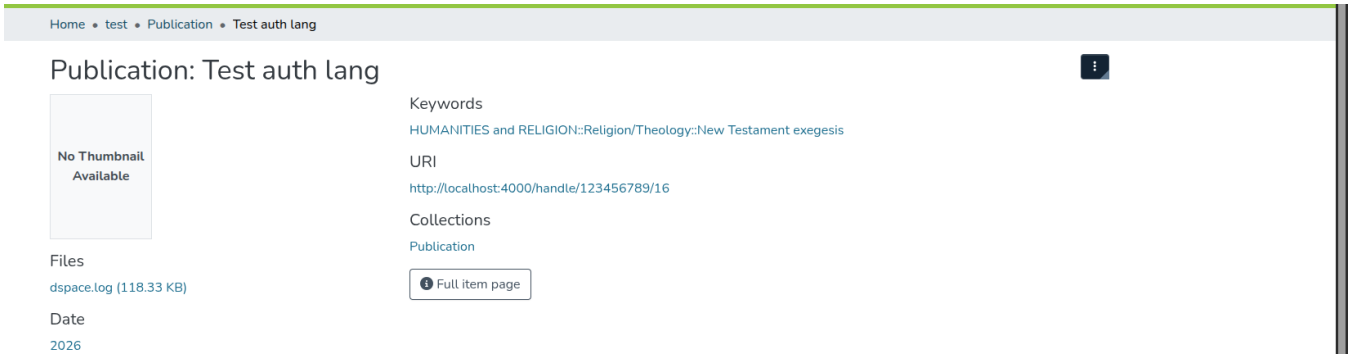
```
webui.supported.locales = en, de
```

we can create a translated vocabulary (e.g. *srsc_de.xml*) and then configure the authority as follows (authority.cfg):

```
vocabulary.plugin.srsc.hierarchy.store = true
vocabulary.plugin.srsc.hierarchy.suggest = true
vocabulary.plugin.srsc.delimiter = "::"
vocabulary.plugin.authority.store = true
authority.controlled.dc.subject = true
```

This will enable to display the translated value in the UI, if the language matches the translated one:

EN:



DE:



Creation of linked entities

- Main authority types
  - OrcidAuthority - External Source (ORCID Registry)
  - ItemAuthority - Local DSpace Entities
- Configuration
  - Link the metadata value to an existing items
  - Link the metadata value to a non-existing items
- Consumer's Logic
  - New Item's Creation
- AuthorityImportFiller
  - ItemMetadataImportFiller
- ItemSearchService and ItemSearcherMapper
  - Skip metadata with empty authority

During the submission of an item it is usual to link this new item to others. This is for instance the case of a publication item that during the submission is linked with person items (authors, editors, etc.), project item, organisation item, journal item, etc

When the item to link is not yet present in the system, i.e. the person item for an author doesn't exist or the project item, etc. it is possible to configure the system to automatically create a new item for him using the data provided in the publication item under submission and/or enriched with extra information retrieved from external sources with a pluggable logic (i.e. an author identified in the ORCID Registry can get a person item filled with data from the ORCID Registry).

The linked items are created upon the acceptance of the submitted item into the archive. This means that no linked items are created when the submitted item is in the workspace or workflow.

The creation is performed by a DSpace consumer named **CrisConsumer** that listen for INSTALL and MODIFY events over items. Again the normal flow is related to the INSTALL event that is triggered when the submitted item is accepted in the repository, the MODIFY events are listed to process edits of the submitted item done after that the item has been archived.

Main authority types

OrcidAuthority - External Source (ORCID Registry)

Searches both local DSpace persons AND the ORCID registry for researchers.

ItemAuthority - Local DSpace Entities

Searches existing items in your repository by entity type (Person, Project, OrgUnit, etc.).

Configuration

The CrisConsumer is configured in the **dspace.cfg**

```
event.consumer.crisconsumer.class = org.dspace.authority.CrisConsumer
event.consumer.crisconsumer.filters = Item+Install|Modify|Modify_Metadata
```

Once an item has been accepted in the repository the consumer will scan all his metadata looking for the ones associated with an ItemAuthority, i.e. used to link the item with other items.

For instance with a default configuration for a publication item the **dc.contributor.author** is used to link the publication with person items. In such case the CrisConsumer will create an item for the authors in the publication that doesn't have one yet or in some case, see below, will associate the author string to an existing person item. The entity type (**cris.entity.type**), Person, to assign to the item to be created is identified analyzing the configuration of the ItemAuthority instance associated with the dc.contributor.author metadata

```
cris.ItemAuthority.AuthorAuthority.entityType = Person

...
choices.plugin.dc.contributor.author = AuthorAuthority

...
```

Link the metadata value to an existing items

For each metadata value that must be processed the consumer will verify if an item for the same value was created before. This could be the case when several submissions are sitting in the workspace / workflow referencing the same “not yet existing” item (person, project, etc) and at a later point the items are progressively accepted. After the first ones, some of the references items would have been created and we expect the same to be reused by the other incoming items.

To do that the consumer will generate a sourceId from the metadata that will be used in the subsequent lookup. The use of a separate sourceId than the exact metadata value allows a more granular control about when an existing item should be used in the association.

Out of box many algorithms are provided to generate the sourceId

- if the authority starts with the prefix “**will be generated::**” or “**will be referenced::**” the sourceId will be the remaining part of the authority
- default, the sourceId will be the md5 hash of the metadata textual value uppercase. In such way the same, case insensitive, textual value will be always linked to the same target item
- uuid, the sourceId will be a random UUID. This strategy guarantee that no match with previous created items will never occur so a new item will be created for each value

To enable the uuid strategy it is needed to add the following property in the configuration

```
cris.import.submission.strategy.uuid.{field_key} = true
```

please note that **the property use the field key and not the dot representation** of the metadata so to enable it over the metadata dc.contributor.author the following property should be added to the dspace configuration (in the dspace.cfg or any overriding file such the local.cfg)

```
cris.import.submission.strategy.uuid.dc_contributor_author = true
```

If the CrisConsumer generates a new item this new item will be created at least with the following metadata

- **cris.sourceId** containing the generated sourceId
- **cris.entity.type** with the entity-type of the generated item
- **dc.title** with the textual value of the metadata that have generated the item

The lookup will also check the entity type of the item to link to identify valid match so that the same sourceId for a Person and a Project will lead to two different items.

For example, with the default hash strategy, the submission of a publication item with the metadata dc.contributor.author = “Mario Rossi” will lead to the calculation of a cris.sourceId as md5sum of “MARIO ROSSI” and a lookup for an item with the following metadata cris.sourceId = md5sum(MARIO ROSSI) AND relationship.type = “Person”

Link the metadata value to a non-existing items

For each metadata value that must create new items when they are not linked to anyone follow this configuration:

1. configure `cris-consumer.skip-empty-authority = false` on cris.cfg in order to manage authority controlled metadata value **without** aut hority set on CrisConsumer
2. configure `cris.ItemAuthority.{authorityName}.entityType` for each authority that must create new items on authority.cfg
   PS: for fields with same authority but with different behaviours (e.g. one must create the new items and another one must keep the simple string inserted) you must create two different authorities: one with `cris.ItemAuthority.{authorityName}.entityType` configured and another one without it
3. remove eventually `authority.required.{fieldName}` for fields that use authority that must create new items on authority.cfg
4. It is also possible (in addition to what defined at point 2) to skip related entity creation on a metadata basis by setting `cris-consumer.skip-empty-authority = false` and `cris-consumer.skip-empty-authority.metadata` with the list of metadata for which, if authority is empty, a related item must not be created.

Consumer’s Logic

Once an item has been archived the CrisConsumer performs the following operations:

1. identifies the metadata that can be associated to entities by consulting the configuration of the Authorities
2. calculates the **cris.sourceId** by adopting one of the strategies described above
3. calculates the **cris.entity.type** using the Authority configuration linked to the specific metadata
4. starting from the calculated cris.sourceId and relationship.type searches for an already existing item using the org.dspace.authority.service.**ItemSe archService** (specific “ItemSearchService and ItemSearcherMapper“ for more details):
   - if such an item exists, its UUID is used to set the metadata authority, in order to link the already existing item to the metadata of the just archived one;
   - if such an item does not exist and the authority didn’t have the prefix “will be referenced::” then the consumer creates a new one and then uses its UUID to enhance the metadata authority of the newly stored item.
5. enriches/modifies the related item identified using a specific **AuthorityImportFiller**

New Item’s Creation

If it is necessary to create a new item to associate with the metadata, the CrisConsumer configures it in the following way::

- the **submitter** of the original item that has just been archived is set as the item's submitter
- as **owning collection** is set the "closest" collection to that of the archived item that has, among its metadata, a relationship.type equal to that calculated for the item under creation. In order to search for such a collection, all the collections present in the community of the archived item are examined: if among them there is a collection with the right relationship.type, it is used, otherwise, in a recursive way, the procedure goes up to the higher community and within it, even among the sub-communities, a collection is searched for, that has the value of the metadata relationship. type searched for. The search ends when there are no longer any higher level communities to examine, For example, assuming the following communities/collections structure

    - Community A
        - Collection A publication
        - Collection A person
    - Community B
        - Collection B publication
        - Collection B person

  a submission in Collection A publication that gives rise to a new "person" item should create it in Collection A person while a submission in Collection B publication should use Collection B person..
  If with the strategy described above the consumer has not been able to identify a proper collection, the creation of the CRIS item is not performed and a warning message is shown in the console. However, this does not preclude the creation of other items to be associated with other possible metadata.
- adds the **cris.sourceId** metadata to the item
- adds the **relationship.type** metadata to the item (if the item hasn't already inherited it from the collection identified with the algorithm described above)

Once the item is configured the consumer can decide whether to install it (default) or to start a workflow associated with it. The choice is made according to the **cris.import.submission.enabled.entity.{field}** configuration, or, if not present, according to the generic **cris.import.submission.enabled.entity** pr operty.

AuthorityImportFiller

The system provides a mechanism to dynamically enrich the items created using additional logic. This mechanism is called the "ImportFiller framework" and is configured through a spring configuration present by default in the cris-plugin.xml file. A possible configuration is shown below.

```
<util:constant id="CrisConsumer-SOURCE_INTERNAL"
      static-field="org.dspace.authority.CrisConsumer.SOURCE_INTERNAL" />

<bean id="org.dspace.authority.filler.AuthorityImportFillerHolder"
      class="org.dspace.authority.filler.AuthorityImportFillerHolder">
    <property name="fillers">
            <map>
                <entry key-ref="CrisConsumer-SOURCE_INTERNAL">
                    <bean class="org.dspace.authority.filler.ItemMetadataImportFiller" parent="fullItemMetadataCo
nfiguration" />
                </entry>
                <entry key="ORCID" value-ref="orcidImportFiller" />
                <entry key="ISSN" value-ref="sherpaImportFiller" />
            </map>
        </property>
</bean>
<bean class="org.dspace.authority.filler.ExternalDataProviderImportFiller" id="orcidImportFiller" >
       <constructor-arg name="externalDataProvider" ref="org.dspace.external.provider.impl.
OrcidV3AuthorDataProvider"/>
       <constructor-arg name="authorityIdentifier" value="ORCID"/>
</bean>

<bean class="org.dspace.authority.filler.ExternalDataProviderImportFiller" id="sherpaImportFiller" >
    <constructor-arg name="externalDataProvider" ref="org.dspace.external.provider.impl.
SHERPAv2JournalISSNDataProvider"/>
    <constructor-arg name="authorityIdentifier" value="ISSN"/>
</bean>
```

The **AuthorityImportFillerHolder** class collects all AuthorityImportFiller configured in the system in a map..

This map associates the possible implementations of the Filler to the value of an authority key, whose value depends on the metadata related to the CRIS it em:

- If the metadata that caused the creation of the new item had no authority, the authority key is the **SOURCE_INTERNAL** value and the **ItemMetad ataImportFiller** filler is used for enrichment.
- If the starting metadata has an authority that starts with "will be generated::ORCID::" then the selected filler will be an instance of `ExternalData ProviderImportFiller` using `OrcidV3AuthorDataProvider` as data provider
- If the starting metadata has an authority that starts with "will be generated::ISSN::" then the selected filler will be an instance of `ExternalDataPr oviderImportFiller` using `SHERPAv2JournalISSNDataProvider` as data provider

ItemMetadataImportFiller

The ItemMetadataImportFiller is configured as Spring's bean and represents the fillers used if the metadata associated with the item has no particular authority. This filler allows you to enrich the CRIS item by copying metadata present in the archived/modified item; the filler also allows you to update CRIS items previously created.

Below is a possible configuration example:

```
<bean class="org.dspace.authority.filler.ItemMetadataImportFiller"
      id="fullItemMetadataConfiguration" abstract="true">
    <property name="itemService" ref="org.dspace.content.ItemServiceImpl" />

    <property name="allowsUpdateByDefault" value = "true|false" />
    <property name="configurations">
        <map>
            <entry key="dc.contributor.author">
                ....
            </entry>
            <entry key="dc.contributor.editor">
                ....
            </entry>
        </map>
    </property>
</bean>
```

where for each entry on the configurations map there is such a configuration:

```
<bean class="org.dspace.authority.filler.MetadataConfiguration">
    <property name="updateEnabled" value="true" />
    <property name="mapping">
        <map>
            <entry key="oairecerif.editor.affiliation">
                <bean class="MetadataConfiguration.MappingDetails">
                    <property name="visibility" value="1|0" />
                    <property name="useAll" value="true|false" />
                    <property name="appendMode" value="true|false" />

                </bean>
            </entry>
        </map>
    </property>
</bean>
```

The **configurations** map shows the possible metadata that can generate the CRIS item, to be enriched with additional metadata. Each of these configured metadata refers to a **MetadataConfiguration** that has the following properties:

- **updateEnabled**: in the presence of a CRIS item that has been found in the database through the cris.sourceId search, this property indicates if the filler can act on it or if the update is not enabled. If this property is not set as default, the value of the **allowsUpdateByDefault** property of the **ItemMetadataImportFiller** is considered (which if not set, has a false value as default)
- **mapping:** iindicates the metadata to be copied from the item stored, in the CRIS item

Il mapping che consente di riportare i metadati dall'item archiviato è rappresentato da una mappa in cui la chiave rappresenta il metadato da copiare e il valore la configurazione con cui effettuare l'operazione:

The mapping that allows you to copy the metadata from the archived item is represented by a map in which the key represents the metadata to be copied and the value represents the configuration by which perform the operation:

- **visibility**: allows to specify the visibility of the metadata to be added to the CRIS item; the eligible values are private (0) or public (1) (to be implemented)
- **appendMode:** indicates whether the new metadata should be added to the set of metadata of the same type eventually already present in the CRI item, or they should overwrite them (default = false)
- **useAll:** indicates whether all the metadata of the type referred to should be reported in the item to be enriched (true) , or only the one which has the same position as the metadata that gave raise to the CRIS item should be reported ((false, default)

With the reported configuration:

- if **useAll = false:** to the CRIS item related to the first dc.contributor.author only the first metadata oairecerif.editor.affiliation of the archived item would be added and so on.
- if **useAll = true:** to each CRIS metadata item dc.contributor.author would be added all metadata oairecerif.editor.affiliation of the archived item

ItemSearchService and ItemSearcherMapper

The cris-plugin.xml contains the definition of a bean of type org.dspace.authority.service.**ItemSearchService** which is used to search an item with a specific strategy. The methods of this interface allow to search for an item for a specific parameter and for relationship type.

In particular, the search done by the org.dspace.authority.service.**ItemSearchServiceImpl** implementation consists of the following steps, executed in the order indicated until an item is found:

1. if the given searchParam is a valid uuid the item is searched by uuid; if the search is also made for relatioship.type, any item found is discarded if the type does not match.
2. a search by cris.sourceId and relatioship.type is done using the given searchParam and relationship type
3. if the search param has the form <identifier>::<value> a search is made using the bean org.dspace.authority.service.**ItemSearcherMapper**

This ItemSearcherMapper handle a Map of <String, ItemSearcher> in which the keys represent the various strategy identifier that can be used and the value corresponds to a particular implementation of the org.dspace.authority.service.**ItemSearcher** interface which, through the searchBy (Context context, String searchParam) method, allows to search the item with a specific strategy. The current provided implementations of the ItemSearcher interface are:

- org.dspace.authority.service.impl.**ItemSearcherById**: search the item by uuid
- org.dspace.authority.service.impl.**ItemSearcherByMetadata**: search the item by the metadata provided during the bean instantiation

The ItemSearcherMapper can be configured with a default ItemSearcher.

An example configuration is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<bean class=org.dspace.authority.service.ItemSearcherMapper" id="itemSearcherMapper">

   <constructor-arg index="0">
      <map>
         <entry key="UUID">
            <bean class="org.dspace.authority.service.impl.ItemSearcherById" />
         </entry>
         <entry key="ORCID">
            <bean class="org.dspace.authority.service.impl.ItemSearcherByMetadata">
               <constructor-arg ref="org.dspace.discovery.SearchService" />
               <constructor-arg value="person.identifier.orcid" />
            </bean>
         </entry>
         <entry key="RID">
            <bean class="org.dspace.authority.service.impl.ItemSearcherByMetadata">
               <constructor-arg ref="org.dspace.discovery.SearchService" />
               <constructor-arg value="person.identifier.rid" />
            </bean>
         </entry>
         <entry key="ISNI">
            <bean class="org.dspace.authority.service.impl.ItemSearcherByMetadata">
               <constructor-arg ref="org.dspace.discovery.SearchService" />
               <constructor-arg value="person.identifier.isni" />
            </bean>
         </entry>
         <entry key="DOI">
            <bean class="org.dspace.authority.service.impl.ItemSearcherByMetadata">
               <constructor-arg ref="org.dspace.discovery.SearchService" />
               <constructor-arg value="dc.identifier.doi" />
            </bean>
         </entry>
      </map>
   </constructor-arg>
   <constructor-arg index="1">
      <bean class="org.dspace.authority.service.impl.ItemSearcherById" />
   </constructor-arg>
</bean>
```

Skip metadata with empty authority

The **cris-consumer.skip-empty-authority** property allows to configure or not the CrisConsumer to skip the metadata that have an empty authority: in this way, therefore, the creation of related items can only be carried out if the authority has the prefixes **will be generated::** or **will be referenced::** .

The property **cris-consumer.skip-empty-authority.metadata** allows you to specify which metadata are an exception to the rule established through the previous property (cris-consumer.skip-empty-authority):

- if **cris-consumer.skip-empty-authority = true** the property metadata list the metadata fields for which an empty authority must not be skipped
- if **cris-consumer.skip-empty-authority = false** the property metadata list the metadata fields for which an empty authority must be skipped

Item details: layout

Work in progress

Inverse Relations

A new component has been implemented to display authority inverse relations on the UI. This component is bound to a *DiscoveryConfiguration* that can be parameterized with the *uuid* of the item. The *defaultFilterQueries* of the *DiscoveryConfiguration* can contains the placeholder {0} that will be replaced at runtime with the *uuid* of the item. Please refer to next paragraph for further details about how to set and configure such queries.

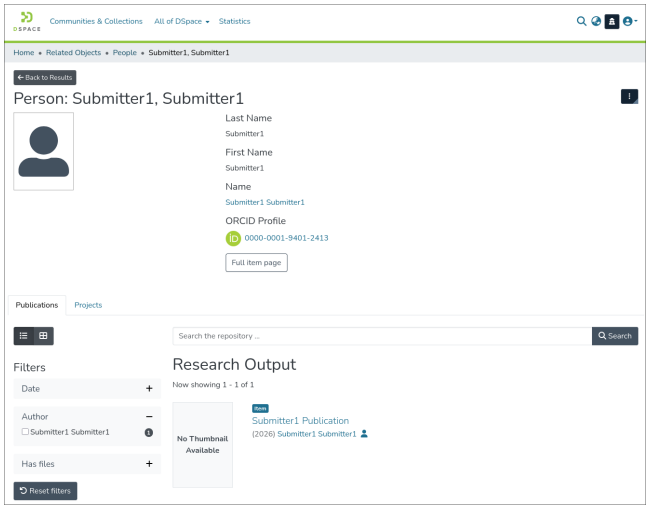
The new relations can be activated via a flag param in the *default.app.config.ts*, inside the item config:

```
item:
  # Enable authority based relations in item page
  showAuthorityRelations: true
```

if true the ui components will use the new authority based relations.

Currently supported Inverse Relations

- Person showing:
  - Publications
  - Projects
- Organizational Unit:
  - Researcher Projects
  - Researchers Publications
  - People
  - Projects



The current functionality needs all the Authority Framework entity to be configured. Please refer to Authority framework functionalities for further details.

Relation Discovery Configuration

Discovery queries are configured via `discovery.xml` file (`<dspace-install-dir>/config/spring/api/discovery.xml`) , in `org.dspace.discovery.configuration.DiscoveryConfigurationService` bean. A map's entry for each relation is provided, standard pattern for key is *RELATION.<Entity>.<relationName>*. For example, *RELATION.Person.researchoutputs* identifies the relation that will be used to populate the UI component.

The **core part of** `DiscoveryConfiguration`, for relation's set up is `defaultFilterQueries`. This section contains one or more filter queries to be performed, given Item's *uuid*, to find linked Items. In case many filter queries are provided, such queries are executed in sequence: the second query filters first query's results and so on.

This is the query that retrieves publications related to a person: *author_authority:{0} AND search.resourcetype:Item*, where {0} is a placeholder for UUID of the person.

Configuration Example

A full example of how relation that finds a Person publications is configured:

```
<bean id="org.dspace.discovery.configuration.DiscoveryConfigurationService" class="org.dspace.discovery.
configuration.DiscoveryConfigurationService">
        <property name="map">
            <map>
                  . . .


                              <entry key="RELATION.Person.researchoutputs" value-ref="
relationAuthorResearchOutputsConfiguration"/>

                  . . .
                        </map>
        </property>
</bean>


<!-- publications of an author -->
    <bean id="relationAuthorResearchOutputsConfiguration" class="org.dspace.discovery.configuration.
DiscoveryRelatedItemConfiguration">
        <!--Which sidebar facets are to be displayed-->
        <property name="sidebarFacets">
            <list>
                <ref bean="searchFilterIssued" />
                <ref bean="searchFilterAuthor" />
                    <ref bean="searchFilterEditor" />
                    <ref bean="searchFilterOrgUnit" />
                    <ref bean="searchFilterSubject" />
                    <ref bean="searchFilterFunding" />
                    <ref bean="searchFilterContentInOriginalBundle"/>
                    <ref bean="searchFilterType" />
            </list>
        </property>
        <!-- Set TagCloud configuration per discovery configuration -->
        <property name="tagCloudFacetConfiguration" ref="defaultTagCloudFacetConfiguration"/>
        <!--The search filters which can be used on the discovery search page-->
        <property name="searchFilters">
            <list>
                <ref bean="searchFilterTitle" />
                <ref bean="searchFilterIssued" />
                <ref bean="searchFilterAuthor" />
                <ref bean="searchFilterEditor" />
                <ref bean="searchFilterOrgUnit" />
                <ref bean="searchFilterSubject" />
                <ref bean="searchFilterFunding" />
                <ref bean="searchFilterContentInOriginalBundle"/>
                <ref bean="searchFilterType" />
            </list>
        </property>
        <!--The sort filters for the discovery search-->
        <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="sortFields">
                    <list>
                        <ref bean="sortByResearchOutputsSelection" />
                        <ref bean="sortTitleAsc" />
                    </list>
                </property>
            </bean>
        </property>
```

```
        <!--Any default filter queries, these filter queries will be used for all
                queries done by discovery for this configuration -->
        <property name="defaultFilterQueries">
            <list>
                <!--Only find related items. The placeholder {0} will be replaced with scope (UUID of item)-->
                <value>author_authority:{0} AND search.resourcetype:Item</value>
                <value>-withdrawn:true AND -discoverable:false</value>
            </list>
        </property>
        <!--Default result per page  -->
        <property name="defaultRpp" value="10" />
        <property name="hitHighlightingConfiguration">
            <bean class="org.dspace.discovery.configuration.DiscoveryHitHighlightingConfiguration">
                <property name="metadataFields">
                    <list>
                        <bean class="org.dspace.discovery.configuration.DiscoveryHitHighlightFieldConfiguration"
>
                            <property name="field" value="dspace.entity.type"/>
                            <property name="snippets" value="5"/>
                        </bean>
                    </list>
                </property>
            </bean>
        </property>
    </bean>
```

CAPTCHA Verification

- Google reCAPTCHA verification
  - Prerequisites
  - How to enable the feature
- ALTCHA verification
  - How to enable the feature

Google reCAPTCHA verification

This feature, when enabled, offers a powerful additional layer of protection against possible unwanted behaviors like massive registrations performed by bots using random or stolen email addresses. Feature can be enabled or disabled by decision of DSpace instance administrator, and is based on **Google reCAPTCHA**.

ReCAPTCHA supported versions are v2 with both invisible (https://developers.google.com/recaptcha/docs/invisible) and checkbox (https://developers.google.com/recaptcha/docs/display) verification modes, and v3 (https://developers.google.com/recaptcha/docs/v3).

Prerequisites

Before enabling the feature, a valid site and secret pair should be obtained from Google reCAPTCHA system, by registering the DSpace application on which verification will be set on reCAPTCHA admin panel (https://www.google.com/recaptcha/admin)

How to enable the feature

Once Google reCAPTCHA site and secret are registered and available, the following properties set in dspace.cfg or local.cfg will enable the CAPTCHA verification

```
registration.verification.enabled = true
```

If version **v2** of Google reCAPTCHA is to be enabled, these properties must be set in configuration:

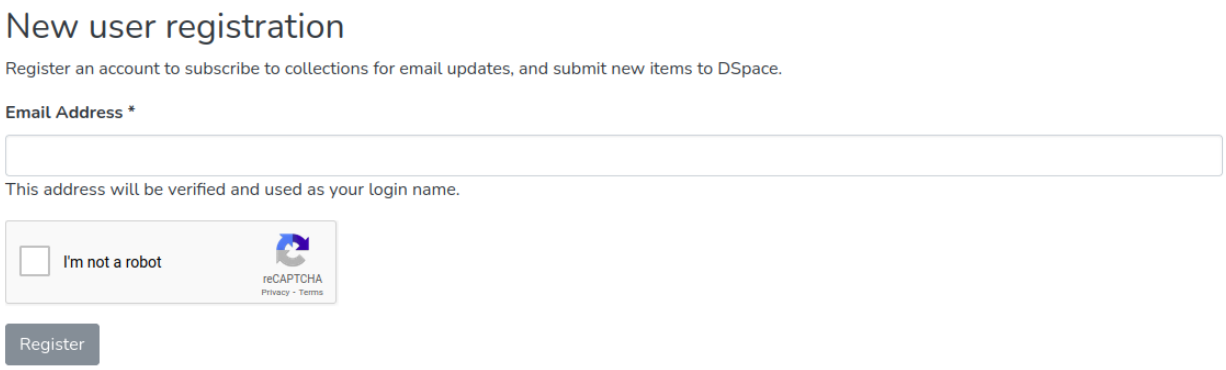
```
google.recaptcha.version = v2
google.recaptcha.mode = <invisible or checkbox depending on which mode is wanted>
google.recaptcha.key.site = <your site here>
google.recaptcha.key.secret = <your secret here>
```

If version **v3** of Google reCAPTCHA is to be enabled, instead the following properties must be set in configuration:

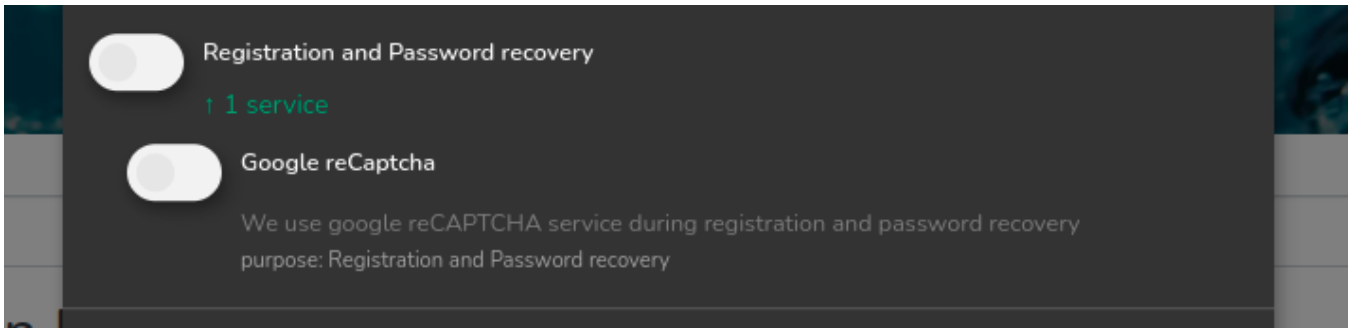
```
google.recaptcha.version = v3
google.recaptcha.key.site = <your site here>
google.recaptcha.key.secret = <your secret here>
google.recaptcha.site-verify = https://www.google.com/recaptcha/api/siteverify
google.recaptcha.key.threshold = <score threshold>
google.recaptcha.mode = invisible
```

The `google.recaptcha.key.threshold` property defines a threshold at which an interaction is considered "not a bot". reCAPTCHA v3 returns a score (1.0 is very likely a good interaction, 0.0 is very likely a bot). By default a good threshold could be 0.5. For further information, see https://developers.google.com/recaptcha/docs/v3#interpreting_the_score

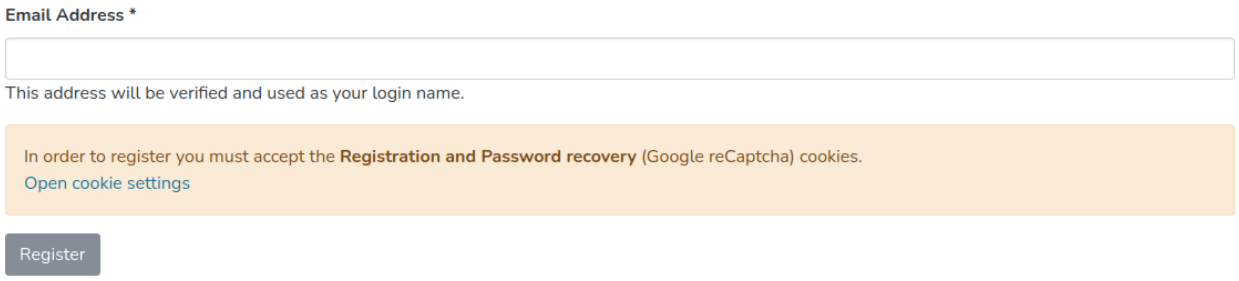
Once this feature is enabled, the user registration will actually be performed if and only if the CAPTCHA token, passed in registration payload, is verified during registration process itself and is considered valid. Each registration request, even if made using DSpace REST APIs must have a captcha token in its header.



A new type of cookie has been added to DSpace cookie set, "Registration and Password Recovery". This cookie is proposed only when CAPTCHA verification is enabled.



This cookie option must be enabled by users before registering, otherwise they won't be able to perform a registration



ALTCHA verification

This feature is avaialble starting from DSpace 9.0

This feature allows the request-a-copy page to be protected by ALTCHA, a GDPR-compliant "Proof of Work" captcha framework, as an alternative to Google reCAPTCHA.

There is no need to register any access keys, and no cookies are used.

How to enable the feature

Set a secret HMAC key in local.cfg or modules/altcha.cfg - this `altcha.hmac.key` property will be used in the computation challenge sent to the browser. It can be any kind of string, like a passphrase.

```
# HMAC key. This is a *secret* key that should be unique to this server
# Recommended use: ovewrite/set the value in local.cfg or environment variable to avoid accidental disclosure
#altcha.hmac.key = YOUR_SECRET_HMAC_KEY
```

How to protect request-a-copy form

To protect the request-a-copy form, set `request.item.create.captcha` to true in local.cfg or modules/requestitem.cfg

```
# Require a captcha for item request creation
# Default: false
request.item.create.captcha=true
```

Configurable Entities

- Introduction
- Default Entity Models
  - Research Entities
  - Journals
- Enabling Entities
  - 1. Configure your entity model (optionally)
  - 2. Import entity model into the database
  - 3. Configure Collections for each Entity type
  - 4. Configure Submission Forms for each Entity type
    - 4.1 Use of collection-entity-type attribute for default Submission forms per Entity Type
  - 5. Configure Workflow for each Entity type (optionally)
  - 6. Configure Virtual Metadata to display for related Entities (optionally)
- Designing your own Entity model
  - Thinking about the object model
  - Configuring the object model
  - Configuring the metadata fields
  - Configuring the item display pages
  - Configuring virtual metadata
  - Configuring discovery
- Additional Technical Details
  - Database Storage
  - Ordering of Relations (place column)
    - Unidirectional example
    - Bidirectional example
  - Tilted relationships
    - Use case: OrgUnit vs Publication relationship
    - Impact of the tilted relationships
- Versioning Support
  - Example of the latest status of a relationship (technical details)
  - Metadata fields that represent relations
  - Configure versioning for an entity type

Introduction

DSpace users have expressed the need for DSpace to be able to provide more support for different types of digital objects related to open access publications, such as authors/author profiles, data sets etc. Configurable Entities are designed to meet that need.

In DSpace, **an Entity is a special type of Item which often has Relationships to other Entities**. Breaking it down with more details...

- **Entity:** Every Entity is an Item.
  - This means they must belong to a Collection, just like a normal Item. (Community & Collection objects are unchanged and unaffected by Entities.)
  - Normal Items are still the "default" Item, and they are unchanged. So, not every Item is an Entity.
  - Because Entities are all Items, they are immediately usable in submission/workflow process, batch import/export, OAI-PMH, etc.
- **Entity (or Item) Type:** Entities all have a "dspace.entity.type" metadata field which defines their Entity/Item "type". For example, this type may be "Person", "Project", "Publication", "Journal", etc. It's highly visible within the User Interface as a label.
- **Relationships:** Based on that "type", an Entity may be related to other Entities via a Relationship. One Entity type may support several relationship types at once. Examples of relationship types include "isPersonOfProject" or "isPublicationOfAuthor". These relationship types are named based on the Entity "type" (as you can likely tell). Relationships also appear on Entities as metadata using the "relation" schema.
- **Virtual Metadata:** Entities of different types may also have customized visualizations in the User Interface. These visualizations may also dynamically pull in metadata from related Entities. For example, a Publication entity may be displayed in the User Interface with an author name dynamically pulled in from a related Person entity. The metadata "appears" as though it is part of the Entity you are viewing, but it is dynamically pulled via the Relationship.

Entities and their Relationships are also completely configurable. DSpace provides some sample models out of the box, which you can use directly or adapt as needed.

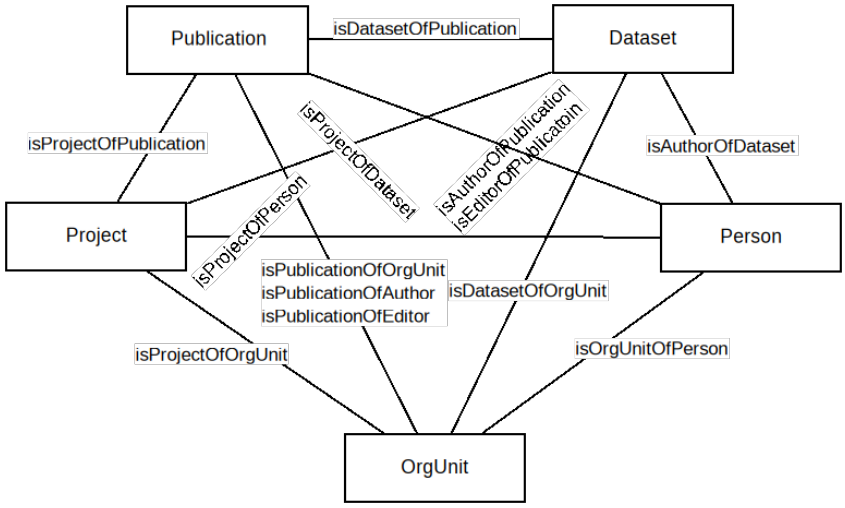
The Entity model also has similarities with the Portland Common Data Model (PCDM), with an Entity roughly mapping to a "pcdm:Object" and existing Communities and Collections roughly mapping to a "pcdm:Collection". However, at this time DSpace Entities concentrate more on building a graph structure of relationships, instead of a tree structure.

Default Entity Models

DSpace currently comes with the following Entity models, both of which are defined in `[dspace]/config/entities/relationship-types.xml`. These Entity models are not used by default, but may be enabled as described below.

Research Entities

Research Entities include Person, OrgUnit, Project, Publication and Dataset. They allow you to create author profiles (Person) in DSpace, and relate those people to their department(s) (OrgUnit), grant project(s) (Project), works (Publication) and research data (Dataset).



- Each publication can link to projects, datasets, people and org units
- Each dataset can link to publications, projects, people and org units.
- Each person can link to projects, publications, datasets and org units
- Each project can link to publications, datasets, people and org units
- Each org units can link to projects, people, publications and datasets

Journals

Journal Entities include Journal, Journal Volume, Journal Issue and Publication (article). They allow you to represent a Journal hierarchy more easily within DSpace, starting at the overall Journal, consisting of multiple Volumes, and each Volume containing multiple Issues. Issues then link to all articles (Publication) which were a part of that journal issue.

NOTE: that this model includes the same "Publication" entity as the Research Entities model described above. This Entity overlap allows you to link an article (Publication) both to its author (Person) as well as the Journal Issue it appeared in.

Enabling Entities

By default, Entities are not used in DSpace. But, as described above several models are available out-of-the-box that may be optionally enabled.

Keep in mind, there are a few DSpace import/export features that do not yet support Entities in DSpace 7.0. These will be coming in future 7.x releases. See DSpace Release 7.0 Status for prioritization information, etc.

- AIP Backup and Restore does not fully support entity types or relationships. In other words, Entities are only represented as normal Items in AIPs
- Importing and Exporting Items via Simple Archive Format does not fully support entity types or relationships. In other words, Entities are only represented as normal Items in SAF. (Note: early work to bring this support is already begun in https://github.com/DSpace/DSpace/pull/3322 which was merged into release 7.1.)
- SWORDv1 Server and SWORDv2 Server does not yet support Entity or relationship creation.

1. Configure your entity model (optionally)

As described above, DSpace provides two default entity models defined in `[dspace]/config/entities/relationship-types.xml`. These models may be used as-is, or modified.

You can also design your own model from scratch (see "Designing your own model" section below). So, feel free to start by modifying `relationship-types.xml`, or creating your own model based on the `relationship-types.dtd`.

A simple XML configuration file is used to determine the relations. This configuration is read into the database (making modeling the data from the UI possible in future versions as well). A sample of how to define a relationship between Publication and Person is shown below:

```
<!-- This relationship defines both the Publication and Person Entities, along with how they are related
together. -->
<relationships>
    <type>
        <leftType>Publication</leftType>
        <rightType>Person</rightType>
        <leftLabel>isAuthorOfPublication</leftLabel>
        <rightLabel>isPublicationOfAuthor</rightLabel>
        <leftCardinality>
            <min>0</min>
            <!--<max></max> not specified, unlimited-->
        </leftCardinality>
        <rightCardinality>
            <min>0</min>
            <!--<max></max> not specified, unlimited-->
        </rightCardinality>
    </type>
</relationships>
```

2. Import entity model into the database

In order to enable a defined entity model, it MUST be imported into the DSpace database This is achieved by using the "initialize-entities" script. The example below will import the "out-of-the-box" entity models into your DSpace installation

```
# The -f command requires a full path to an Entities model configuration file.
[dspace]/bin/dspace initialize-entities -f [dspace]/config/entities/relationship-types.xml
```

If an Entity (of same type name) already exists, it will be updated with any new relationships defined in relationship-types.xml

If an Entity (of same type name) doesn't exist, the new Entity type will be created along with its relationships defined in relationship-types.xml

Once imported into the Database, the overall structure is as follows:

- All valid Entity Types are stored in the "entity_type" database table.
- All Relationship type definitions are stored in the "relationship_type" database table
- All Relationships between entities get stored in the "relationship" table.
- Entities themselves are stored alongside Items in the 'item' table. Every Entity must have a "dspace.entity.type" metadata field whose value is a valid Entity Type (from the "entity_type" table).

Keep in mind, your currently enabled Entity model is defined in your database, and NOT in the "relationship-types.xml". Anytime you want to update your data model, you'd update/create a configuration (like relationship-types.xml) and re-run the "initialize-entities" command.

3. Configure Collections for each Entity type

Because all Entities are Items, they MUST belong to a Collection. Therefore, the *recommended way* to create a different submission forms per Entity type (e.g. Person, Project, Journal, Publication, etc) is to *ensure you create a Collection for each Entity Type* (as each Collection can have a custom Submission Form).

1. Create at least one Collection for each Entity Type needing a custom Submission form. For example, a Collection for "Person" entities, and a separate one for "Publication" entities.
2. Edit the Collection. On the "Edit Metadata" page, use the "Entity Type" dropdown to select the Entity Type for this Collection.
    a. This "Entity Type" selection will ensure that every Item submitted to this collection is automatically assigned that Entity type. So, it ties this Collection to that type of Entity (i.e. *no other type of Entity can be submitted to this Collection).*
        i. *NOTE: Entity Type is currently not modifiable after being set.* This is because changing the Entity type may result in odd behavior (or errors) with in-progress submissions (as they will continue to use the old Entity Type). If you really need to modify the Entity Type, you can do so by changing the "dspace.entity.type" metadata value on the Collection object. At this time, changing that metadata field would need to be done at the database level.
3. In the Edit Collection page, switch to the "Assign Roles" tab and create a "Submitters" group. Add any people who should be allowed to submit /create this new Entity type.
    a. If you only want Administrators to create this Entity type, you can skip this step. Administrators can submit to any Collection.
4. If you want to hide this Collection, you can choose to only make it visible to that same Submitters group (or Administrators). This does NOT hide the Entities from search or browse, but it will hide the Collection itself.
    a. In the Edit Collection page, switch to the "Authorizations" tab.
    b. Add a new Authorization of TYPE_CUSTOM, restricting "READ" to the Submitters group created above (or Administrators if there is no Submitters group). You can also add multiple READ policies as needed. WARNING: The Submitters group MUST have READ privileges to be able to submit/create new Entities.
    c. Remove the default READ policy giving Anonymous permissions.
    d. Assuming you want the Entities to still be publicly available, make sure the DEFAULT_ITEM_READ policy is set to "Anonymous"!

Each of these Collections must be contained in a Community. The simplest thing to do would be to create a new top-level Community, place your entity Collections there, and (optionally) hide the Community in a fashion similar to "hide this Collection" above.

Obviously, how you organize your Entity Types into Collections is up to you. You can create a single Collection for all Entities of that type (e.g. an "Author Profiles" collection could be where all "Person" Entities are submitted/stored). Or, you could create many Collections for each Entity Type (e.g. each Department in your University may have it's own Community, and underneath have a "Staff Profiles" Collection where all "Person" Entities for that department are submitted/stored). A few example structures are shown below.

Example Structure based on the departments:

- Department of Architecture
  - Building Technology Program
  - Theses - Department of Architecture
- Department of Biology
  - Theses - Biology
  - Research Data - Biology
- People
- Projects

OR

- Department of Architecture
  - Building Technology Program
  - Theses - Department of Architecture
  - People in Department of Architecture
  - Projects in Department of Architecture
- Department of Biology
  - Theses - Biology
  - Research Data - Biology
  - People in Department of Biology
  - Projects in Department of Biology

Example Structure based on the publication type:

- Books
  - Book Chapter
  - Edited Volume
  - Monograph
- Theses
  - Bachelor Thesis
  - Doctoral Thesis
  - Habilitation Thesis
  - Master Thesis
- Research Data
- People
- Projects

4. Configure Submission Forms for each Entity type

You should have already created Entity-specific Collections in the previous step. Now, we just need to map those Collections to Submission processes specific to each Entity.

On the backend, you will now need to modify the `[dspace]/config/item-submission.xml` to "map" this Collection (or Collections) to the submission process for this Entity type.

- DSpace comes with sample submission forms for each Entity type.
  - The sample <submission-process> is defined in `item-submission.xml` and named based on the Entity type (e.g. Publication, Dataset, Person, Project, etc).
  - The metadata fields captured for each Entity are defined in a custom step in `submission-forms.xml`, and named in the format "[entityType]Step" (where the entity type is camelcased). For example: "publicationStep", "datasetStep", "personStep", "projectStep".
- Optionally, modify those sample submission forms. See Submission User Interface for hints/tips on customizing the `item-submission.xml` or `submission-forms.xml` files
- As of 7.6, you can simply map each Entity Type to a specific submission form as follows in your `item-submission.xml` (This section already exists, just uncomment it)

```
<name-map collection-entity-type="Publication" submission-name="Publication"/>
<name-map collection-entity-type="Dataset" submission-name="Dataset"/>
<name-map collection-entity-type="Person" submission-name="Person"/>
<name-map collection-entity-type="Project" submission-name="Project"/>
<name-map collection-entity-type="OrgUnit" submission-name="OrgUnit"/>
<name-map collection-entity-type="Journal" submission-name="Journal"/>
<name-map collection-entity-type="JournalVolume" submission-name="JournalVolume"/>
<name-map collection-entity-type="JournalIssue" submission-name="JournalIssue"/>
```

  - WARNING: If you create a new Collection using a specific Entity Type, you must currently restart your servlet container (e.g. Tomcat) for the submission form configuration to take effect for the new Collection. This is the result of a known bug where the Submission forms are cached until the servlet container is restarted. See this issue ticket: https://github.com/DSpace/DSpace/issues/7985

- In 7.5 and earlier, you needed to map each Collection's handle one by one to a Submission form in `item-submission.xml`. Map your Collection's handle (findable on the Collection homepage) to the submission form you want it to use. In the below example, we've mapped a single Collection to each of the out-of-the-box Entity types.

```
<name-map collection-handle="123456789/5" submission-name="Publication"/>
<name-map collection-handle="123456789/6" submission-name="Dataset"/>
<name-map collection-handle="123456789/7" submission-name="Person"/>
<name-map collection-handle="123456789/8" submission-name="Project"/>
<name-map collection-handle="123456789/9" submission-name="OrgUnit"/>
<name-map collection-handle="123456789/28" submission-name="Journal"/>
<name-map collection-handle="123456789/29" submission-name="JournalVolume"/>
<name-map collection-handle="123456789/30" submission-name="JournalIssue"/>
```

Once your modifications to the submission process are complete, you will need to quickly reboot Tomcat (or your servlet container) to reload the current settings.

4.1 Use of collection-entity-type attribute for default Submission forms per Entity Type

Alternatively to a collection's Handle, Entities Types can be used as an attribute. So, instead of specifying the collection handle, you will need to use the `collection-entity-type` attribute and what Entity Type to use (like: Person, Project). Please mind that your Collections with Entity Type need to be previously created.

```
<name-map collection-entity-type="Publication" submission-name="Publication"/>
<name-map collection-entity-type="Dataset" submission-name="Dataset"/>
<name-map collection-entity-type="Person" submission-name="Person"/>
<name-map collection-entity-type="Project" submission-name="Project"/>
<name-map collection-entity-type="OrgUnit" submission-name="OrgUnit"/>
<name-map collection-entity-type="Journal" submission-name="Journal"/>
<name-map collection-entity-type="JournalVolume" submission-name="JournalVolume"/>
<name-map collection-entity-type="JournalIssue" submission-name="JournalIssue"/>
```

Once your modifications to the submission process are complete, you will need to quickly reboot Tomcat (or your servlet container) to reload the current settings.

For DSpace 7.6 release it requires Tomcat Restart for every new collection

Due to the way SubmissionConfigReader is loaded into memory (on a initialize process) currently there is no implemented way to reload submission forms. So, every time you assign an entity type to a collection, or create a new collection with an associated entity type, **you will need to do a Tomcat restart** for that collection to be available at the item submission config. There is an on going fix for that.

DSpace 7.6.1 introduced a fix and you don't need to do a Tomcat Restart anymore

DSpace 7.6.1 adds a way to reload Submission Configs, so you no longer need to do a Tomcat Restart after creating a new collection with an entity type, or assigning to a existing one.

5. Configure Workflow for each Entity type (optionally)

The DSpace workflow can be used for reviewing all objects in the Object Model since these objects are all Items, and separate collections can be used. The workflow used for e.g. a Person Object can be configured to be identical to a publication, different from a publication, or use no workflow at all.

See Configurable Workflow for more information on configuring workflows per Collection.

6. Configure Virtual Metadata to display for related Entities (optionally)

"Virtual Metadata" is metadata that is dynamically determined (at the time of access) based on an Entity's relationship to other Entities. A basic example is displaying a Person Entity's name in the "dc.contributor.author" field of a related Publication Entity. That "dc.contributor.author" field doesn't actually exist on the Publication, but is dynamically added as "virtual metadata" simply because the Publication is linked to the Person (via a relationship).

Virtual Metadata is configurable for all Entities and all relationships. DSpace comes with default settings for its default Entity model, and those can be found in `[dspace]/config/spring/api/virtual-metadata.xml`. In that Spring Bean configuration file, you'll find a map of each relationship type to a metadata field & its value. Here's a summary of how it works:

- The "org.dspace.content.virtual.VirtualMetadataPopulator" bean maps every Relationship type (from `relationship-types.xml`) to a <util: map> definition (of a given ID) also in the `virtual-metadata.xml`

```
<!-- For example, the isAuthorOfPublication relationship is linked to a map of ID
"isAuthorOfPublicationMap" -->
<entry key="isAuthorOfPublication" value-ref="isAuthorOfPublicationMap"/>
```

- That <util:map> defintion defines which DSpace metadata field will store the virtual metadata. It also links to the bean which will dynamically define the value of this metadata field.

```
<!-- In this example, isAuthorOfPublication will be displayed in the "dc.contributor.author" field -->
<!-- The *value* of that field will be defined by the "publicationAuthor_author" bean -->
<util:map id="isAuthorOfPublicationMap">
    <entry key="dc.contributor.author" value-ref="publicationAuthor_author"/>
</util:map>
```

- A bean of that ID then defines the value of the field, based on the related Entity. In this example, these fields are pulled from the related Person entity and concatenated. If the Person has "person.familyName=Jones" and "person.givenName=Jane", then the value of "dc.contributor.author" on the related Publication will be dynamically set to "Jones, Jane.

```
 <bean class="org.dspace.content.virtual.Concatenate" id="publicationAuthor_author">
    <property name="fields">
        <util:list>
            <value>person.familyName</value>
            <value>person.givenName</value>
            <value>organization.legalName</value>
        </util:list>
    </property>
    <property name="separator">
        <value>, </value>
    </property>
    <property name="useForPlace" value="true"/>
    <property name="populateWithNameVariant" value="true"/>
</bean>
```

If the default Virtual Metadata looks good to you, no changes are needed. If you make any changes, be sure to restart Tomcat to update the bean definitions.

Designing your own Entity model

When using a different entities model, the new model has to be configured an loaded into your repository

Thinking about the object model

First step: identify the entity types

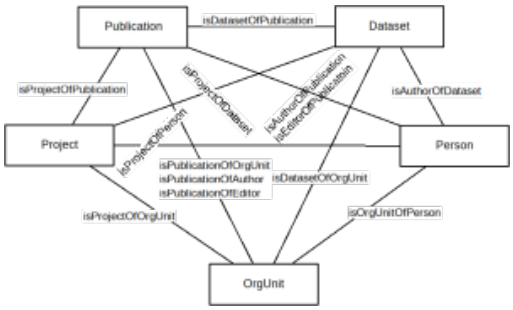
- Which types of objects would you want to create items for: e.g. Person, Publication, JournalVolume
- Be careful not to confuse a type with a relationship. A Person is a type, an author is a relationship between the publication and the person

Second step: identify the relationship types

- Which relationship types would you want to create between the entity items from the previous step: e.g. isAuthorOfPublication, isEditorOfPublication, isProjectOfPublication, isOrgUnitOfPerson, isJournalIssueOfPublication
- Multiple relationships between the same 2 types can be created: isAuthorOfPublication, isEditorOfPublication
- Relationships are automatically bidirectional, so no need to worry about whether you want to display the authors in a publication or the publications of an author

Third step: visualize your model

- By creating a drawing of your model, you'll be able to quickly verify whether anything is missing



Configuring the object model

Configure the model in relationship-types.xml

- Similar to the default relationship-types.xml, configure a relationship type per connection between two Entity Types
- Include the two Entity Type names which are being connected.
- Determine a clear an unambiguous name for the relation in both directions
- Optionally: determine the cardinality (min/max occurrences) for the relationships
- Optionally: determine default behavior for copying metadata if the relationship is deleted

Configuring the metadata fields

Determining the metadata fields to use

- Dublin Core works for publications, but not for a Person, JournalVolume, …
- There are many standards which can be easily configured: schema.org, eurocris, datacite, …
- Pick a schema which suits your needs

Configure the submission forms

- Add a form in submission-forms.xml for each entity type, containing the relevant metadata fields
- See also Submission User Interface documentation.
- Configure which relationships to create

Configuring the item display pages

- The metadata configuration is not specific to configurable entities.
- Similar to other customizations to the item display pages, configure in Angular which metadata fields to display and their label. A template per entity type can be created
- The relationship display is similar to the metadata configuration
- Similar to the metadata configuration: configure in Angular which relationship to display and their label

Configuring virtual metadata

- The isAuthorOfPublication relationship can be displayed for the Publication item as dc.contributor.author
- The isOrgUnitOfPerson relationship can be displayed for the Person item as organization.legalName
- This can be configured in virtual-metadata.xml

Configuring discovery

- Configure the discovery facets, filters, sort options, …
  - The facets for a Person can be job title, organization, project, …
  - The filters for a Person can be person.familyName, person.givenName, …

Additional Technical Details

The original Entities design document is available in Google Docs at: https://docs.google.com/document/d/1wEmHirFzrY3qgGtRr2YBQwGOvH1IuTVGmxDIdnqvwxM/edit While this content has been copied into the DSpace wiki, the file remains available as needed.

A talk on Configurable Entities was also presented at DSpace 7 at OR2021

Database Storage

The actual type of entity is stored in the "dspace.entity.type" metadata field.

Flyway will create tables to store the entity type and relationship configuration. The entity type table contains a unique ID and name for the entity type. This table typically only contains a few rows. It uses database indexes to easily and quickly find the entity types.

Table "entity_type"

```
 Column |         Type          | Modifiers
--------+-----------------------+-----------
 id     | integer               | not null
 label  | character varying(32) | not null
```

The relationship type table contains a unique ID, the 2 entity type IDs (foreign keys), the labels for the relation, and the cardinality details (min and max occurrences in each direction). This table typically only contains a few rows. It uses database indexes to easily and quickly find the relations types.

Table "relationship_type"

```
        Column         |         Type          | Modifiers
-----------------------+-----------------------+-----------
 id                    | integer               | not null
 left_type             | integer               | not null
 right_type            | integer               | not null
 left_label            | character varying(32) | not null
 right_label           | character varying(32) | not null
 left_min_cardinality  | integer               |
 left_max_cardinality  | integer               |
 right_min_cardinality | integer               |
 right_max_cardinality | integer               |
```

The actual relations are stored in another new database table, containing a row per relation between two entities.

Table "relationship"

```
   Column    |  Type   | Modifiers
-------------+---------+-----------
 id          | integer | not null
 left_id     | uuid    | not null
 type_id     | integer | not null
 right_id    | uuid    | not null
 left_place  | integer |
 right_place | integer |
```

It contains a unique ID, the ID of the relationship type (a foreign key to the table above), and the left and right item (foreign keys to the actual items), and place values to keep track of the position of the entity for the given relationship type. It uses database indexes to easily and quickly find all relations from the current item.

Ordering of Relations (place column)

The “place” columns for relations are similar to the "place" column in the metadatavalue table, which also determines the order of the values. It gets updated automatically with the next number if a relation is created, similar to the current solution to populate the "place" column in the metadatavalue table.

The reason why there's a left and right place separately, is because we don't want the model to restrict which part of the relation has a "place".

Unidirectional example



For a journal, there are relations to the journal volumes as visible in REST. The relevant section is:

https://demo.dspace.org/entities/journalvolume/f9b89a11-b44e-4a64-a3b4-ab24a33553c7

"leftId": "a23eae5a-7857-4ef9-8e52-989436ad2955",
"rightId": "f9b89a11-b44e-4a64-a3b4-ab24a33553c7",
"leftPlace": 1,
"rightPlace": 1,

https://demo.dspace.org/entities/journalvolume/343d3263-2733-4367-9dc4-216a01b4a461

"leftId": "a23eae5a-7857-4ef9-8e52-989436ad2955",
"rightId": "343d3263-2733-4367-9dc4-216a01b4a461",
"leftPlace": 2,
"rightPlace": 1,

For this particular relation, the leftPlace is relevant because the leftId contains the current item ID. First the item with leftPlace 1 is displayed. Hereafter the item with leftPlace 2 is displayed.

Bidirectional example



For an article, there are relations to the OrgUnits visible in REST. The relevant section is:

https://demo.dspace.org/entities/orgunit/d30de96b-1e76-40ae-8ef9-ab426b6f9763

"leftId": "96715576-3748-4761-ad45-001646632963",
"rightId": "d30de96b-1e76-40ae-8ef9-ab426b6f9763",
"leftPlace": 1,
"rightPlace": 2,

https://demo.dspace.org/entities/orgunit/c216201f-ed10-4361-b0e0-5a065405bd3e

"leftId": "96715576-3748-4761-ad45-001646632963",
"rightId": "c216201f-ed10-4361-b0e0-5a065405bd3e",
"leftPlace": 2,
"rightPlace": 2,

For this particular relation, the leftPlace is relevant because the leftId contains the current item ID. First the item with leftPlace 1 is displayed. Hereafter the item with leftPlace 2 is displayed.

In the other direction, on the OrgUnit item page there are 6 publications visible in REST. The relevant section is:

https://demo.dspace.org/entities/publication/e98b0f27-5c19-49a0-960d-eb6ad5287067

"leftId": "e98b0f27-5c19-49a0-960d-eb6ad5287067",
"rightId": "d30de96b-1e76-40ae-8ef9-ab426b6f9763",
"leftPlace": 1,
"rightPlace": 1,

https://demo.dspace.org/entities/publication/96715576-3748-4761-ad45-001646632963

"leftId": "96715576-3748-4761-ad45-001646632963",
"rightId": "d30de96b-1e76-40ae-8ef9-ab426b6f9763",
"leftPlace": 1,
"rightPlace": 2,

https://demo.dspace.org/entities/publication/2f4ec582-109e-4952-a94a-b7d7615a8c69

"leftId": "2f4ec582-109e-4952-a94a-b7d7615a8c69",
"rightId": "d30de96b-1e76-40ae-8ef9-ab426b6f9763",
"leftPlace": 2,
"rightPlace": 3,

For this particular relation, the rightPlace is relevant because the rightId contains the current item ID. First the item with rightPlace 1 is displayed. Hereafter the item with rightPlace 2 is displayed, …

The relation with rightPlace 2 is the same relation as mentioned for the article above

The default DSpace database tables will not need to be modified as the entity type is part of the regular metadata.

Tilted relationships

The tilted relationships are a default DSpace 7 feature, developed in https://github.com/DSpace/DSpace/pull/3134

It's designed to improve performance when an entity has 1000s of relationships. It will avoid loading the relationships in the configured direction unless explicitly requested to retrieve them. It's mainly used for setups where there are so many related items that it doesn't make sense to list them all, and they are rather made available via search. Using tilted relationships, those setups can get a big performance boost.

The behavior can most easily be explained using a real-life example

Use case: OrgUnit vs Publication relationship

DSpace 7 has an `isOrgUnitOfPublication` relationship type defined by default. When there are OrgUnits with 1000s or more Publications linked to them, the default `isOrgUnitOfPublication` relationship can run into performance issues.

When setting the tilted to `left`, where the left entity type is `Publication` and the right entity type is `OrgUnit`, you are specifying that the `Publication` should still load the related `OrgUnit`s (expected to be a small amount anyway), but the `OrgUnit` is not supposed to load all the related `Publication`s.

The implications of this setup are:

- When checking the `OrgUnit` metadata, it won't populate the `relation.isPublicationOfOrgUnit` virtual metadata
- When checking the `OrgUnit` relationships, it won't return the relationships based on this type
- But using the relationship search angular component, the related `Publication`s can still be displayed on the `OrgUnit` item page
- It is also still possible to use the relationship list angular component on the `OrgUnit` item page without a performance impact, but for the `OrgUnit` entities, the pageable list doesn't have much added value
- When checking the `Publication` metadata, it will still populate the `relation.isOrgUnitOfPublication` virtual metadata, and can also populate any other virtual metadata such as the OrgUnit name on the Publication
- When checking the `Publication` relationships, it will include the relationships based on this type
- The relationship list angular component on the `Publication` item page can be used, and won't have any issues

Impact of the tilted relationships

Performance tests were performed in https://github.com/DSpace/DSpace/pull/3134 (some tests were specific to improvements which are independent from tilted relationships, and which are already applied in all cases)

In general:

- Retrieving the item REST representation which includes virtual metadata when using the tilted relationships for items with 600 relationships reduced the duration to 5% (so a 95% reduction in time spent)

- Creating new relationships when using the tilted relationships for items with 1000 relationships or more reduced the duration as well, but more recent tests revealed this reduced the duration to 25% (so a 75% reduction in time spent)

Using tilted relationships, there are production environments with single entities having over 40k relationships using a single relationship type. Using the tilted relationships, this setup doesn't impact the performance.

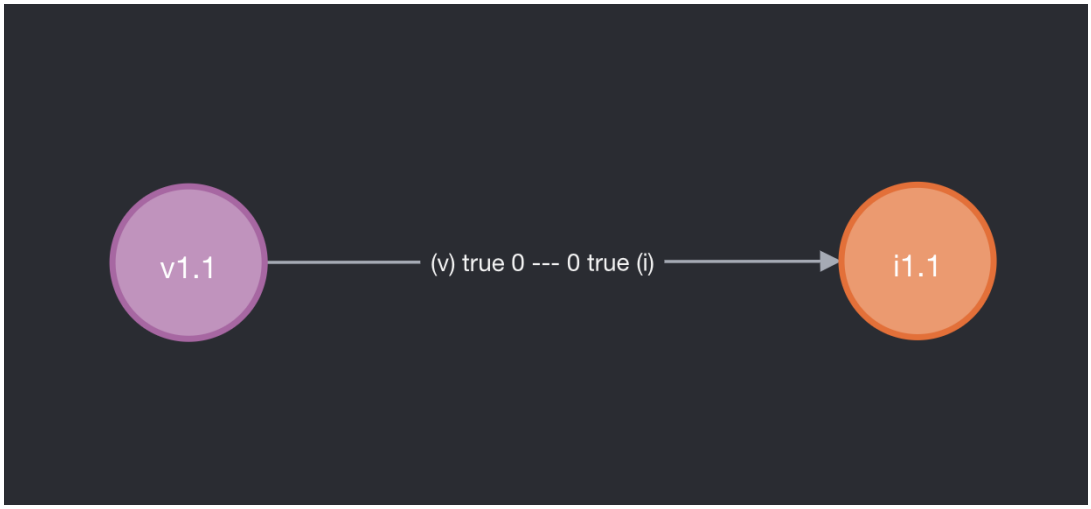
Using the OrgUnit to Person relationship, it's possible to have an OrgUnit with 40k Person entities, and the virtual metadata of the Person entity contains the Org Unit's name, ID, ….

Versioning Support

DSpace entities **fully support versioning**. For the most part, this works like any other item. For example, when creating a new version of an item, a new item is created and all metadata values of the preceding item are copied over to the new item. Special care was taken to version relationships between entities.

Example of the latest status of a relationship (technical details)

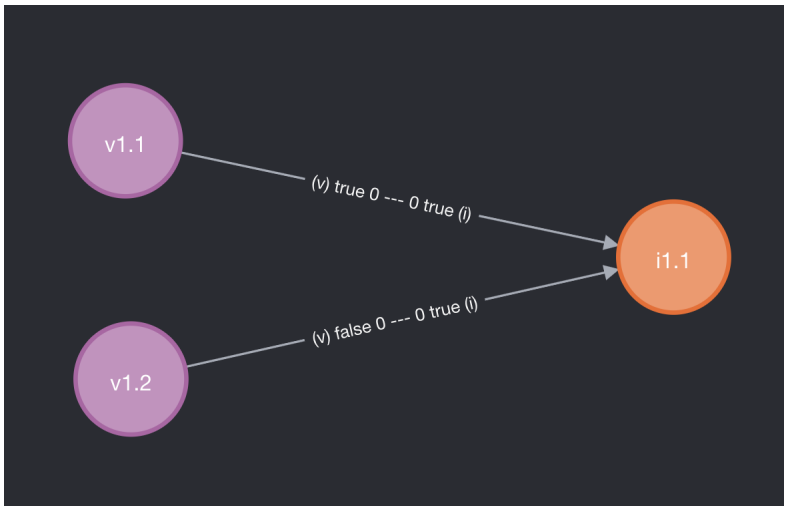
To understand how versioning between entities with relationships works, let's walk through the following example:



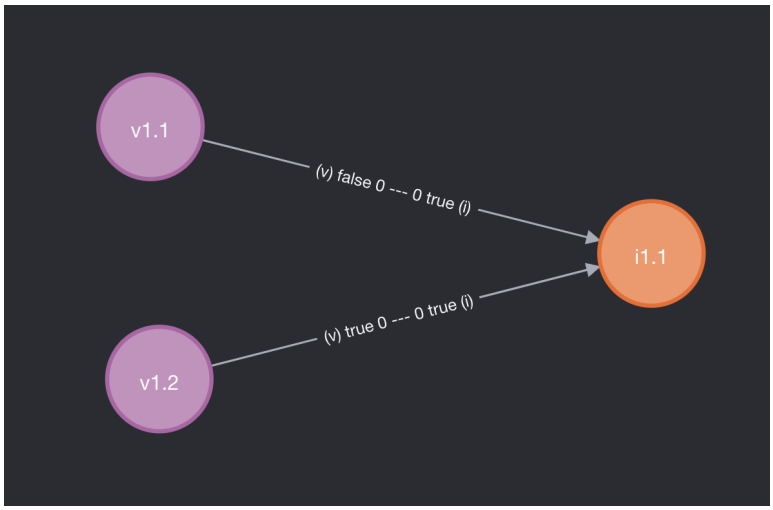
Consider Volume 1.1 (left side) and Issue 1.1 (right side). Both are archived and both are the first version. Note that on the arrow, representing the relation between the volume and the issue, two booleans and two numbers are indicated.

- The boolean on side `(v)` is true if and only if volume 1.1 is the latest version that is relevant to issue 1 (even though it may be possible that volume 1.2, the second version of volume 1, exists). This means that on the item page of issue 1.1, a link to the item page of volume 1.1 should be displayed. It also means that searching for (the uuid of) issue 1.1 should yield volume 1.1.
- The boolean on side `(i)` is true if and only if issue 1.1 is the latest version that is relevant to volume 1.1 (even though it may be possible that issue 1.1, the second version of issue 1, exists). This means that on the item page of volume 1.1, a link to the item page of issue 1.1 should be displayed. It also means that searching for (the uuid of) volume 1.1 should yield issue 1.1.
- The number on side `(v)` indicates the place at which the virtual metadata representing this relationship (if any) will appear on volume 1.1. E.g. using the out-of-the-box configuration in `virtual-metadata.xml`, metadata field `publicationissue.issueNumber` of issue 1.1 would appear as metadata field `publicationissue.issueNumber` on volume 1.1 on place 0 (i.e. as the first metadata value).
- The number on side `(i)` indicates the place at which the virtual metadata representing this relationship (if any) will appear on issue 1.1. E.g. using the out-of-the-box configuration in `virtual-metadata.xml`, metadata field `publicationvolume.volumeNumber` of volume 1.1 would appear as metadata field `publicationvolume.volumeNumber` on issue 1.1 on place 0 (i.e. as the first metadata value).

With the groundwork out of the way, let's see what happens when we create a new version of volume 1.1. The new version is not yet archived, because it still has to be edited in the submission UI.

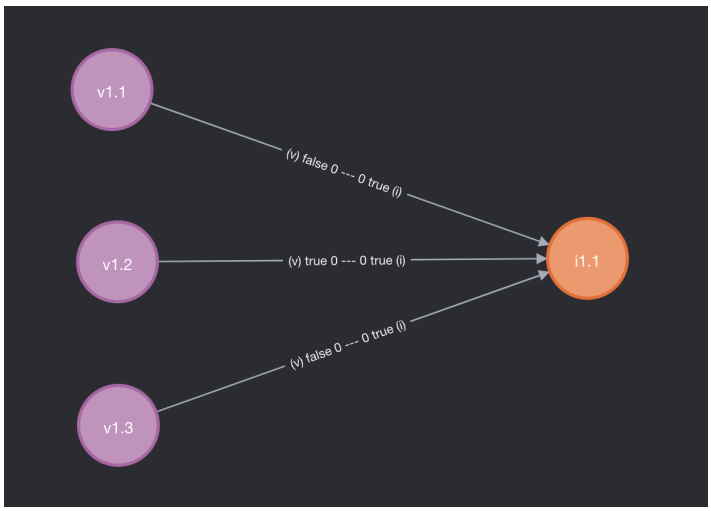


At this moment, when viewing the item page of issue 1.1, the user should only see volume 1.1 (as volume 1.2 is not yet archived). When viewing the item page of volume 1.1, nothing has changed: only a link to issue 1.1 will appear. When viewing the item page of volume 1.2 (e.g. as an admin), a link to issue 1.1 will appear as well.

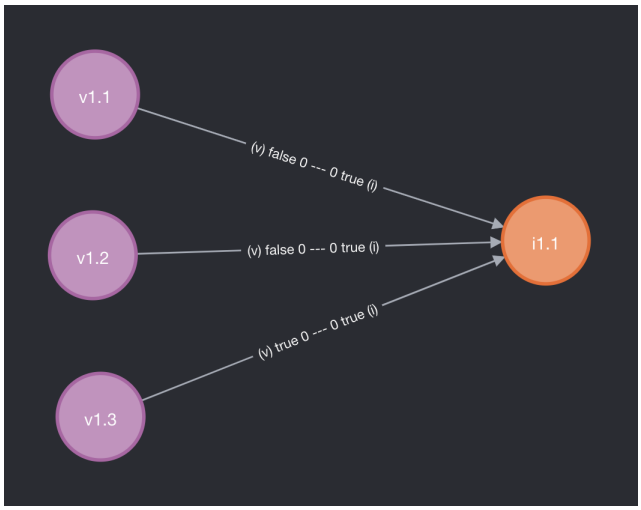


As soon as volume 1.2 is deposited (archived), the "latest status" of both volume 1.1 and volume 1.2 are updated. When viewing the item page of issue 1.1, volume 1.2 should be visible. When viewing the item pages of the volumes, nothing has changed.

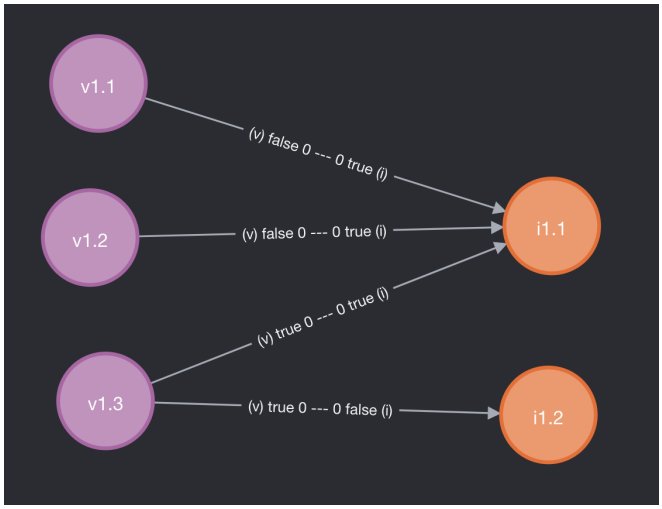
Let's create another version of the volume (not archived):



And after archiving volume 1.3:



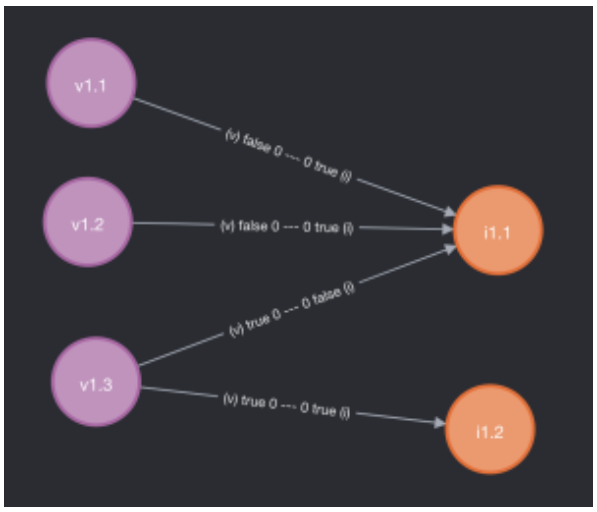
What happens if we create a new version of issue 1.1?



Only the relationship with volume 1.3 is copied. For issue 1.1, no relationship was displayed with volume 1.1 and 1.2. (The relationships still exist in the database, but are not visible in the UI.). For volume 1.1, a relationship to issue 1.1 remains present, but it should not be updated to issue 1.2. For issue 1.2, these relationships are longer relevant, so they are not copied.

On the item pages of volume 1.1, volume 1.2 and volume 1.3, you should see issue 1.1 (as 1.2 is not archived yet)

Because issue 1.2 is not yet archived, all volumes are still pointing to issue 1.1. Let's archive it:



Now on the item pages of volume 1.1 and volume 1.2, you should see issue 1.1; it's the latest issue at the time that those volumes were superseded by volume 1.3. On the item page of volume 1.3, you'll see issue 1.3. On the item page of issue 1.1 you'll still see volume 1.3 as well.

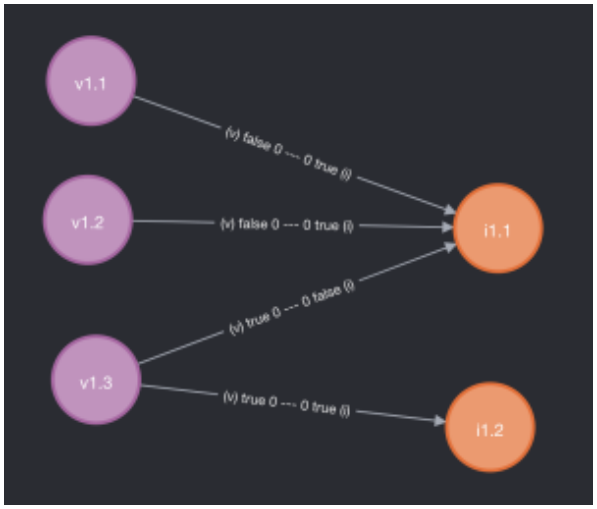
Metadata fields that represent relations

If you have a closer look at items with relationships, you'll notice two categories of metadata fields that are controlled by DSpace:

- `relation.*` fields, for example `relation.isIssueOfJournalVolume` on volume items
- `relation.*.latestForDiscovery` fields, for example `relation.isIssueOfJournalVolume.latestForDiscovery` on volume items

Metadata fields of the first category (`relation.*`) contain all uuids of related items that the current item can see. I.e. a relationship has to exist between the current item and the other item, and the other item needs to have "latest status" for that specific relationship.

As an example take the following state of the previous section:



Item issue 1.1 will contain metadata field `relation.isJournalVolumeOfIssue` with as value the uuid of volume 1.3. Volume 1.1 and 1.2 are not included because they don't have "latest status" on the relevant relationships.

Metadata fields of the second category (`relation.*.latestForDiscovery`) contain all uuids of the items for which the current item is visible. I.e. a relationship has to exist between the current item and the other item, and the current item needs to have "latest status" for that specific relationship. These fields are particularly important for indexing and search, because they allow to us to surface all the items that a particular item is referring to.

Continuing on the example above, issue 1.1 will have metadata field `relation.isJournalVolumeOfIssue.latestForDiscovery` containing the uuids of volume 1.1 and 1.2.

With issue 1.1 containing volume 1.1 and 1.2 in `relation.isJournalVolumeOfIssue.latestForDiscovery`, a search on the volume 1.1 page for all issues containing volume 1.1 will display issue 1.1 thanks to this setup.

Configure versioning for an entity type

DSpace contains a bunch of example entity types that support versioning out of the box. What follows is an overview of the requirements to make entity versioning work.

1. when introducing a relationship type, make sure to add four new metadata fields to `config/registries/relationship-formats.xml`. E.g. `relation.isAuthorOfPublication`, `relation.isAuthorOfPublication.latestForDiscovery`, `relation.isPublicationOfAuthor` and `relation.isPublicationOfAuthor.latestForDiscovery`
2. when introducing an entity type, filter items on `latestVersion:true` in `discovery.xml`. This will be the default search, which ensures older versions are not shown
    - If you want to show all related items, including older versions, you can create another discovery config without `latestVersion:true`. This should be used for item pages displaying the related items to the current item using the discovery search.
    - The entity types configured out-of-the-box have discovery config `<entity-type>` and discovery config `<entity-type>Relationships` for that purpose.

Note that versioning support is enabled by default, but can be turned off by setting `versioning.enabled = false` in `versioning.cfg` or `local.cfg` . For more details on item versioning, see: https://wiki.lyrasis.org/display/DSDOC7x/Item+Level+Versioning.

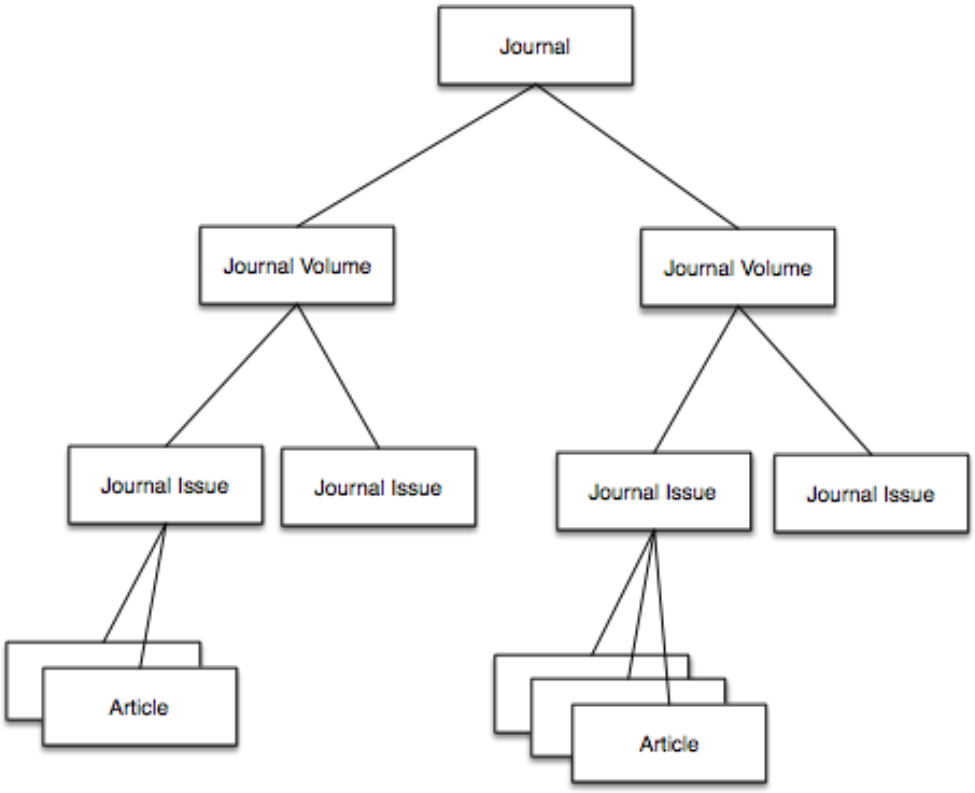
Entities Use Case: Compound Journal

This page describes an out-of-the-box Configurable Entities use case which is to represent a "Compound Journal" hierarchy.

Structure

The structure displays the types of entities, and the relations between them.

Each Journal consists of multiple volumes, and each volume contains multiple issues.



Data Model Storage

Storage in the database

The entity types represent the 3 blocks from the structure.

The labels displayed here will also be stored in the item metadata to easily identify the type of item.

```
select * from entity_type;

 id |     label
----+---------------
  1 | Publication
  5 | Journal
  6 | JournalVolume
  7 | JournalIssue
```

The relationship types are the lines connecting the blocks from the structure. The cardinality columns have been omitted in this document to reduce the size of the table. The labels displayed here will also be used for e.g. the DSpace CSV import to build relations and for displaying the relations in REST.

```
select id, left_type, right_type, left_label, right_label from relationship_type;

 id | left_type | right_type |       left_label            |      right_label
----+-----------+------------+-----------------------------+------------------------
  7 |         5 |          6 | isVolumeOfJournal           | isJournalOfVolume
  8 |         6 |          7 | isIssueOfJournalVolume      | isJournalVolumeOfIssue
  9 |         7 |          1 | isPublicationOfJournalIssue | isJournalIssueOfPublication
```

Data Model Design in XML

The XML file designed to create the rows in the database mentioned above.

```
<?xml version="1.0" encoding="UTF-8"?>

<relationships>
    <type>
        <leftType>Journal</leftType>
        <rightType>JournalVolume</rightType>
        <leftLabel>isVolumeOfJournal</leftLabel>
        <rightLabel>isJournalOfVolume</rightLabel>
        <leftCardinality>
            <min>0</min>
            <!--<max></max> not specified, unlimited-->
        </leftCardinality>
        <rightCardinality>
            <min>1</min> <!--each volume should have one journal-->
            <max>1</max> <!--each volume should have one journal-->
        </rightCardinality>
    </type>
    <type>
        <leftType>JournalVolume</leftType>
        <rightType>JournalIssue</rightType>
        <leftLabel>isIssueOfJournalVolume</leftLabel>
        <rightLabel>isJournalVolumeOfIssue</rightLabel>
        <leftCardinality>
            <min>0</min>
        </leftCardinality>
        <rightCardinality>
            <min>1</min> <!--each issue should have one volume—>
            <max>1</max> <!--each volume should have one journal-->
        </rightCardinality>
    </type>
    <type>
        <leftType>JournalIssue</leftType>
        <rightType>Publication</rightType>
        <leftLabel>isPublicationOfJournalIssue</leftLabel>
        <rightLabel>isJournalIssueOfPublication</rightLabel>
        <leftCardinality>
            <min>0</min>
        </leftCardinality>
        <rightCardinality>
            <min>0</min>
            <max>1</max>
        </rightCardinality>
    </type>
</relationships>
```

Importing this XML using the command:

[install_dir]/bin/dspace dsrun org.dspace.app.util.InitializeEntities -f [install_dir]/config/relationship-types.xml

creates the database tables.

Relation Storage

Storage in the database

The relations are the actual items which are connected, and contain a foreign key to both items, and to the relationship type. The place column has been omitted to reduce the size of the table. The item IDs have been reduced in size to reduce the size of the table.

The table below indicates in the first row that:

the item (a journal)

https://demo.dspace.org/entities/journal/d4af6c3e-53d0-4757-81eb-566f3b45d63a

is related to the item (a journal volume)

https://demo.dspace.org/entities/journalvolume/07c6249f-4bf7-494d-9ce3-6ffdb2aed538

via the relation `isVolumeOfJournal`.

The row ID 60 states the item (a journal volume)

https://demo.dspace.org/entities/journalvolume/07c6249f-4bf7-494d-9ce3-6ffdb2aed538

is related to the item (a journal issue)

https://demo.dspace.org/entities/journalissue/44c29473-5de2-48fa-b005-e5029aa1a50b

via the relation `isIssueOfJournalVolume`.

```
select id, left_id, type_id, right_id from relationship;

 id |               left_id           | type_id |               right_id
----+----------------------------------+---------+--------------------------------
 56 | d4af6c3e-53d0-81eb-566f3b45d63a |       7 | 07c6249f-4bf7-9ce3-6ffdb2aed538
 57 | d4af6c3e-53d0-81eb-566f3b45d63a |       7 | 66bb4e5d-b419-a648-f270a527f17c
 58 | a23eae5a-7857-8e52-989436ad2955 |       7 | f9b89a11-b44e-a3b4-ab24a33553c7
 59 | a23eae5a-7857-8e52-989436ad2955 |       7 | 343d3263-2733-9dc4-216a01b4a461
 60 | 07c6249f-4bf7-9ce3-6ffdb2aed538 |       8 | 44c29473-5de2-b005-e5029aa1a50b
 61 | 07c6249f-4bf7-9ce3-6ffdb2aed538 |       8 | c3076837-e5df-80bc-2661cd390a7b
 62 | 66bb4e5d-b419-a648-f270a527f17c |       8 | a4a63ab5-8c0b-b5f7-5b5d9828cb69
 63 | f9b89a11-b44e-a3b4-ab24a33553c7 |       8 | 77877343-3f75-9492-6ed7c98ed84e
 64 | f9b89a11-b44e-a3b4-ab24a33553c7 |       8 | f4dcd8a6-4cc4-8bb9-a7e8202e05b0
 65 | 343d3263-2733-9dc4-216a01b4a461 |       8 | b7003f66-80e9-99a2-3695e8150b80
 66 | 343d3263-2733-9dc4-216a01b4a461 |       8 | db55298c-a21f-8793-a21f1194a226
141 | 44c29473-5de2-b005-e5029aa1a50b |       9 | e98b0f27-5c19-960d-eb6ad5287067
142 | c3076837-e5df-80bc-2661cd390a7b |       9 | 96715576-3748-ad45-001646632963
143 | a4a63ab5-8c0b-b5f7-5b5d9828cb69 |       9 | 047556d1-3d01-bc68-0cee7ad7ed4e
```

Virtual Metadata

For Journal Volumes, it's relevant to include the ISSN and journal name of the parent journal in the metadata. A sample of this functionality can be seen at e.g. https://demo.dspace.org/server/#https://demo.dspace.org/server/api/core/items/66bb4e5d-b419-42b7-a648-f270a527f17c where `journal.identifier.issn` and `journal.title` is populated with the metadata of the parent journal

For Journal Issues, it's relevant to include the volume number of the parent journal volume in the metadata. A sample of this functionality can be seen at e.g. https://demo.dspace.org/server/#https://demo.dspace.org/server/api/core/items/a4a63ab5-8c0b-4456-b5f7-5b5d9828cb69 where `journalvolume.identifier.volume` is populated with the metadata of the parent journal

Since this is virtual metadata, the metadata is not duplicated, but only displayed.

The configuration of this virtual metadata can be found at https://github.com/DSpace/DSpace/blob/main/dspace/config/spring/api/virtual-metadata.xml

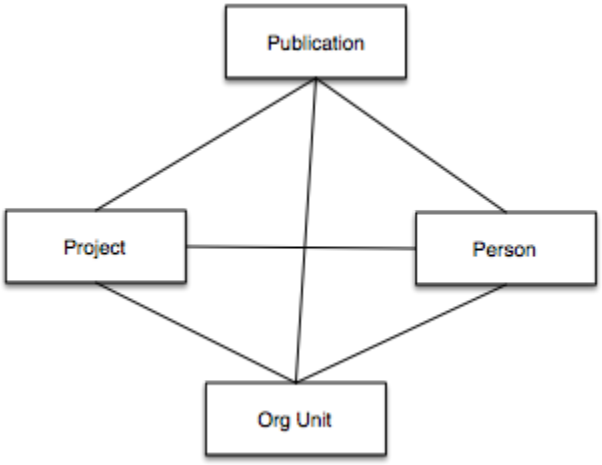
Entities Use Case: Research Management

This page describes an out-of-the-box Configurable Entities use case which is to represent research data similar to a "Current Research Information System" (CRIS) or "Research Information Management System" (RIMS).

Structure

The structure displays the types of entities, and the relations between them.

- Each publication can link to projects, people and org units
- Each person can link to projects, publications and org units
- Each project can link to publications, people and org units
- Each org units can link to projects, people and publications



Model storage

Storage in the database

The entity types represent the 4 blocks from the structure.

The labels displayed here will also be stored in the item metadata to easily identify the type of item.

```
select * from entity_type;

 id |     label
----+---------------
  1 | Publication
  2 | Person
  3 | Project
  4 | OrgUnit
```

The relationship types are the lines connecting the blocks from the structure. The cardinality columns have been omitted in this document to reduce the size of the table. The labels displayed here will also be used for e.g. the DSpace CSV import to build relations and for displaying the relations in REST.

```
select id, left_type, right_type, left_label, right_label from relationship_type;

 id | left_type | right_type |       left_label       |      right_label
----+-----------+------------+------------------------+------------------------
  1 |         1 |          2 | isAuthorOfPublication  | isPublicationOfAuthor
  2 |         1 |          3 | isProjectOfPublication | isPublicationOfProject
  3 |         1 |          4 | isOrgUnitOfPublication | isPublicationOfOrgUnit
  4 |         2 |          3 | isProjectOfPerson      | isPersonOfProject
  5 |         2 |          4 | isOrgUnitOfPerson      | isPersonOfOrgUnit
  6 |         3 |          4 | isOrgUnitOfProject     | isProjectOfOrgUnit
```

Data Model Design in XML

The XML file designed to create the rows in the database mentioned above

```
<relationships>
    <type>
        <leftType>Publication</leftType>
        <rightType>Person</rightType>
        <leftLabel>isAuthorOfPublication</leftLabel>
        <rightLabel>isPublicationOfAuthor</rightLabel>
        <leftCardinality>
            <min>0</min>
            <!--<max></max> not specified, unlimited-->
        </leftCardinality>
        <rightCardinality>
            <min>0</min>
            <!--<max></max> not specified, unlimited-->
        </rightCardinality>
    </type>
    <type>
        <leftType>Publication</leftType>
        <rightType>Project</rightType>
        <leftLabel>isProjectOfPublication</leftLabel>
        <rightLabel>isPublicationOfProject</rightLabel>
        <leftCardinality>
            <min>0</min>
        </leftCardinality>
        <rightCardinality>
            <min>0</min>
        </rightCardinality>
    </type>
    <type>
        <leftType>Publication</leftType>
        <rightType>OrgUnit</rightType>
        <leftLabel>isOrgUnitOfPublication</leftLabel>
        <rightLabel>isPublicationOfOrgUnit</rightLabel>
        <leftCardinality>
            <min>0</min>
        </leftCardinality>
        <rightCardinality>
            <min>0</min>
        </rightCardinality>
    </type>
    <type>
        <leftType>Person</leftType>
        <rightType>Project</rightType>
        <leftLabel>isProjectOfPerson</leftLabel>
        <rightLabel>isPersonOfProject</rightLabel>
        <leftCardinality>
            <min>0</min>
        </leftCardinality>
        <rightCardinality>
            <min>0</min>
        </rightCardinality>
    </type>
    <type>
        <leftType>Person</leftType>
        <rightType>OrgUnit</rightType>
        <leftLabel>isOrgUnitOfPerson</leftLabel>
        <rightLabel>isPersonOfOrgUnit</rightLabel>
        <leftCardinality>
            <min>0</min>
        </leftCardinality>
        <rightCardinality>
            <min>0</min>
        </rightCardinality>
    </type>
    <type>
        <leftType>Project</leftType>
        <rightType>OrgUnit</rightType>
        <leftLabel>isOrgUnitOfProject</leftLabel>
        <rightLabel>isProjectOfOrgUnit</rightLabel>
```

```
        <leftCardinality>
            <min>0</min>
        </leftCardinality>
        <rightCardinality>
            <min>0</min>
        </rightCardinality>
    </type>
</relationships>
```

Importing this file created the database tables

Relation Storage

Storage in the database

The relations are the actual items which are connected, and contain a foreign key to both items, and to the relationship type. The place column has been omitted to reduce the size of the table. The item IDs have been reduced in size to reduce the size of the table.

The table below indicates

in the first row the item (a publication)

https://demo.dspace.org/entities/publication/e98b0f27-5c19-49a0-960d-eb6ad5287067

is related to the item (a person)

https://demo.dspace.org/entities/person/0ffbee3f-e7ea-42bc-92fe-2fbef1a52c0f

via the relation `isAuthorOfPublication`.

```
select id, left_id, type_id, right_id from relationship;

 id |               left_id           | type_id |               right_id
----+----------------------------------+---------+--------------------------------
 52 | e98b0f27-5c19-960d-eb6ad5287067 |       1 | 0ffbee3f-e7ea-92fe-2fbef1a52c0f
 53 | e98b0f27-5c19-960d-eb6ad5287067 |       1 | 5a3f7c7a-d3df-b8a2-f00ede62c60a
 66 | 72635f7f-37b5-b4f2-5ff45d97a09b |       1 | 0ffbee3f-e7ea-92fe-2fbef1a52c0f
 67 | 674f695e-8001-8f9c-095c536a6bcb |       1 | 5a3f7c7a-d3df-b8a2-f00ede62c60a
 68 | a64719f8-ba7b-8eb6-f8feb0c000b7 |       1 | f2235aa6-6fe7-a690-598b72dd8e44
 69 | a64719f8-ba7b-8eb6-f8feb0c000b7 |       1 | 0ffbee3f-e7ea-92fe-2fbef1a52c0f
 70 | e98b0f27-5c19-960d-eb6ad5287067 |       2 | 0de99067-c898-a82c-9555f3311288
 71 | e7bd0d24-e83a-bc0c-8aaaeb19dc7d |       2 | 0de99067-c898-a82c-9555f3311288
 76 | 2f4ec582-109e-a94a-b7d7615a8c69 |       2 | 18e7924c-f15b-9fe3-3de370bccc97
 77 | 99c2e55c-6326-9f36-fcac333b0e8c |       2 | 18e7924c-f15b-9fe3-3de370bccc97
 78 | 96715576-3748-ad45-001646632963 |       2 | 18e7924c-f15b-9fe3-3de370bccc97
 79 | e98b0f27-5c19-960d-eb6ad5287067 |       3 | d30de96b-1e76-8ef9-ab426b6f9763
 80 | e98b0f27-5c19-960d-eb6ad5287067 |       3 | 506a7e54-8d7c-8636-d5f6411483de
 81 | e98b0f27-5c19-960d-eb6ad5287067 |       3 | c216201f-ed10-b0e0-5a065405bd3e
 82 | 96715576-3748-ad45-001646632963 |       3 | d30de96b-1e76-8ef9-ab426b6f9763
 95 | a64719f8-ba7b-8eb6-f8feb0c000b7 |       3 | c216201f-ed10-b0e0-5a065405bd3e
 96 | a64719f8-ba7b-8eb6-f8feb0c000b7 |       3 | d30de96b-1e76-8ef9-ab426b6f9763
 97 | 0de99067-c898-a82c-9555f3311288 |       6 | d30de96b-1e76-8ef9-ab426b6f9763
```

Virtual Metadata

For publications, it's relevant to include the names of the related authors in the metadata. A sample of this functionality can be seen at e.g. https://demo.dspace.org/server/#https://demo.dspace.org/server/api/core/items/e98b0f27-5c19-49a0-960d-eb6ad5287067 where `dc.contributor.author` is populated with the name of the related author. `dc.contributor.other` is also populated with the name of the Org Unit here

For a person, it's relevant to include the names of the org units in the metadata. A sample of this functionality can be seen at e.g. https://demo.dspace.org/server/#https://demo.dspace.org/server/api/core/items/f2235aa6-6fe7-4174-a690-598b72dd8e44 where `person.contributor.other` is populated with the name of the related Org Unit

Since this is virtual metadata, the metadata is not duplicated

The configuration of this virtual metadata can be found at https://github.com/DSpace/DSpace/blob/main/dspace/config/spring/api/virtual-metadata.xml

Content Management System

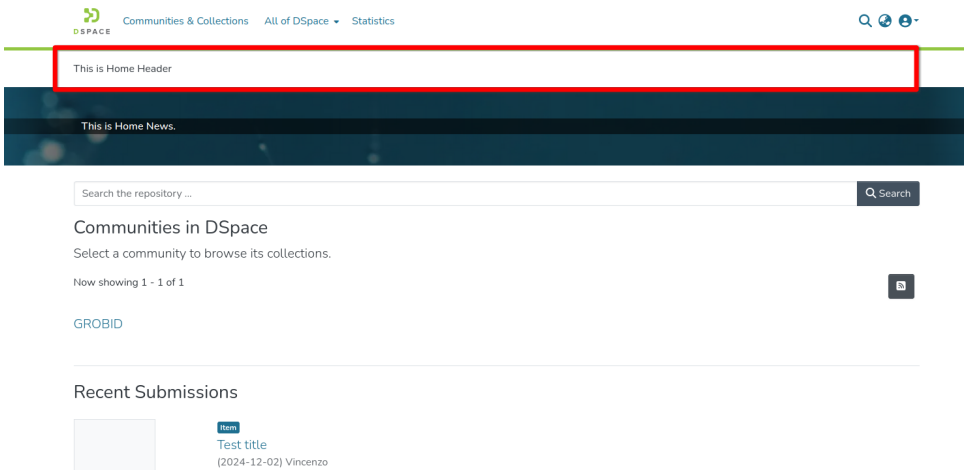
This page offers access to all aspects of the documentation relevant to editing DSpace website content without needing specialized technical knowledge or coding skills.

- Editing CMS metadata
- Editing User Agreement

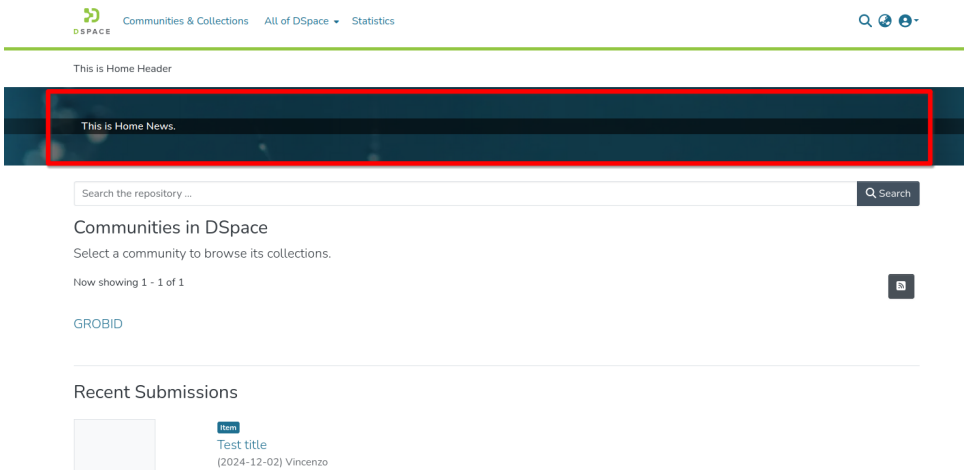
Editing CMS metadata

DSpace 10 allows you to edit the content displayed on the website in the following sections:

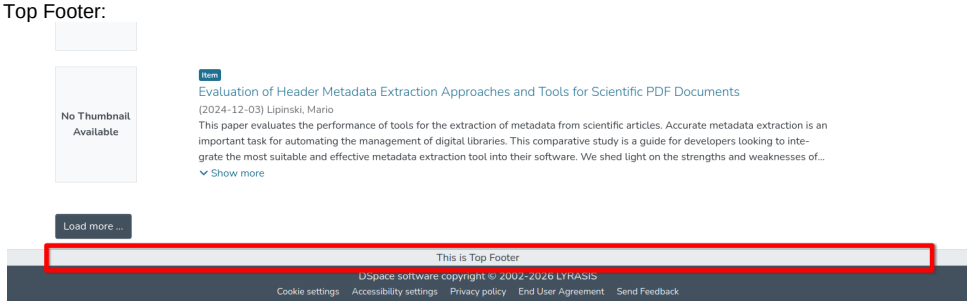
1. Home Header:



2. Home News:

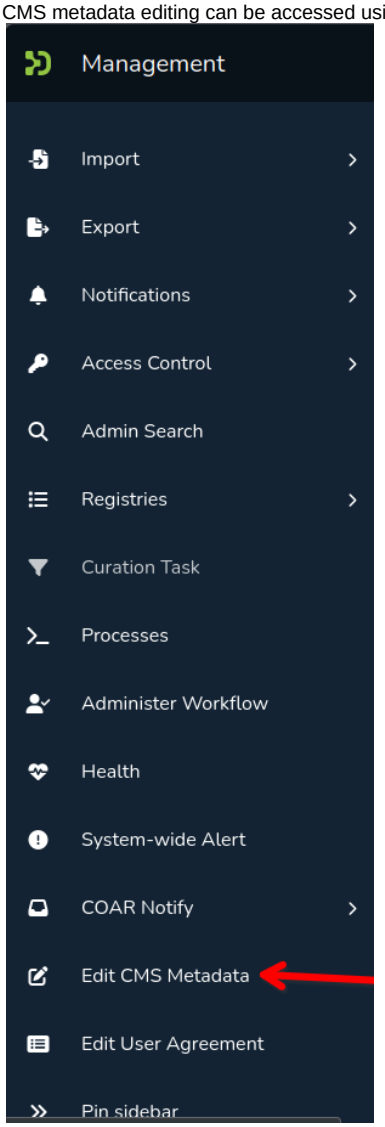


3. Top Footer:



Access to the Functionality

CMS metadata editing can be accessed using the admin sidebar:



This menu entry is configurable with an i18n label on the Angular side. The key is: **menu.section.edit-cms-metadata**.

Functionality Overview

The CMS Metadata edit functionality allows for editing content in DSpace without changing the source code.

Markdown Support

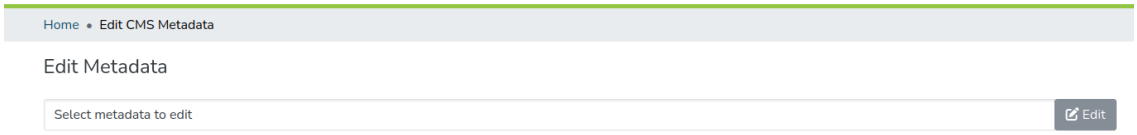
CMS Metadata Edit functionality supports Markdown.

However, in order for it to display properly, you need to enable it by enabling the following setting in your **config.yml** file:

```
markdown:
  enabled: true
```

The edit website allows you to specify the content for each language declared in your **languages** section in your **config.yml** configuration file.

Upon initial visit the page will allow you to specify the CMS metadata that you wish to edit:



Click the ***Select metadata to edit*** selector will display editable metadata.

Footer Content

Since the top footer content can only be edited via CMS metadata edit functionality if it's enabled, you need to change the following configuration in **config.yml** for it to display properly:

```
homePage:
  # Whether to show the top footer section or not
  showTopFooter: true
```

Otherwise you will not see the metadata for the footer in the dropdown.

Metadata Schema

Editable metadata can be changed in the frontend **config.yml** configuration file under the following:

```
cms:
  metadataList: [
        'metadata1',
        'metadata2',
        'etc...'
  ]
```

Then these metadata values will be saved on the main **Site** object in the database. Because of this, you can then use the metadata key in the source code by pulling the metadata of the **Site** object, similar to the following:

```
this.site$ = this.route.data.pipe(
  map((data) => data.site as Site),
);

this.homeHeaderMetadataValue$ = combineLatest({
  site: this.site$,
  language: this.locale.getCurrentLanguageCode(),
}).pipe(
  take(1),
  map(({ site, language }) => site?.firstMetadataValue('your.metadata.key', { language })),
);
```

By default, DSpace 10 defines the following metadata keys:

1. **dspace.cms.home-header** - for the Home Header panel.
2. **dspace.cms.home-news** - for the Home News panel
3. **dspace.cms.footer** - for the Footer panel

You can inspect how they are used and change the hard-coded metadata keys in the .ts and .html files of those components:

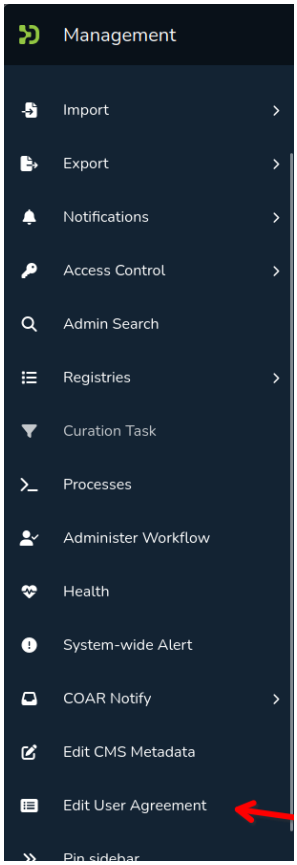
1. Home Header: ***home-page.component.ts*** + ***home-page.component.html*** (*ds-base-home-page* selector)
2. Home News: ***home-news.component.ts*** + ***home-news.component.html*** (*ds-base-home-news* selector)
3. Footer: ***footer.component.ts*** + ***footer.component.html*** (*ds-base-footer* selector)

Editing User Agreement

DSpace 10 allows to edit the User Agreement displayed when the user logs into the system for the first time.

Access to the Functionality

User Agreement edit page can be accessed using the administrator sidebar:



This menu entry is configurable with an i18n label on the Angular side. The key is: **menu.section.edit-user-agreement**.

Functionality Overview

User Agreement page allows for editing User Agreement shown when the user logs in for the first time. You can edit the User Agreement for all languages specified under the **languages** section in your **config.yml** file.

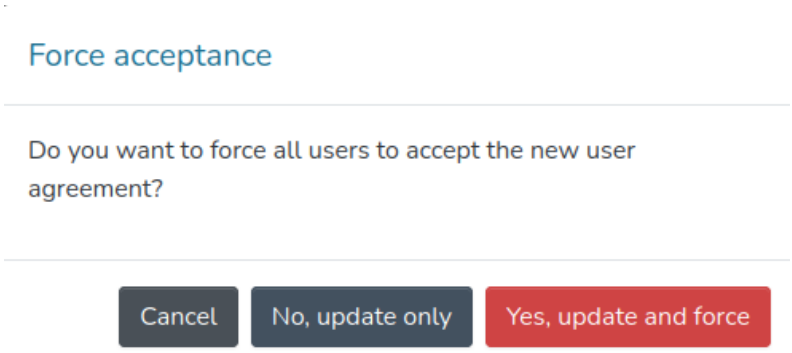
Markdown Support

User Agreement editing supports Markdown.

However, in order for it to display properly, you need to enable it by enabling the following setting in your **config.yml** file:

```
markdown:
  enabled: true
```

Once you edit the metadata for a specific language and clicking the **Save** button at the bottom of the page, you have an option to either save the new User Agreement only for new users or force all users to accept the edited User Agreement again:



If you select the red button above, all users will be forced to accept the user license agreement upon website refresh or another login:



Metadata schema

In DSpace 10, in order to determine whether a user has accepted the End User Agreement a metadata field called **dspace.agreement.end-user** is used. When this metadata exists on a user, the system assumes they have agreed to the Agreement.
The content of the End User Agreement is stored in the **dc.rights** metadata on the **Site** object fetched using the **SiteService.find()** function. When entering the End User Agreement page, DSpace 10 will attempt to find the value of this metadata in the language of the user that is trying to log in. If the metadata field entry with the correct language is not specified, the End User Agreement content for the English language will be used.
If there are is no entry for the **dc.rights** metadata in both the target language and in English, the default DSpace End User Agreement is used.

Curation System

DSpace supports running curation tasks, which are described in this section. DSpace includes several useful tasks out-of-the-box, but the system also is designed to allow new tasks to be added between releases, both general purpose tasks that come from the community, and locally written and deployed tasks.

1 Tasks
2 Activation
3 Task Invocation
    3.1 On the command line
    3.2 In the admin UI
    3.3 Protecting reporter and task file directories
    3.4 In workflow
    3.5 In arbitrary user code
4 Asynchronous (Deferred) Operation
5 Task Output and Reporting
    5.1 Status Code
    5.2 Result String
    5.3 Reporting Stream
6 Relative Path Calcuation
7 Task Properties
8 Task Parameters
9 Scripted Tasks

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.

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:

```
### Task Class implementations
plugin.named.org.dspace.curate.CurationTask = org.dspace.ctask.general.NoOpCurationTask = noop
plugin.named.org.dspace.curate.CurationTask = org.dspace.ctask.general.ProfileFormats = profileformats
plugin.named.org.dspace.curate.CurationTask = org.dspace.ctask.general.RequiredMetadata = requiredmetadata
plugin.named.org.dspace.curate.CurationTask = org.dspace.ctask.general.ClamScan = vscan
plugin.named.org.dspace.curate.CurationTask = org.dspace.ctask.general.MicrosoftTranslator = translate
plugin.named.org.dspace.curate.CurationTask = 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`

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 options:

option	meaning
-t taskname	name of task to perform.
-T filename	name of file containing a list of tasknames to be performed. This option is available only via the command line.
-e epersonID	(required) email address or netid of the E-Person performing the task. This option is available only via the command line.
-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 filename	emit reporting to the named file. '-r -' writes reporting to standard out. If not specified, report is discarded silently. This option is available only via the command line.
-p name=value	set a runtime task parameter `name` to the value `value`. May be repeated as needed. See "Task parameters" below.

As with other command-line tools, these invocations could be placed in a cron table and run on a fixed schedule, or run on demand by an administrator.

In the admin UI

In the UI, there are several ways to execute configured Curation Tasks:

1. **From the "Curate" tab/button that appears on each "Edit Community/Collection/Item" page:** this tab allows an Administrator, Community Administrator or Collection Administrator to run a Curation Task on that particular Community, Collection or Item. When running a task on a Community or Collection, that task will also execute on all its child objects, unless the Task itself states otherwise (e.g. running a task on a Collection will also run it across all Items within that Collection).
    - NOTE: Community Administrators and Collection Administrators can only run Curation Tasks on the Community or Collection which they administer, along with any child objects of that Community or Collection. For example, a Collection Administrator can run a task on that specific Collection, or on any of the Items within that Collection.
2. **From the Administrator's "Curation Tasks" page:** This option is only available to DSpace Administrators, and appears in the Administrative side-menu. This page allows an Administrator to run a Curation Task across a single object, or all objects within the entire DSpace site.
    - In order to run a task from this interface, you must enter in the handle for the DSpace object. To run a task site-wide, you can use the handle: `[your-handle-prefix]/0`

Each of the above pages exposes a drop-down list of configured tasks, with a button to 'perform' the task, or queue it for later operation (see section below). Not all activated tasks need appear in the Curate tab - you filter them by means of a configuration property. This property also permits you to assign to the task a more user-friendly name than the PluginManager *taskname*. The property resides in `[dspace]/config/modules/curate.cfg`:

```
curate.ui.tasknames = profileformats = Profile Bitstream Formats
curate.ui.tasknames = 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):

```
curate.ui.statusmessages = -3 = Unknown Task
curate.ui.statusmessages = -2 = No Status Set
curate.ui.statusmessages = -1 = Error
curate.ui.statusmessages = 0 = Success
curate.ui.statusmessages = 1 = Fail
curate.ui.statusmessages = 2 = Skip
curate.ui.statusmessages = other = Invalid Status
```

Report output from tasks run in this way is collected by configuring a Reporter plugin. You must have exactly one Reporter configured. The default is to use the FileReporter, which writes a single report of the output of all tasks in the run over all of the selected objects, to a file in the reports directory (configured as report.dir). See `[DSpace]/config/modules/submission-configuration.cfg` for the value of `plugin.single.org.dspace.curate.Reporter`. Other Reporter implementations are provided, or you may supply your own.

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
curate.ui.taskgroups = replication = Backup and Restoration Tasks
curate.ui.taskgroups = integrity = Metadata Integrity Tasks
.....
# each group membership list is a separate property, whose value is comma-separated list of logical task names
curate.ui.taskgroup.integrity = profileformats, requiredmetadata
....
```

Protecting reporter and task file directories

As at DSpace 10.0, directories for reading task files and writing reporter output are enforced. If you wish to write curation reports to locations outside the defaults listed, you may modify or add to this configuration

```
# allowed base path(s) of curation task files
# it is recommended to restrict this path as much as possible
# so that the DSpace Processes framework may only load files as "tasks"
# from a trusted location. For multiple paths, repeat this configuration
# property for each trusted path
# Default: ${dspace.dir}
#curate.taskfile.base = ${dspace.dir}

# allowed base path of reporter output.
# Default: ${dspace.dir}/log
#curate.reporter.base = ${dspace.dir}/log
```

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="editstep">
            <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 the "editstep" of workflow for any collection, and automatically reject any submissions with infected files. It would further notify (via email) both the reviewers ("editstep" role/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="editstep" 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.

As with task invocation from the administrative UI, workflow tasks need to have a Reporter configured in `submission-configuration.cfg`.

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.setReporter(System.out);
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).

The above directs report output to standard out. Any class which implements Appendable may be set as the reporter class.

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, "123456789/4", "monthly");
```

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.

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 may be sent to a file or to standard out, when running a task from the command line. Tasks run from the administrative UI or a workflow use a configured Reporter class to collect report output. Your own code may collect the report using any implementation of the Appendable interface. Unlike the result string, there is no limit to the amount of data that may be pushed to this stream.

Relative Path Calcuation

The utility that validates input and output for the Curation framework expects an absolute path. If a relative path is given to `-r` or `-T` , the Curation system will calculate an absolute path using the CLI user's current working directory as a root directory. This is then validated against the allowed base directories in `curate.taskfile.base` or `curate.reporter.base` . For the most predictable results, it is recommended to use absolute paths for these CLI paramters.

Task Properties

DSpace 1.8 introduces a new "idiom" for tasks that require configuration data. It is available to any task whose implementation extends `AbstractCuratio nTask`, 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`. It could look up its configuration properties in the ordinary way. 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 the task properties facility instead of ordinary configuration lookup. For example, each asks for the property `clamav.service.host`. At runtime, the curation system **resolves** this request to a set of configuration properties, and it uses the *name the task has been configured as* as the prefix of the properties. So, for example, if both were installed (in, say, `curate.cfg`) as:

```
org.dspace.ctask.general.ClamAv = vscan,
org.community.ctask.ConflictTask = virusscan,
....
```

then the task property `foo` will resolve to the property named `vscan.foo` when called from ClamAv task, but `virusscan.foo` 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.

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 (perhaps) `[dspace] /config/modules/thumbnail.cfg`:

```
...other properties...
thumbnail.thumbnail.maxheight = 80
thumbnail.thumbnail.maxwidth = 80
thumbnail.forceupdate=false
```

The thumbnail generating task code would then resolve "forcedupdate" to see whether filtering should be forced.

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 perhaps called: `thumbnail.force.cfg`. In this file, we put the properties:

```
thumbnail.force.thumbnail.maxheight = 80
thumbnail.force.thumbnail.maxwidth = 80
thumbnail.force.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 uses the `thumbnail.*` properties and operates in "non-force" profile (since the value is false), but when we run the task "`thumbnail.force`" the curation system uses the `thumbnail.force.*` properties. 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.

See Task Properties in Curation Tasks for details of how properties are resolved in task code.

Task Parameters

New in DSpace 7, you can pass parameters to a task at invocation time. These runtime parameters will be presented to the task as if they were task properties (see above) and, if present, will override the value of identically-named properties. Example:

Task parameters

```
bin/dspace curate -t reticulate -i 123456789/36 -p foreground=red -p background=green
```

Scripted Tasks

The procedure to set up curation tasks in Jython is described on a separate page: Curation tasks in Jython

DSpace 1.8 includes limited (and somewhat experimental) support for deploying and running tasks written in languages other than Java. Since version 6, Java has provided a standard way (API) to invoke so-called scripting or dynamic language code that runs on the java virtual machine (JVM). Scripted tasks are those written in a language accessible from this API. The exact number of supported languages will vary over time, and the degree of maturity of each language, or suitability of the language for curation tasks will also vary significantly. However, preliminary work indicates that Ruby (using the JRuby runtime) and Groovy may prove viable task languages.

Support for scripted tasks does **not** include any DSpace pre-installation of the scripting language itself - this must be done according to the instructions provided by the language maintainers, and typically only requires a few additional jars on the DSpace classpath. Once one or more languages have been installed into the DSpace deployment, task support is fairly straightforward. One new property must be defined in `[dspace]/config/modules/curate.cfg`:

```
curate.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.

Bundled Tasks

DSpace bundles a small number of tasks of general applicability. Those that do not require configuration (or have usable default values) are activated by default to demonstrate the use of the curation system. They may be deactivated by means of configuration, if desired, without affecting system integrity. Those that require configuration may be enabled (activated) by means editing DSpace configuration files. Each task is briefly described in this section.

All bundled tasks are in the package `org.dspace.ctask.general`. So, for example, to activate the no-operation task, which is implemented in the class `NoOpCurationTask`, one would configure:

```
plugin.named.org.dspace.curate.CurationTask = org.dspace.ctask.general.NoOpCurationTask = noop
```

Bitstream Format Profiler Task

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.

Link Checker Tasks

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.

MetadataWebService Task

DSpace item metadata can contain any number of identifiers or other field values that participate in networked information systems. For example, an item may include a DOI which is a controlled identifier in the DOI registry. Many web services exist to leverage these values, by using them as 'keys' to retrieve other useful data. In the DOI case for example, CrossRef provides many services that given a DOI will return author lists, citations, etc. The MetadataWebService task enables the use of such services, and allows you to obtain and (optionally) add to DSpace metadata the results of any web service call to any service provider. You simply need to describe what service you want to call, and what to do with the results. Using the task code (`[task code]`), you can create as many distinct tasks as you have services you want to call.

Each task description lives in a configuration file in 'config/modules' (or in your local.cfg), and is a simple properties file, like all other DSpace configuration files (see Configuration Reference). All of the settings associated with a given task should be prepended with the task name (as assigned in `config/modules/curate.cfg`). For example, if the task name is `issn2pubname` in curate.cfg, then all settings should start with "`issn2pubname.`" Your settings can either be set in your `local.cfg` , or in a new configuration file which is included (`include = path/to/new/file.cfg`) into either your local.cfg or the dspace.cfg. See the Configuration Reference for examples of including configuration files, or modifying your `local.cfg`

There are a few required properties you must configure for any service, and for certain services, a few additional ones. An example will illustrate best.

ISSN to Publisher Name

Suppose items (holding journal articles) include 'dc.identifier.issn' when available. We might also want to catalog the publisher name (in 'dc.publisher'). The cataloger could look up the name given the ISSN in various sources, but this 'research' is tedious, costly and error-prone. There are many good quality, free web services that can furnish this information. So we will configure a MetadataWebService task to call a service, and then automatically assign the publisher name to the item metadata. As noted above, all that is needed is a description of the service, and what to do with the results. Create a new file in 'config/modules' called 'issn2pubname.cfg' (or whatever is mnemonically useful to you). The first property in this file describes the service in a 'template'. The template is just the URL to call the web service, with parameters to substitute values in. Here we will use the 'Sherpa/Romeo' service:

```
[taskcode].template=http://www.sherpa.ac.uk/romeo/api29.php?issn={dc.identifier.issn}
```

When the task runs, it will replace '{dc.identifier.issn}' with the value of that field in the item, If the field has multiple values, the first one will be used. As a web service, the call to the above URL will return an XML document containing information (including the publisher name) about that ISSN. We need to describe what to do with this response document, i.e. what elements we want to extract, and what to do with the extracted content. This description is encoded in a property called the 'datamap'. Using the example service above we might have:

```
[taskcode].datamap=//publisher/name=>dc.publisher,//romeocolor
```

Each separate instruction is separated by a comma, so there are 2 instructions in this map. The first instruction essentially says: find the XML element 'publisher name' and assign the value or values of this element to the 'dc.publisher' field of the item. The second instruction says: find the XML element 'romeocolor', but do not add it to the DSpace item metadata - simply add it to the task result string (so that it can be seen by the person running the task). You can have as many instructions as you like in a datamap, which means that you can retrieve multiple values from a single web service call. A little more formally, each instruction consists of one to three parts. The first (mandatory) part identifies the desired data in the response document. The syntax (here '//publisher/name') is an XPath 1.0 expression, which is the standard language for navigating XML trees. If the value is to be assigned to the DSpace item metadata, then 2 other parts are needed. The first is the 'mapping symbol' (here '=>'), which is used to determine how the assignment should be made. There are 3 possible mapping symbols, shown here with their meanings:

```
'->' mapping will add to any existing value(s) in the item field
'=>' mapping will replace any existing value(s) in the item field
'~>' mapping will add *only if* item field has no existing value(s)
```

The third part (here 'dc.publisher') is simply the name of the metadata field to be updated. These two mandatory properties (template and datamap) are sufficient to describe a large number of web services. All that is required to enable this task is to edit '`config/modules/curate.cfg`' (or your `local.cfg`), and add '`issn2pubname`' to the list of tasks:

```
plugin.named.org.dspace.curate.CurationTask = org.dspace.ctask.general.MetadataWebService = issn2pubname
plugin.named.org.dspace.curate.CurationTask = org.dspace.ctask.general.MetadataWebService = doi2crossref
```

If you wish the task to be available in the Admin UI, see the Invocation from the Admin UI documentation (above) about how to configure it. The remaining sections describe some more specialized needs using the MetadataWebService task.

HTTP Headers

For some web services, protocol and other information is expressed not in the service URL, but in HTTP headers. Examples might be HTTP basic auth tokens, or requests for a particular media type response. In these cases, simply add a property to the configuration file (our example was 'issn2pubname.cfg') containing all headers you wish to transmit to the service:

```
[taskcode].headers=Accept: application/xml||Cache-Control: no-cache
```

You can specify any number of headers, just separate them with a 'double-pipe' ('||'). Ensure that any commas in values are escaped (with backslash comma, i.e. '\,').

Transformations

One potential problem with the simple parameter substitutions performed by the task is that the service might expect a different format or expression of a value than the way it is stored in the item metadata. For example, a DOI service might expect a bare prefix/suffix notation ('10.000/12345'), whereas the DSpace metadata field might have a URI representation ('http://dx.doi.org/10.000/12345'). In these cases one can declare a 'transformation' of a value in the template. For example:

```
[taskcode].template=http://www.crossref.org/openurl/?id={doi:dc.relation.isversionof}&format=unixref
```

The 'doi:' prepended to the metadata field name declares that the value of the 'dc.relation.isversionof' field should be *transformed* before the substitution into the template using a transformation named 'doi'. The transformation is itself defined in the same configuration file as follows:

```
[taskcode].transform.doi=match 10. trunc 60
```

This would be read as: exclude the value string up to the occurrence of '10.', then truncate any characters after length 60. You may define as many transformations as you want in any task, although generally 1 or 2 will suffice. They keywords 'match', 'trunc', etc are names of 'functions' to be applied (in the order entered). The currently available functions are:

```
'cut' <number> = remove number leading characters
'trunc' <number> = remove trailing characters after number length
'match' <pattern> = start match at pattern
'text' <characters> = append literal characters (enclose in ' ' when whitespace needed)
```

When the task is run, if the transformation results in an invalid state (e.g. cutting more characters than there are in the value), the un-transformed value will be used and the condition will be logged. Transformations may also be applied to values returned from the web service. That is, one can apply the transformation to a value before assigning it to a metadata field. In this case, the declaration occurs in the datamap property, not the template:

```
[taskcode].datamap=//publisher/name=>shorten:dc.publisher,//romeocolor
```

Here the task will apply the 'shorten' transformation (which must be defined in the same config file) before assigning the value to 'dc.publisher'.

Result String Programatic Use

Normally a task result string appears in a window in the admin UI after it has been invoked. The MedataWebService task will concatenate all the values declared in the 'datamap' property and place them in the result string using the format: 'name:value name:value' for as many values as declared. In the example above we would get a string like 'publisher: Nature romeocolor: green'. This format is fine for simple display purposes, but can be tricky if the values contain spaces. You can override the space separator using an optional property 'separator' (put in the config file, with all other properties). If you use:

```
[taskcode].separator=||
```

for example, it becomes easy to parse the result string and preserve spaces in the values. This use of the result string can be very powerful, since you are essentially creating a map of returned values, which can then be used to populate a user interface, or any other way you wish to exploit the data (drive a workflow, etc).

Limits and Use

A few limitations should be noted. First, since the response parsing utilizes XPath, the service can only operate on XML, (not JSON) response documents. Most web services can provide either, so this should not be a major obstacle. The MetadataWebService can be used in many ways: showing an admin a value in the result string in a UI, run in a batch to update a set of items, etc. One excellent configuration is to wire these tasks into submission workflow, so that 'automatic cataloging' of many fields can be performed on ingest.

MicrosoftTranslator Task

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
translator.field.language = dc.language

## Metadata fields you wish to have translated
translator.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
translator.language.default = en

## Target languages for translation
translator.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.
translator.api.key.microsoft = YOUR_MICROSOFT_API_KEY_GOES_HERE
translator.api.key.google = YOUR_GOOGLE_API_KEY_GOES_HERE
```

NoOp 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.

Required Metadata Task

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 Task

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 list of curation tasks.

```
### Task Class implementations
plugin.named.org.dspace.curate.CurationTask = org.dspace.ctask.general.NoOpCurationTask = noop
plugin.named.org.dspace.curate.CurationTask = org.dspace.ctask.general.ProfileFormats = profileformats
plugin.named.org.dspace.curate.CurationTask = org.dspace.ctask.general.RequiredMetadata = requiredmetadata
# This is the ClamAV scanner plugin
plugin.named.org.dspace.curate.CurationTask = org.dspace.ctask.general.ClamScan = vscan
plugin.named.org.dspace.curate.CurationTask = org.dspace.ctask.general.MicrosoftTranslator = translate
plugin.named.org.dspace.curate.CurationTask = org.dspace.ctask.general.MetadataValueLinkChecker = checklinks
```

- Optionally, add the vscan friendly name to the configuration to enable it in the administrative it in the administrative user interface.

```
curate.ui.tasknames = profileformats = Profile Bitstream Formats
curate.ui.tasknames = requiredmetadata = Check for Required Metadata
curate.ui.tasknames = checklinks = Check Links in Metadata
# Enable ClamAV from UI
curate.ui.tasknames = vscan = Virus Scan
```

- In `[dspace]/config/modules`, edit configuration file clamav.cfg:

```
clamav.service.host = 127.0.0.1
# Change if not running on the same host as your DSpace installation.
clamav.service.port = 3310
# Change if not using standard ClamAV port
clamav.socket.timeout = 120
# Change if longer timeout needed
clamav.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`:

NOT YET SUPPORTED IN 7.x or later

```
submission-curation.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`:

NOT YET SUPPORTED IN 7.x or later

```
submission-curation.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	T	Report on 1st infected bitstream
Item	F	Report on all infected bitstreams

Custom URL for Entities

Dspace 10 allows to set a custom user defined URL to refer at a given Entity (Person, Publication, etc.) replacing the standard one in use in the system.

This URL is set in a submission form panel, `custom-url` , that can be placed in different submission definitions, in order to define who and when is able to define this URL (Submitter during submission, an user allowed to edit the Entity, just the administrators when editing an Entity, etc.).

Custom URL is then used to refer to DSpace entity where it has been set. Custom URL can be changed, previously existing value(s) can be kept or deleted. When kept, previously defined URLs will continue to redirect to the Entity. In this case, if /custom-url is called, browser will redirect to whereas, if /custom-url is deleted each http(s) call toward this URL will result in a 404.

- Submission Configuration Example
  - Configuration in item-submission.xml
  - UI outcome
- Custom URL auto generation
  - Supported Entity Types
  - Metadata Mapping
    - Property format
      - Supported metadata field formats
    - Important notes
    - Examples
    - Complete Example Configuration
    - Event Consumer Configuration
- Custom URL Handling During Item Versioning

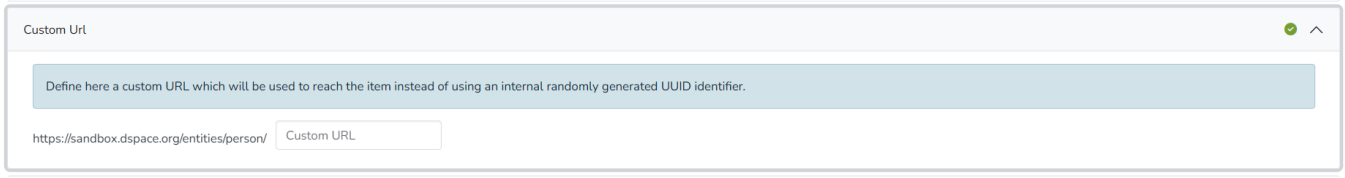
Submission Configuration Example

Configuration in *item-submission.xml*

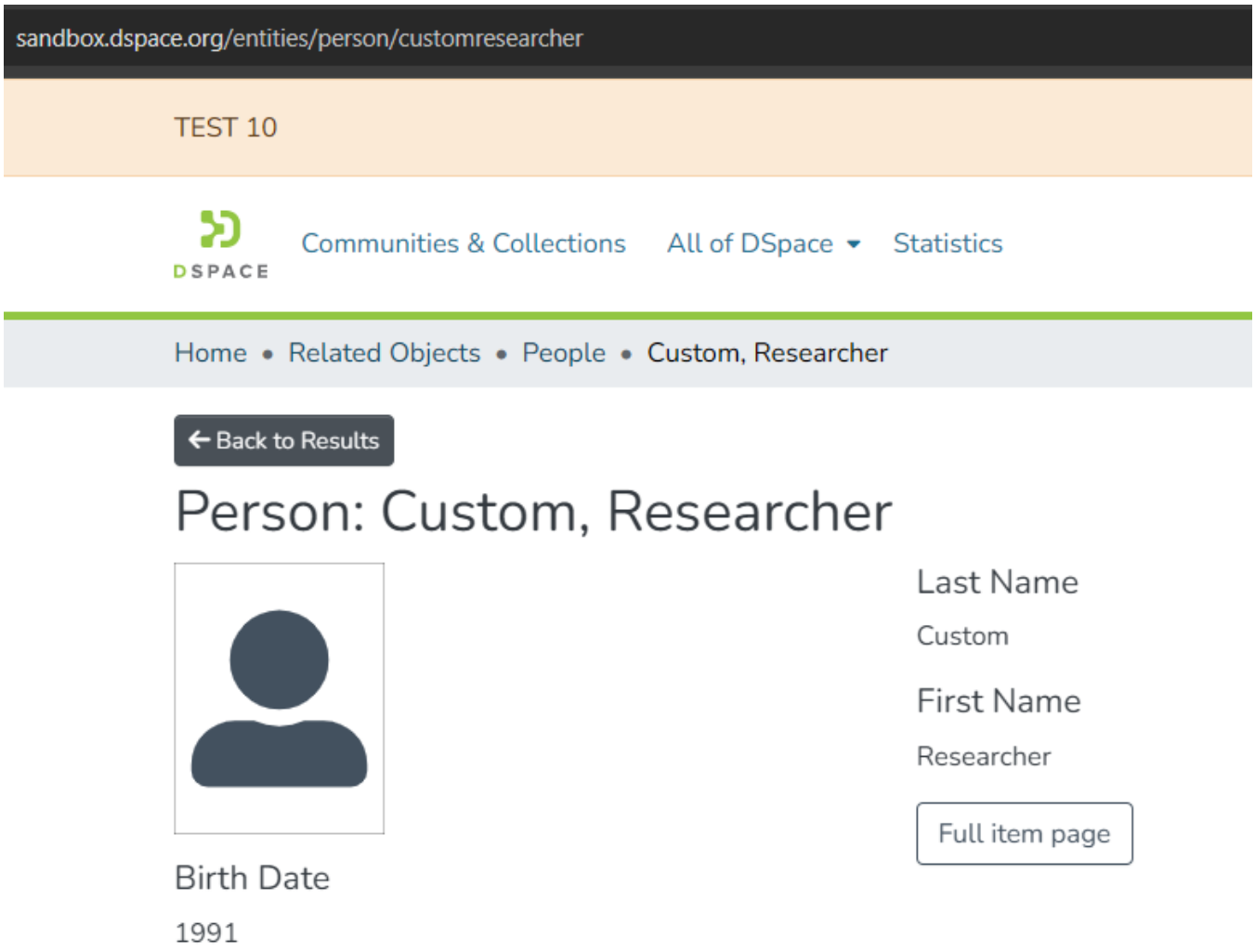
```
<submission-process name="Person">
    <step id="collection"/>
    <step id="personStep"/>
    <step id="custom-url"/>
    <step id="upload"/>
    <step id="license"/>
</submission-process>
```

UI outcome

During the submission workflow, a **dedicated submission step** allows the user to choose or edit the Custom URL.



Once published, the Person item page is accessible using a URL that includes the custom segment.



Warning - Duplicate custom URLs

Custom URLs must be **globally unique**.
Assigning the same `dspace.customurl` value to more than one entity will cause **all affected items to become inaccessible** in the User Interface and display an *"Error fetching item"* message.

The **Administer Metadata** screen does not prevent duplicate values.
Manual edits to `dspace.customurl` should be avoided unless strictly necessary and must always ensure uniqueness.

In case of duplication, the issue can only be resolved by an Administrator by accessing the item edit page directly

Custom URL auto generation

It is possible to define, for one or many entities, an auto-generation of the custom url, created starting from configurable metadata of the chosen Entity.

Supported Entity Types

The following property defines which **entity types** are eligible for automatic custom URL generation:

```
dspace.custom-url.consumer.supported-entities = EntityType1,EntityType2
```

Only items whose **entity type label exactly matches** one of the values listed in this property will be processed by the **`CustomUrlConsumer`**.

Metadata Mapping

`For each entity type listed in dspace.custom-url.consumer.supported-entities, a corresponding` **`metadata mapping`** `must be defined.`
`This mapping specifies which metadata fields are used to generate the custom URL.`

The **CustomUrlConsumer** concatenates the values of the configured metadata fields **in the order listed**.

Property format

```
dspace.custom-url.consumer.entity-metadata-mapping.{EntityType} = field1,field2,...,fieldN
```

Supported metadata field formats

- **Two-part format**: `schema.element`
  Example: `dc.title`
- **Three-part format**: `schema.element.qualifier`
  Example: `dc.contributor.author`

Important notes

- If an entity type is listed in `supported-entities` **but no metadata mapping is defined for it**, **no custom URL will be generated** for that entity
- Both the supported entity list and the metadata mapping **must be correctly configured** for custom URL generation to work

Examples

Single metadata field:

```
dspace.custom-url.consumer.entity-metadata-mapping.Publication = dc.title
```

Multiple metadata fields (concatenated):

```
dspace.custom-url.consumer.entity-metadata-mapping.Publication = dc.title,dc.contributor.author
```

Person entity using structured metadata:

```
dspace.custom-url.consumer.entity-metadata-mapping.Person = person.familyName,person.givenName
```

Complete Example Configuration

```
dspace.custom-url.consumer.supported-entities = Person, Publication, OrgUnit

dspace.custom-url.consumer.entity-metadata-mapping.Person = person.familyName,person.givenName

dspace.custom-url.consumer.entity-metadata-mapping.Publication = dc.title

dspace.custom-url.consumer.entity-metadata-mapping.OrgUnit = dc.title
```

Event Consumer Configuration

The **CustomUrlConsumer** is registered as an event consumer and reacts to item lifecycle events:

```
event.consumer.customurl.class = org.dspace.app.customurl.consumer.CustomUrlConsumer

event.consumer.customurl.filters = Item+Install|Modify|Modify_Metadata
```

This means custom URLs are generated or updated when:

- An item is installed
- An item is modified
- Item metadata is modified

Warning - Duplicate custom URLs

If the Entity for which the custom url is automatically created already has a custom url set, it will not be created.

Custom URL Handling During Item Versioning

When an item is versioned in DSpace, the custom URL behaves as follows:

1. The new version inherits the custom URL
    - When a new version of an item is created, it automatically receives the same custom URL as the original item
2. The original item's URL is preserved in legacy field
    - The original item's custom URL is moved to the dspace.customurl.old metadata field
    - This preserves the history but removes the URL from the current version
    - The original item can no longer be found by its custom URL
3. URL lookups always return the latest version
    - Searching by custom URL will always resolve to the most recent version of the item
    - Users are redirected to the latest version automatically

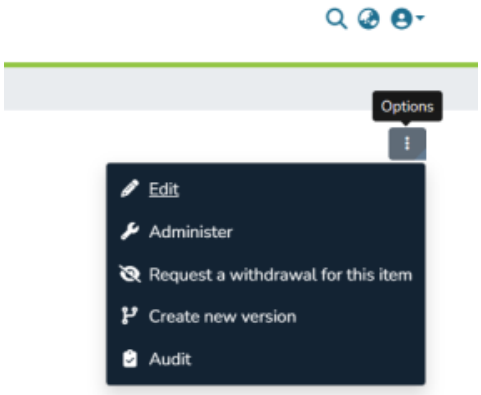
Edit Item in Submission mode

- Access to the Functionality
- Edit Modes Configuration
  - Edit Item Service Configuration
    - Security Levels
  - Configure a custom security level
    - Custom level for user / group
    - Custom level using metadata
  - Additional Filter

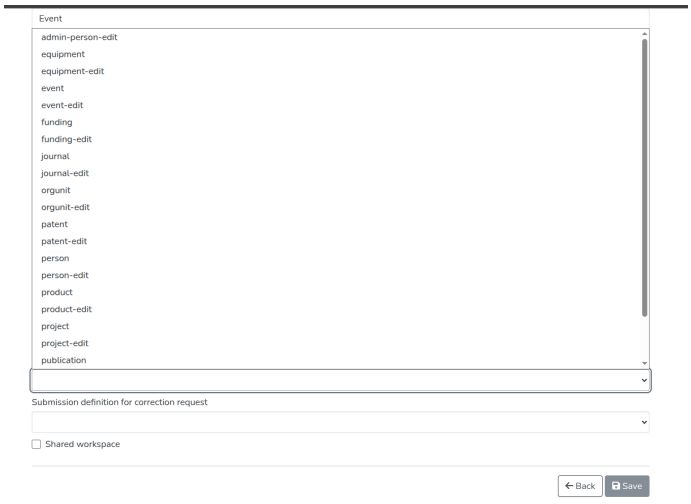
The new **Edit** menu option allows administrators to open a **submission form populated with the existing metadata** metadata for further enrichment of the item even if an item is archived.

Access to the Functionality

This modality is accessible from the item page.



Along this feature, the possibility to pick which form to use for each collection submission has been introduced:



Edit Modes Configuration

To configure one or more edit modalities for a specific entity type is necessary to add the configuration in the property ***editModesMap*** of bean ***org.dspace.content.edit.service.impl.EditItemModeServiceImpl*** inside ***edititem-service.xml*** file.

The first step is the creation of a new submission-process definition for the entity type in the *item-submission.xml* file.
For example:

```
<submission-process name="admin-person-edit">
     <step id="personStep"/>
     <step id="owner" />
     <step id="upload" />
</submission-process>
```

Once the submission definitions are created you have to add a new entry in the property ***editModesMap*** of the bean ***org.dspace.content.edit.service.impl.EditItemModeServiceImpl***.

In an instance without edit modes configuration the property *editModesMap* is configured as follow:

```
<bean class="org.dspace.content.edit.service.impl.EditItemModeServiceImpl">

</bean>
```

The *editModesMap* property is a Map that has as key the entity type name (in lowercase) and (optional) a submission definition, and as value a list of edit modes.

Edit Item Service Configuration

`dspace/config/spring/api/edititem-service.xml`

Define edit modes for each entity type:

```
<bean class="org.dspace.content.edit.service.impl.EditItemModeServiceImpl">
       <property name="editModesMap">
           <map>
               <entry key="person">
                   <list>
                       <bean class="org.dspace.content.edit.EditItemMode">
                           <property name="name" value="FULL" />
                           <property name="security">
                               <value type="org.dspace.content.security.CrisSecurity">
                                   ITEM_ADMIN
                               </value>
                           </property>
                           <property name="submissionDefinition" value="admin-person-edit" />
                       </bean>
                       <bean class="org.dspace.content.edit.EditItemMode">
                           <property name="name" value="OWNER" />
                           <property name="security">
                               <value type="org.dspace.content.security.CrisSecurity">
                                   OWNER
                               </value>
                           </property>
                           <property name="submissionDefinition" value="person-edit" />
                       </bean>
                   </list>
               </entry>
           </map>
       </property>
   </bean>
```

As you can see, an edit mode is represented by class ***org.dspace.content.edit.EditItemMode***, this class has follow properties:

- **name**: this is a unique identifier for the edit mode at entity type level;
- **label**: this is an optional property, if set, the UI will use its value as i18n key otherwise the UI will use the value of name property;

- **submissionDefinition**: contains the name of the submission definition to use with the current edit mode (e.g. person-edit or admin-person-edit); Edit lookup logic checks for <entity>.<submission_definition> where <submission_definition> is the submission defined for the collection who the Item to be edited belongs. In case no matches are found, a fallback on the entry with key <entity> is done.
- **securities**: define the security levels available for the current edit mode and its visibility. It must be set with a list of “security” values.
- **security (alternative to “securities”)**: defines the security level for the current edit mode and its visibility. This property is an Enum (*org.dspace.content.edit.EditItemMode*) and accepts one of the following values:
    - **ADMIN**: this value allows only the Administrator to use this edit mode
    - **OWNER**: this value allows only the item owner to use this edit mode
    - **SUBMITTER**: this value allows the Submitter of the Item to use this edit mode
    - **SUBMITTER_GROUP**: this value allows all users in the the group of submitters of the Item owner to use this edit mode
    - **CUSTOM**: this value is used to define a fine-grained security access. This security level is explained in following paragraph
    - **GROUPS**: this value is used to define a fine-grained security access to users belonging to a group. This security level is a shortcut for “CUSTOM” value with access policies to be evaluated only according to what defined in “groups” node.
    - **ALL:** always allowed access, it should be used in pair with custom additional Filter (see in “Additional Filter” paragraph).

Security Levels

Either a single property named `security` or a list of `securities` are admitted. This second case applies when the same edit item mode must be made available according to different policies on the item to be edited:

```
<bean class="org.dspace.content.edit.EditItemMode">
    <property name="name" value="MODE-MANY-SECURITIES" />
    <property name="label" value="edititem.mode.test2" />
    <property name="securities">
      <list>
        <value>
          ADMIN
        </value>
        <value>
          OWNER
        </value>
      </list>
      </property>
    <property name="submissionDefinition" value="edit_admin_publication" />
</bean>
```

Configure a custom security level

Custom level for user / group

This security level allows to use the edit mode only to the users/groups that are present inside a list of metadata associated to the item. When configuring the CUSTOM security levels you need to set two other properties within the configuration: users and groups. These properties contains a list of metadata used to check if a specific user/group are allowed to use this edit mode

```
<bean class="org.dspace.content.edit.EditItemMode">
    <property name="name" value="MODE1" />
    <property name="label" value="edititem.mode.test2" />
    <property name="security">
      <value type="org.dspace.content.edit.EditItemModeSecurity">
        CUSTOM
      </value>
    </property>
    <property name="submissionDefinition" value="admin-person-edit" />

    <property name="groups">
        <list>
            <value>group1</value>
            <value>group2</value>
        </list>
    </property>
    <property name="users">
        <list>
            <value>user</value>
        </list>
    </property>
</bean>
```

Custom level using metadata

In some cases, you don't have a direct reference to the owner inside a metadata, and you want to be fine-grained as you could be with the `users` configura tion.

In case you have a referenceable metadata (i.e. `dspace.custom-profile`) with a mapped `dspace.object.owner` on itself, you could use the `items` property inside the `EditItemMode` configuration.

A valuable example could be the usage of a `researcher-profile` reference inside a metadata. Usually, the `researcher-profile` has a reference to its owner (i.e. the user that has claimed the profile) inside the metadata `dspace.object.owner`. Whenever you want to make editable some items using a *metadata-reference* to a `researcher-profile` you can configure the `items` property with this metadata.

```
<bean class="org.dspace.content.edit.EditItemMode">
    <property name="name" value="ITEMS-MODE" />
    <property name="label" value="edititem.mode.researcher-profile" />
    <property name="security">
      <value type="org.dspace.content.edit.EditItemModeSecurity">
        CUSTOM
      </value>
    </property>
    <property name="submissionDefinition" value="edit_item_researcher" />

    <property name="items">
        <list>
            <value>my.custom.researcher</value>
        </list>
    </property>
</bean>
```

In the previous example we have configured the `EditMode` for the *logged-user* that owns the `researcher-profile` referenced by the `my.custom. researcher` metadata. So, whenever the logged user matches the `dspace.object.owner` referenced by the item behind `my.custom.researcher`, that logged user will be able to edit the item.

Additional Filter

`AccessMode` Implementing beans (e.g. `org.dspace.content.edit.EditItemMode`) can be used with an `additionalFilter` parameter. This parameter represents a bean implementing `org.dspace.content.logic.Filter` interface. This custom bean is applied in addition to evaluation performed by standard security policies.

For example, in this case:

```
<bean class="org.dspace.content.edit.EditItemMode">
  <property name="name" value="FIRST" />
  <property name="security">
    <value type="org.dspace.content.security.CrisSecurity">
      ADMIN
    </value>
  </property>
  <property name="submissionDefinition" value="modeA" />
  <property name="additionalFilter" ref="adminNameAdditionalFilter"/>
</bean>
...
<bean class="org.dspace.content.logic.AdminNameFilter" id="adminNameAdditionalFilter"/>
```

```
public class AdminNameFilter implements Filter {
    @Override
    public Boolean getResult(Context context, Item item) throws LogicalStatementException {

        return "John".equals(context.getCurrentUser().getFirstName());
    }
...
}
```

“FIRST” edit mode would be granted to all users having ADMIN grant (CrisSecurity), and having “John” as first name (`AdminNameFilter` bean).

Exporting Content and Metadata

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

- Signposting
- OpenAIRE4 Guidelines Compliancy
- OAI
- Exchanging Content Between Repositories
- SWORDv1 Client
- Linked (Open) Data
- Rioxx v3 schema compliance

Signposting

Overview

The concept of Signposting is aimed at facilitating machine agents in navigating scholarly information systems easily. Signposting uses typed links to clarify patterns found in scholar portals, offering a standard approach to address the issue of making the descriptive metadata and links in landing pages, usually optimized for human use, readable for machine agents.

To provide machine-friendly authorship information, the publisher can include author links in the Link header of the HTTP response. Additionally, the publisher can use a "cite-as" link to fetch the persistent identifier of the resource. These links enable bots to follow them and discover relevant additional information related to the resource.

By adopting Signposting techniques, the users contribute to improving the machine accessibility and navigation of scholarly web resources, enhancing the overall efficiency and interoperability of scholarly information systems.

More information can be found on the Signposting website: https://signposting.org/

DSpace supports FAIR Signposting Profile at Level 2: By supporting the FAIR Signposting Profile at Level 2, your platform demonstrates a commitment to improving the machine accessibility, interoperability, and reusability of scholarly resources. It ensures that the information you provide is standardized, consistent, and easily navigable by both human users and machine agents, contributing to a more efficient and FAIR scholarly web ecosystem. More information on: https://github.com/DSpace/RestContract/blob/main/signposting.md

The FAIR Signposting profile (more information on: https://signposting.org/FAIR/) is based on the FAIR principles (Findable, Accessible, Interoperable, and Reusable - https://www.go-fair.org/fair-principles/).

- Findability: Your system ensures that scholarly resources are easily discoverable by both humans and machines. It includes the use of persistent identifiers, such as DOIs (Digital Object Identifiers), to uniquely identify and locate resources. These identifiers are included in the signposting links provided in the HTTP responses.
- Accessibility: Your system supports accessibility by providing machine-readable metadata and links that facilitate automated processing. The Signposting Patterns specified in the profile guide the inclusion of links in the HTTP Link headers, HTML link elements, or Link Sets. These links convey essential information about the resource, such as authorship, identifiers, and relationships to other resources.
- Interoperability: Your system promotes interoperability by adopting standardized formats and protocols. It ensures that the signposting links and metadata adhere to established conventions and vocabularies, making it easier for machines to interpret and process the information consistently. By implementing the FAIR Signposting Profile, your system aligns with a community-accepted standard for interoperability.
- Reusability: Your system supports reusability by providing clear and structured metadata about scholarly resources. This includes information about licenses, permissions, and terms of use. By including this information in the signposting links or associated metadata, your system enables users and machines to understand the conditions under which the resources can be reused.

Enabling / Disabling

Signposting is enabled by default in DSpace 7 (starting with version 7.6). When enabled on the backend, the `${dspace.server.url}/signposting/` REST Endpoint will be available and can be used based on the documentation at https://github.com/DSpace/RestContract/blob/main/signposting.md. When disabled, this endpoint will return a 404.

However, if you wish to disable it, you can change this configuration in your local.cfg

```
signposting.enabled = false
```

Modifications to this setting require rebooting your servlet container (e.g. Tomcat)

Configuration

Additional signposting configuration options are available in `[dspace]/config/modules/signposting.cfg`. For most sites, the default settings should be all you need.

OpenAIRE4 Guidelines Compliancy

Loading of Entities and Fields

OpenAIRE4 features depends on Configurable Entities feature and its default configurations. In order to have your repository compliant with OpenAIRE4 guidelines you need to follow some steps:

The default submission-forms.xml file configures the form fields that allow the creation of the specific OpenAIRE entities and their relationships. In order to use those forms you need to configure your item-submission.xml and add these to the `<submission-map>`:

item-submission.xml

```
<name-map collection-handle="123456789/2" submission-name="openAIREPublicationSubmission" />
<name-map collection-handle="123456789/3" submission-name="openAIREPersonSubmission" />
<name-map collection-handle="123456789/5" submission-name="openAIREProjectSubmission" />
<name-map collection-handle="123456789/4" submission-name="openAIREOrganizationSubmission" />
```

Please note that my `collection-handle="123456789/4"` will be different in your system and it refers to the collection that will gather a specific Entity type like Publications, Persons, Projects or Organizations.

To load OpenAIRE Entities model you must firstly run the following:

loading openaire entity-relationships model

```
[/dspace]/bin/dspace initialize-entities -f [/dspace]/config/entities/openaire4-relationships.xml
```

and load the required metadata fields

loading openaire registries

```
[/dspace]/bin/dspace registry-loader -metadata [/dspace]/config/registries/openaire4-types.xml
```

After those steps your repository will have the required fields and entities for the compliancy.

OAI interface

As decided in our Entities meeting (2019-11-19 DSpace 7 Entities WG Meeting), the XOAI Default Context should only display Publications or non Entity Items. For OpenAIRE4 it will also be considered only Publications as the main Entity to be processed and all the related ones will be loaded in the process.

OpenAIRE4 is accessible in a specific OAI context through the URL:
`http://[dspace-server-url]/oai/openaire4?verb=ListRecords&metadataPrefix=oai_openaire`

in order to use it, you must first ensure you have the oai.cfg setting uncommented:
`oai.enabled = true`
(NOTE: you may need to restart your tomcat service)

If you need to display additional metadata at the oai_openaire metadata format, you could rename the file:

`[/dspace/]config/spring/api/virtual-metadata.xml.openaire`

and replace it with the existing one:

```
[/dspace/]config/spring/api/virtual-metadata.xml
```

Please note if you do this you should restart your tomcat service container.

This additional virtual metadata will enable to represent something like this in this XML in the oai_openaire metadata format, where you have, for instance, author identifiers:

oai datacite:creators example

```
<datacite:creators>
 <datacite:creator>
   <datacite:creatorName>Evans, R.J.</datacite:creatorName>
   <datacite:affiliation>Institute of Science and Technology</datacite:affiliation>
   <datacite:nameIdentifier nameIdentifierScheme="ORCID" schemeURI="http://orcid.org"> 1234-1234-1234-1234 <
/datacite:nameIdentifier>
 </datacite:creator>
</datacite:creators>
```

Then you may need to run the OAI import from the command line with the cleaning cache parameter to reload all data to OAI:

```
[/dspace/]/bin/dspace oai import -c
```

OAI

OAI Interfaces

1 OAI-PMH Server
    1.1 OAI-PMH Server Activation
    1.2 OAI-PMH Server Maintenance
2 OAI-PMH / OAI-ORE Harvester (Client)
    2.1 Harvesting from another DSpace
    2.2 OAI-PMH / OAI-ORE Harvester Configuration
    2.3 Setting up a harvest to import content into a collection
        2.3.1 Using the "harvest" script
            2.3.1.1 Examples of harvesting a collection through CLI commands
        2.3.2 Setting up a harvest content source from the UI
3 DSpace Demo - OAI-PMH

OAI-PMH Server

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

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

OAI-PMH Server Activation

DSpace's OAI-PMH server is enabled by default. However, you can choose to enable/disable it in your local.cfg using these configurations:

```
# Enable (true) or disable (false) OAI-PMH server
oai.enabled = true

# When enabled, OAI-PMH server is available at this path
oai.path = oai
```

If you modify either of these configuration, you must restart your Servlet Container (usually Tomcat).

- You can test that it is working by sending a request to: `[dspace.server.url]/[oai.path]/request?verb=Identify` (e.g. http://localhost:8080/server/oai/request?verb=Identify)
- The response should look similar to the response from the DSpace Demo Server: https://demo.dspace.org/server/oai/request?verb=Identify

If you're using a recent browser, you should see a HTML page describing your repository. What you're getting from the server is in fact an XML file with a link to an XSLT stylesheet that renders this HTML in your browser (client-side). Any browser that cannot interpret XSLT will display pure XML. The default stylesheet is located in `[dspace-source]/dspace-oai/src/main/resources/static/style.xsl` and can be changed by configuring the `stylesheet` attribute of the `Configuration` element in `[dspace]/config/crosswalks/oai/xoai.xml`.

Relevant Links

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

OAI-PMH Server Maintenance

After activating the OAI-PMH server, you need to also ensure its index is updated on a regular basis. Currently, this doesn't happen automatically within DSpace. Instead, you must schedule the `[dspace.dir]/bin/dspace oai import` commandline tool to run on a regular basis (usually at least nightly, but you could schedule it more frequently).

Here's an example cron that can be used to schedule an OAI-PMH reindex on a nightly basis (for a full list of recommended DSpace cron tasks see Scheduled Tasks via Cron):

```
# Update the OAI-PMH index with the newest content at midnight every day
# NOTE: ONLY NECESSARY IF YOU ARE RUNNING OAI-PMH
# (This ensures new content is available via OAI-PMH)
0 0 * * * [dspace.dir]/bin/dspace oai import > /dev/null
```

More information about the `dspace oai` commandline tool can be found in the OAI Manager documentation.

OAI-PMH / OAI-ORE Harvester (Client)

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

Supported in 7.1 or above

OAI Harvesting was not available in DSpace 7.0. It was restored in DSpace 7.1. See DSpace Release 7.0 Status

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.

If the external DSpace is running v6.x or below, it must be running both the OAI-PMH interface and the XMLUI interface to support harvesting content from it via OAI-ORE.

If the external DSpace is running v7.x or above, it just needs to be running the OAI-PMH interface.

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/server/oai/request?verb=ListRecords&metadataPrefix=ore
2. For 6.x or below, 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

For examples of how to set up harvesting via the User Interface, see the "Content Source" settings of the "Edit Collection" documentation

OAI-PMH / OAI-ORE Harvester Configuration

There are many possible configuration options for the OAI harvester. Most of these are contained in the `[dspace]/config/modules/oai.cfg` file (unless otherwise noted below). They may be updated there or overridden in your `local.cfg` config file (see Configuration Reference).

Configuration File:	**`[dspace]/config/modules/oai.cfg`**
Property:	`oai.harvester.eperson`
Example Value:	`oai.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:	`oai.url`
Example Value:	`oai.url = ${dspace.server.url}/${oai.path}`
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:	`oai.ore.authoritative.source`
Example Value:	`oai.ore.authoritative.source = oai`
Informational Note:	The webapp responsible for minting the URIs for ORE Resource Maps. If using oai, the `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 `oai.url` above) • The URIs generated for ORE ReMs follow the following convention for either setting: http://\[base-URL\]/metadata/handle/\[item-handle\]/ore.xml
Property:	`oai.harvester.autoStart`
Example Value:	`oai.harvester.autoStart = false`
Informational Note:	Determines whether the harvest scheduler process starts up automatically when DSpace webapp is redeployed.

Property:	`oai.harvester.metadataformats.PluginName`
Example Value:	`oai.harvester.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 `oai.harvester.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:	`oai.harvester.oreSerializationFormat.OREPrefix`
Example Value:	`oai.harvester.oreSerializationFormat.OREPrefix = \` `http://www.w3.org/2005/Atom`
Informational Note:	This field works in much the same way as `oai.harvester.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:	`oai.harvester.timePadding`
Example Value:	`oai.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:	`oai.harvester.harvestFrequency`
Example Value:	`oai.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:	`oai.harvester.minHeartbeat`
Example Value:	`oai.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:	`oai.harvester.maxHeartbeat`
Example Value:	`oai.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:	`oai.harvester.maxThreads`
Example Value:	`oai.harvester.maxThreads = 3`
Informational Note:	How many harvest process threads the scheduler can spool up at once. Default value is 3.
Property:	`oai.harvester.threadTimeout`
Example Value:	`oai.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:	`oai.harvester.unknownField`

Example Value:	`oai.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:	`oai.harvester.unknownSchema`
Example Value:	`oai.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:	`oai.harvester.acceptedHandleServer`
Example Value:	`oai.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:	`oai.harvester.rejectedHandlePrefix`
Example Value:	`oai.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* .

Setting up a harvest to import content into a collection

There are two options to set up a collection for harvesting. One is by using the DSpace scripts "harvest", the other is by setting up the content source of a collection through the UI.

Using the "harvest" script

The harvest script can be called from both the CLI and REST API by calling "harvest". It uses the paramaters as defined in the following table.

Short option	Long option	Argument	Explanation
-p	--purge	*[none]*	Delete all the items in the collection provided with the `-c` parameter.
-r	--run	*[none]*	Run the standard harvesting procedure for the collection provided with the `-c` parameter.
-g	--ping	*[none]*	Verify that the server provided through the `-a` parameter and the set provided through the `-i` parameter can be resolved and work.
-s	--setup	*[none]*	Set the collection provided with the `-c` parameter up for harvesting. The server will need to be provided through the `-a` parameter, and the oai set id needs to be provided by the `-i` parameter.
-S	--start	*[none]*	Start the harvest loop for all collections.
-R	--reset	*[none]*	Reset the harvest status on all collections.
-P	--purgeCollections	*[none]*	Purge all harvestable collections.
-o	--reimport	*[none]*	Reimport all items the items in the collection provided by the `-c` parameter. This is the equivalent of running both the `-p` and the `-r` command for the provided collection.
-c	--collection	[id-or-handle]	The harvesting collection (handle or id)
-t	--type	[type-code]	The type of harvesting: 0 for no harvesting, 1 for metadata only, 2 for metadata and bitstream references (requires ORE support), 3 for metadata and bitstreams (requires ORE support)
-a	--address	[url]	The address of the OAI-PMH server to be harvested
-i	--oai_set_id	[set-id]	The id of the PMH set representing the harvested collection. In case all sets need to harvested the value "all" should be provided.

-m	--metadata_format	[format]	The name of the desired metadata format for harvesting, resolved to namespace and crosswalk in the dspace.cfg
-h	--help	*[none]*	Print the help message
-e	--eperson	[email]	**(CLI ONLY)** The eperson that performs the harvest. When the command is used from the REST API, the currently logged in user will be used.

Examples of harvesting a collection through CLI commands

1. Verify whether the harvester source can be reached

```
dspace/bin/dspace harvest -g -a https://harvest.source.org -i harvest-set
```

Replace `https://harvest.source.org` with the source you want to use, the `harvest-set` with the set/sets you want to harvest or `all` in case you want to harvest all sets.

2. Set up a collection for harvesting

```
dspace/bin/dspace harvest -s -c 123456789/123 -e harvest-user@dspace.org -a https://harvest.source.org -i harvest-set -m dc -t 1
```

Replace the `123456789/123` with your collection, the `harvest-user@dspace.org` with an existing user in DSpace that has sufficient rights to perform the ingestion, `https://harvest.source.org` with the source you want to use, the `harvest-set` with the set/sets you want to harves or `all` in case you want to harvest all sets. The `-m` parameter indicated the metadata format to be used and the `-t` parameter indicates the harvest type to be used. When the value `0` is used for `-t` , harvesting will be disabled.

3. Run the harvest for the set up collection

```
dspace/bin/dspace harvest -r -c 123456789/123 -e harvest-user@dspace.org
```

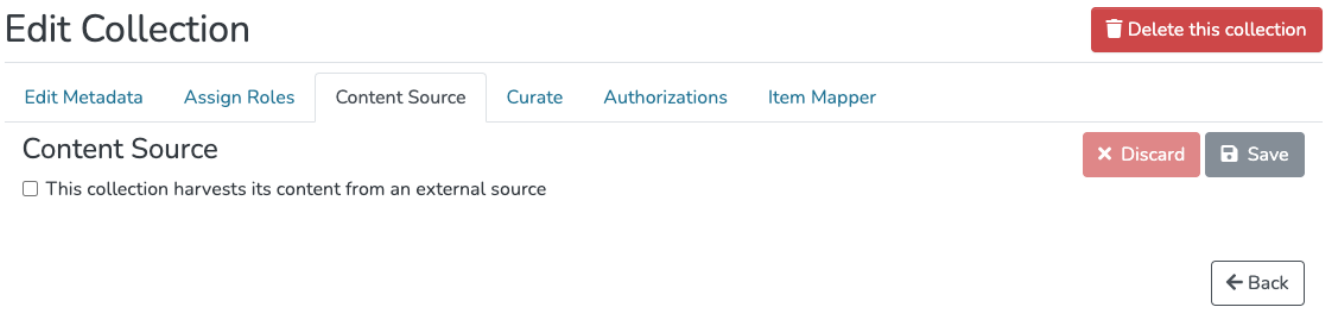
Replace the `123456789/123` with your collection, the `harvest-user@dspace.org` with an existing user in DSpace that has sufficient rights to perform the ingestion.

Setting up a harvest content source from the UI

A collection can be configured to retrieve its content from an external source. This can be done from the "Edit Collection" UI by using the following steps.

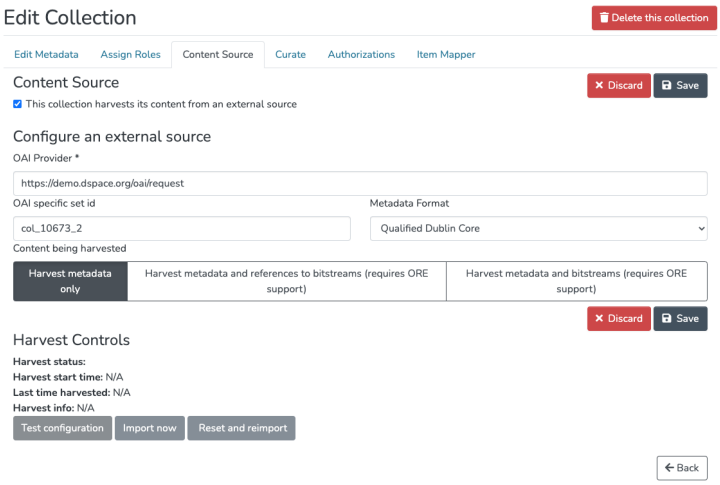
1. Configure the collection to harvest its content from an external source

Navigate to the "Edit collection" > "Content Source" tab. Tick the checkbox "This collection harvests its content from an external source".



2. Configure the harvest source

Once the checkbox has been ticket, the OAI provider, set id and metadata format can be configured. An example of the configuration can be found in the image below.

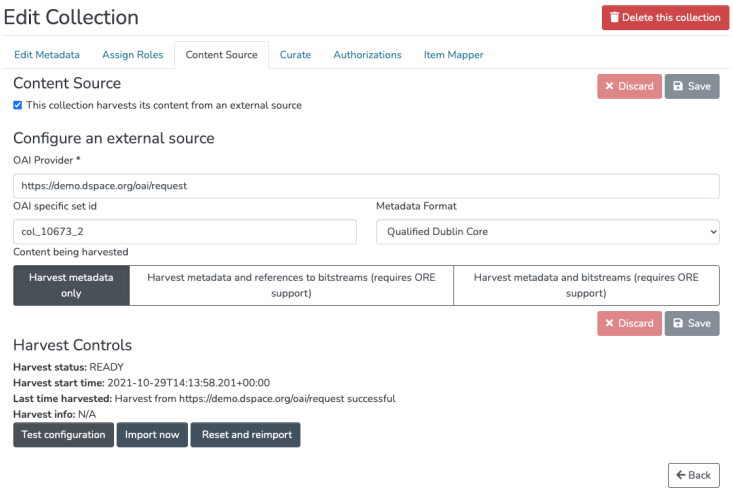


When all sets need to be harvested, the field can be left empty.

The server configuration will be tested upon clicking the "Save" button.

3. Start the harvest

Click the "Import Now" button to start the import. When the import has started, the button will indicate that the import is in progress, however, there is no need to remain on this page as the harvest will continue to run after leaving this page.



If the current server configuration needs to be retested at a later point, the "Test configuration" button can be used. To fully reset the collection by purging all items and starting a reimport, click the "Reset and reimport" button.

DSpace Demo - OAI-PMH

- https://demo.dspace.org/server/oai/request?verb=Identify

OAI 2.0 Server

1 Introduction
- 1.1 What is OAI 2.0?
- 1.2 Why OAI 2.0?
- 1.3 Concepts (XOAI Core Library)

2 OAI 2.0
- 2.1 Indexing OAI content
  - 2.1.1 OAI Manager
  - 2.1.2 Scheduled Tasks
- 2.2 Client-side stylesheet
- 2.3 Metadata Formats
- 2.4 Encoding problems

3 Configuration
- 3.1 Basic Configuration
- 3.2 Advanced Configuration
  - 3.2.1 General options
  - 3.2.2 Add/Remove Metadata Formats
  - 3.2.3 Add/Remove Metadata Fields

4 Driver/OpenAIRE compliance
- 4.1 Driver Compliance
- 4.2 OpenAIRE compliance

5 Sanity check your OAI interface with the OAI Validator

Introduction

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

What is OAI 2.0?

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

Why OAI 2.0?

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

Concepts (XOAI Core Library)

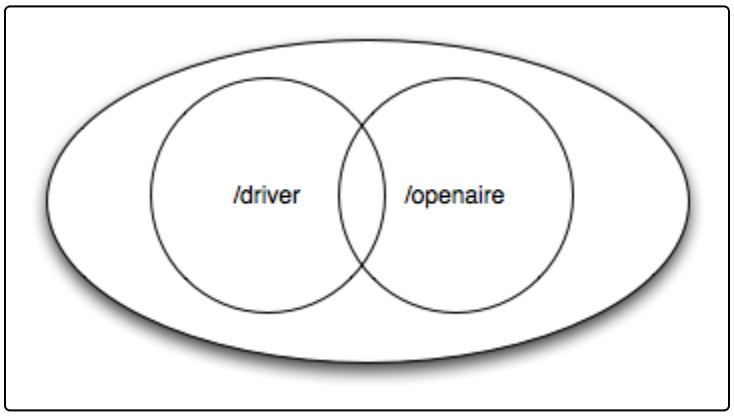
To understand how XOAI works, one must understand the concept of Filter, Transformer and Context. With a Filter it is possible to select information from the data source. A Transformer allows one to make some changes in the metadata before showing it in the OAI interface. XOAI also adds a new concept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

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

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

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

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



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

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

OAI 2.0

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

Details about OAI 2.0 internals can be found here.

The OAI 2.0 Server only uses Solr for its indexing. The previous capability to use Database indexing has been removed.

Indexing OAI content

OAI 2.0 uses Solr for all indexing of content.

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

OAI Manager

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

Syntax

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

Actions

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

Parameters

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

Scheduled Tasks

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

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

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

Client-side stylesheet

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

Metadata Formats

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

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

At /driver context it provides:

1. OAI_DC
2. DIDL
3. METS

At /openaire context it provides:

1. OAI_DC
2. METS

And at /openaire4 context it provides:

1. OAI_OPENAIRE

Encoding problems

There are two main potential sources of encoding problems:

a) The servlet connector port has to use the correct encoding. E.g. for Tomcat, this would be `<Connector port="8080" ... URIEncoding="UTF-8" />`, where the port attribute specifies port of the connector that DSpace is configured to access Solr on (this is usually 8080, 80 or in case of AJP 8009).

b) System locale of the dspace command line script that is used to do the oai import. Make sure the user account launching the script (usually from cron) has the correct locale set (e.g. en_US.UTF-8). Also make sure the locale is actually present on your system.

Configuration

Basic Configuration

Configuration File:	**`[dspace]/config/modules/oai.cfg`**
Property:	oai.enabled
Example Value:	`oai.enabled = true (default)`
Information Note:	Allows you to enable or disable the OAI module/endpoint
Property:	oai.path
Example Value:	`oai.path = oai (default)`
Information Note:	Allows you to specify the path where the OAI module will be deployed. This path is relative to the dspace.server.url. So, for example, if "dspace.server.url=http://localhost:8080/server", then by default the OAI module is available at http://localhost:8080/server/oai/
Property:	`oai.storage`

Example Value:	`oai.storage = solr`
Information Note:	This allows to choose the OAI data source between solr and database. ONLY "solr" is supported at this time.
Property:	`oai.solr.url`
Example Value:	`oai.solr.url = ${solr.server}/oai`
Informational Note:	Solr Server location
Property:	`oai.identifier.prefix`
Example Value:	`oai.identifier.prefix = ${dspace.hostname}`
Informational Note:	OAI persistent identifier prefix. Format - oai:PREFIX:HANDLE
Property:	`oai.config.dir`
Example Value:	`oai.config.dir = ${dspace.dir}/config/crosswalks/oai`
Informational Note:	Configuration directory, used by XOAI (core library). Contains xoai.xml, metadata format XSLTs and transformer XSLTs.
Property:	`oai.cache.enabled`
Example Value:	`oai.cache.enabled = true`
Informational Note:	Whether to enable the OAI cache. Default is true (for better performance).
Property:	`oai.cache.dir`
Example Value:	`oai.cache.dir = ${dspace.dir}/var/oai`
Informational Note:	Directory to store runtime generated files (for caching purposes).

Advanced Configuration

OAI 2.0 allows you to configure following advanced options:

- Contexts
- Transformers
- Metadata Formats
- Filters
- Sets

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

General options

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

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

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

```
<Configuration xmlns="http://www.lyncode.com/XOAIConfiguration"
        identation="false"
        maxListIdentifiersSize="100"
        maxListRecordsSize="100"
        maxListSetsSize="100"
        stylesheet="static/style.xsl">
```

Add/Remove Metadata Formats

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

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

Then one would have:

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

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

```
<Format id="[IDENTIFIER]">
  <Prefix>[PREFIX]</Prefix>
  <XSLT>metadataFormats/[XSLT]</XSLT>
  <Namespace>[NAMESPACE]</Namespace>
  <SchemaLocation>[SCHEMA_LOCATION]</SchemaLocation>
</Format>
```

where:

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

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

Add/Remove Metadata Fields

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

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

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

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

If you need to add/remove metadata fields, you're changing the output format. Therefore it is recommended to create a new metadata format as a copy of the one you want to modify. This way the old format will remain available along with the new one and any upgrades to the original format during DSpace upgrades will not overwrite your customizations. If you need the format to have the same name as the original format (e.g. the default oai_dc format), you can create a new context in xoai.xsl containing your modified format with the original name, which will be available as /oai/context-name.

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

Driver/OpenAIRE compliance

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

- Driver context, which only exposes Driver compliant items;
- OpenAIRE context, which only exposes OpenAIRE compliant items;

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

Driver Compliance

DRIVER Guidelines for Repository Managers and Administrators on how to expose digital scientific resources using OAI-PMH and Dublin Core Metadata, creating interoperability by homogenizing the repository output. The OAI-PMH `driver` set is based on DRIVER Guidelines 2.0.

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

What specific metadata values are expected?

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

- Must have a publication date - dc.date.issued (already configured in DSpace items)
- dc.language must use ISO639-3
- the value of dc.type must be one of the 16 types named in the guidelines

How do you easily add those metadata values?

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

OpenAIRE compliance

For OpenAIRE v4 compliance, see OpenAIRE4 Guidelines Compliancy

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

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

See the Application Profile of OpenAIRE.

What specific metadata values are expected?

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

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

Optionally:

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

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

How do you easily add those metadata values?

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

Relevant Links

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

Sanity check your OAI interface with the OAI Validator

There is a very useful validator for OAI interfaces available at http://validator.oaipmh.com, we urge you to use this validator to confirm your OAI interface is in fact usable.

OAI-PMH Data Provider 2.0 (Internals)

1 OAI-PMH Data Provider 2.0 (Internals)
- 1.1 Sets
- 1.2 Unique Identifier
- 1.3 Access control
- 1.4 Modification Date (OAI Date Stamp)
- 1.5 "About" Information
- 1.6 Deletions
- 1.7 Flow Control (Resumption Tokens)

OAI-PMH Data Provider 2.0 (Internals)

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

The DSpace build process builds a single backend webapp, which optionally includes an OAI-PMH endpoint (when oai.enabled=true) In a typical configuration, this endpoint is deployed at `${dspace.server.url}/oai` (configured by "oai.path"), containing request, driver and openaire contexts, for example:

```
http://dspace.myu.edu/server/oai/request?verb=Identify
```

The "base URL" of this DSpace deployment would be:

```
http://dspace.myu.edu/server/oai/request
```

But one could also provide the Driver or OpenAIRE contexts:

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

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

DSpace provides implementations of the XOAI data sources interfaces.

Sets

OAI-PMH allows repositories to expose an hierarchy of sets in which records may be placed. A record can be in zero or more sets.

DSpace exposes collections and communities as sets.

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

```
col_1721.1_1234
```

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

Unique Identifier

Every item in OAI-PMH data repository must have 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:PREFIX:handle
```

For example:

```
oai:dspace.myu.edu:123456789/345
```

If you wish to use a different scheme, this can easily be changed by editing the value of identifier.prefix at [dspace]/config/modules/oai.cfg file.

Access control

OAI provides no authentication/authorisation details, although these could be implemented using standard HTTP methods. It is assumed that all access will be anonymous for the time being.

A question is, "is all metadata public?" Presently the answer to this is yes; all metadata is exposed via OAI-PMH, even if the item has restricted access policies. The reasoning behind this is that people who do actually have permission to read a restricted item should still be able to use OAI-based services to discover the content. But, exposed data could be changed by changing the XSLT defined at [dspace]/config/crosswalks/oai/metadataFormats.

Modification Date (OAI Date Stamp)

OAI-PMH harvesters need to know when a record has been created, changed or deleted. DSpace keeps track of a "last modified" date for each item in the system, and this date is used for the OAI-PMH date stamp. This means that any changes to the metadata (e.g. admins correcting a field, or a withdrawal) will be exposed to harvesters.

"About" Information

As part of each record given out to a harvester, there is an optional, repeatable "about" section which can be filled out in any (XML-schema conformant) way. Common uses are for provenance and rights information, and there are schemas in use by OAI communities for this. Presently DSpace does not provide any of this information, but XOAI core library allows its definition. This requires to dive into code and perform some changes.

Deletions

As DSpace supports two forms of deletions (withdrawals or permanent expunging), this has an impact on how OAI-PMH exposes delitions. During a permanent deletion (expunge), DSpace no longer retains any information about the deleted object. Therefore, permanent deletions "disappear" from OAI-PMH, as DSpace no longer has any information about the object. This is considered a "transient" approach to deletion based on OAI-PMH definitions.

When an item is withdrawn in DSpace, the item still exists but it hidden from public view. Withdrawn items will report a "<header status="deleted">" in OAI-PMH when a GetRecord request is made for a withdrawn item (however, they are NOT shown in an OAI-PMH "ListRecords" request by default). Keep in mind that the OAI-PMH index does NOT update automatically, so withdrawn items will not show this "deleted" status until "./dspace oai import" is next run.

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.

Flow Control (Resumption Tokens)

An OAI data provider can prevent any performance impact caused by harvesting by forcing a harvester to receive data in time-separated chunks. If the data provider receives a request for a lot of data, it can send part of the data with a resumption token. The harvester can then return later with the resumption token and continue.

DSpace supports resumption tokens for "ListRecords", "ListIdentifiers" and "ListSets" OAI-PMH requests.

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

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

Resumption tokens contain all the state information required to continue a request.

Exchanging Content Between Repositories

1 Transferring Content via Export and Import
    1.1 Transferring Communities, Collections, or Items using Packages
2 Transferring Items using Simple Archive Format
3 Transferring Items using OAI-ORE/OAI-PMH Harvester

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

For more information see AIP Backup and Restore.

Transferring Items using Simple Archive Format

Where items are to be moved between DSpace instances (for example from a test DSpace into a production DSpace) the Item Exporter and Item Importer can be used.

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

Transferring Items using OAI-ORE/OAI-PMH Harvester

You may also choose to enable the OAI-ORE Harvester. This OAI-ORE Harvester allows one DSpace installation to harvest Items (via OAI-PMH) from another DSpace Installation (or any other system supporting OAI-PMH). Items are harvested from a remote DSpace Collection into a local DSpace Collection. Harvesting can also be scheduled to run automatically (or by demand).

See OAI - Harvesting from another DSpace

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).

DSpace 7.0 does not yet support

The SWORDv1 Client is not available in DSpace 7.0. It may be restored in a later 7.x release, see DSpace Release 7.0 Status

1 Enabling the SWORD Client
2 Configuring the SWORD Client

Enabling the SWORD Client

The SWORDv1 Client is not available in DSpace 7.0. It may be restored in a later 7.x release, see DSpace Release 7.0 Status

Configuring the SWORD Client

All the relevant configuration can be found in `sword-client.cfg`. These may be overridden in your `local.cfg` config (see Configuration Reference).

Configuration File:	**`[dspace]/config/modules/sword-client.cfg`**
Property:	`sword-client.targets`

Example value:	`sword-client.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:	`sword-client.file-types`
Example value:	`sword-client.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:	`sword-client.package-formats`
Example value:	`sword-client.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.

Linked (Open) Data

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

Introduction

Exchanging repository contents

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

Terminology

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

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

Linked (Open) Data Support within DSpace

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

Architecture / Concept

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

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

Store public data only in the triple store!

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

The org.dspace.rdf.conversion package contains the classes used to convert the repository content to RDF. The conversion itself is done by plugins. The or g.dspace.rdf.conversion.ConverterPlugin interface is really simple, so take a look at it you if can program in Java and want to extend the conversion. The only thing important is that plugins must only create RDF that can be made publicly available, as the triple store provides it using a sparql endpoint for which the DSpace access restrictions do not apply. Plugins converting metadata should check whether a specific metadata field needs to be protected or not (see org.dspace.app.util.MetadataExposure on how to check that). The MetadataConverterPlugin is heavily configurable (see below) and is used to convert the metadata of Items. The StaticDSOConverterPlugin can be used to add static RDF Triples (see below). The SimpleDSORelationsConverterPlugi creates links between items and collections, collections and communities, subcommunitites and their parents, and between top-level communities and the information representing the repository itself.

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

Install a Triple Store

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

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

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

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

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

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

Default configuration and what you should change

First, you'll want to ensure the Linked Data endpoint is enabled/configured. In your `local.cfg`, add `rdf.enabled = true`. You can optionally change it's path by setting `rdf.path` (it defaults to "rdf" which means the Linked Data endpoint is available at `[dspace.server.url]/rdf/` (where `dspace.server.url` is also specified in your `local.cfg`)

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

As the Linked Data support of DSpace is highly configurable this section gives a short list of things you probably want to configure before using it. Below you can find more information on what is possible to configure.

In the file `[dspace]/config/modules/rdf.cfg` you want to configure the address of the public sparql endpoint and the address of the writable endpoint DSpace use to connect to the triple store (the properties `rdf.public.sparql.endpoint`, `rdf.storage.graphstore.endpoint`). In the same file you want to configure the URL that addresses the dspace-rdf module which is depending on where you deployed it (property `rdf.contextPath`) and switch content negotiation on (set property `rdf.contentNegotiation.enable = true`).

In the file `[dspace]/config/modules/rdf/constant-data-general.ttl` you should change the links to the Web UI of the repository and the public readable SPARQL endpoint. The URL of the public SPARQL endpoint should point to a URL that is proxied by a webserver to the Triple Store. See the section Install a Triple Store above for further information.

In the file `[dspace]/config/modules/rdf/constant-data-site.ttl` you may add any triples that should be added to the description of the repository itself.

If you want to change the way the metadata fields are converted, take a look into the file `[dspace]/config/modules/rdf/metadata-rdf-mapping.ttl`. This is also the place to add information on how to map metadata fields that you added to DSpace. There is already a quite acceptable default configuration for the metadata fields which DSpace supports out of the box. If you want to use some specific prefixes in RDF serializations that support prefixes, you have to edit `[dspace]onfig/modules/rdf/metadata-prefixes.ttl`.

Configuration Reference

There are several configuration files to configure DSpace's LOD support. The main configuration file can be found under `[dspace-source]/dspace/config/modules/rdf.cfg`. Within DSpace we use Spring to define which classes to load. For DSpace's LOD support this is done within `[dspace-source]/dspace/config/spring/api/rdf.xml`. All other configuration files are positioned in the directory `[dspace-source]/dspace/config/modules/rdf/`. Configurations in `rdf.cfg` can be modified directly, or overridden via your `local.cfg` config file (see Configuration Reference). You'll have to configure where to find and how to connect to the triple store. You may configure how to generate URIs to be used within the generated Linked Data and how to convert the contents stored in DSpace into RDF. We will guide you through the configuration file by file.

[dspace-source]/dspace/config/modules/rdf.cfg

Property:	rdf.enabled
Example Value:	rdf.enabled = true
Informational Note:	Defines whether the RDF endpoint is enabled or disabled (disabled by default). If enabled, the RDF endpoint is available at ${dspace.server.url}/${rdf.path}. Changing this value requires rebooting your servlet container (e.g. Tomcat)
Property:	rdf.path
Example Value:	rdf.path = rdf
Informational Note:	Defines the path of the RDF endpoint, if enabled. For example, a value of "rdf" (the default) means the RDF interface/endpoint is available at ${dspace.server.url}/rdf (e.g. if "dspace.server.url = http://localhost:8080/server", then it'd be available at "http://localhost:8080/server/rdf". Changing this value requires rebooting your servlet container (e.g. Tomcat)

Property:	rdf.contentNegotiation.enable
ExampleValue:	rdf.contentNegotiation.enable = true
InformationalNote:	Defines whether content negotiation should be activated. Set this true, if you use Linked Data support.
Property:	rdf.contextPath
ExampleValue:	rdf.contextPath = ${dspace.baseUrl}/rdf
InformationalNote:	The content negotiation needs to know where to refer if anyone asks for RDF serializations of content stored within DSpace. This property sets the URL where the dspace-rdf module can be reached on the Internet (depending on how you deployed it).
Property:	rdf.public.sparql.endpoint
ExampleValue:	rdf.public.sparql.endpoint = http://${dspace.baseUrl}/sparql
InformationalNote:	Address of the read-only public SPARQL endpoint supporting SPARQL 1.1 Query Language.
Property:	rdf.storage.graphstore.endpoint
ExampleValue:	rdf.storage.graphstore.endpoint = http://localhost:3030/dspace/data
InformationalNote:	Address of a writable SPARQL 1.1 Graph Store HTTP Protocol endpoint. This address is used to create, update and delete converted data in the triple store. If you use Fuseki with the configuration provided as part of DSpace 5, you can leave this as it is. If you use another Triple Store or configure Fuseki on your own, change this property to point to a writeable SPARQL endpoint supporting the SPARQL 1.1 Graph Store HTTP Protocol.

Property:	rdf.storage.graphstore.authentication
Example Value:	rdf.storage.graphstore.authentication = no
Informational Note:	Defines whether to use HTTP Basic authentication to connect to the writable SPARQL 1.1 Graph Store HTTP Protocol endpoint.
Properties:	rdf.storage.graphstore.login rdf.storage.graphstore.password
Example Values:	rdf.storage.graphstore.login = dspace rdf.storage.graphstore.password =ecapsd
Informational Note:	Credentials for the HTTP Basic authentication if it is necessary to connect to the writable SPARQL 1.1 Graph Store HTTP Protocol endpoint.
Property:	rdf.storage.sparql.endpoint
Example Value:	rdf.storage.sparql.endpoint = http://localhost:3030/dspace/sparql
Informational Note:	Besides a writable SPARQL 1.1 Graph Store HTTP Protocol endpoint, DSpace needs a SPARQL 1.1 Query Language endpoint, which can be read-only. This property allows you to set an address to be used to connect to such a SPARQL endpoint. If you leave this property empty the property ${rdf.public.sparql.endpoint} will be used instead.
Properties:	rdf.storage.sparql.authentication rdf.storage.sparql.login rdf.storage.sparql.password
Example Values:	rdf.storage.sparql.authentication = yes rdf.storage.sparql.login = dspace rdf.storage.sparql.password = ecapsd
Informational Note:	As for the SPARQL 1.1 Graph Store HTTP Protocol you can configure DSpace to use HTTP Basic authentication to authenticate against the (read-only) SPARQL 1.1 Query Language endpoint.

Property:	rdf.converter.DSOtypes
Example Value:	rdf.converter.DSOtypes = SITE, COMMUNITY, COLLECTION, ITEM
Informational Note:	Define which kind of DSpaceObjects should be converted. Bundles and Bitstreams will be converted as part of the Item they belong to. Don't add EPersons here unless you really know what you are doing. All converted data is stored in the triple store that provides a publicly readable SPARQL endpoint. So all data converted into RDF is exposed publicly. Every DSO type you add here must have an HTTP URI to be referenced in the generated RDF, which is another reason not to add EPersons here currently.
The following properties configure the StaticDSOConverterPlugin.	
Properties:	rdf.constant.data.GENERAL rdf.constant.data.COLLECTION rdf.constant.data.COMMUNITY rdf.constant.data.ITEM rdf.constant.data.SITE
Example Values:	rdf.constant.data.GENERAL = ${dspace.dir}/config/modules/rdf/constant-data-general.ttl rdf.constant.data.COLLECTION = ${dspace.dir}/config/modules/rdf/constant-data-collection.ttl rdf.constant.data.COMMUNITY = ${dspace.dir}/config/modules/rdf/constant-data-community.ttl rdf.constant.data.ITEM = ${dspace.dir}/config/modules/rdf/constant-data-item.ttl rdf.constant.data.SITE = ${dspace.dir}/config/modules/rdf/constant-data-site.ttl
Informational Note:	These properties define files to read static data from. These data should be in RDF, and by default Turtle is used as serialization. The data in the file referenced by the property ${rdf.constant.data.GENERAL} will be included in every Entity that is converted to RDF. E.g. it can be used to point to the address of the public readable SPARQL endpoint or may contain the name of the institution running DSpace. The other properties define files that will be included if a DSpace Object of the specified type (collection, community, item or site) is converted. This makes it possible to add static content to every Item, every Collection, ...
The following properties configure the MetadataConverterPlugin.	
Property:	rdf.metadata.mappings
Example Value:	rdf.metadata.mappings = ${dspace.dir}/config/modules/rdf/metadata-rdf-mapping.ttl
Informational Note:	Defines the file that contains the mappings for the MetadataConverterPlugin. See below the description of the configuration file [dspace-source] /dspace/config/modules/rdf/metadata-rdf-mapping.ttl.
Property:	rdf.metadata.schema
Example Value:	rdf.metadata.schema = file://${dspace.dir}/config/modules/rdf/metadata-rdf-schema.ttl

Informational Note:	Configures the URL used to load the RDF Schema of the DSpace Metadata RDF mapping Vocabulary. Using a file:// URI makes it possible to convert DSpace content without having an internet connection. The version of the schema has to be the right one for the used code. In DSpace 5.0 we use the version 0.2.0. This Schema can be found here as well: http://digital-repositories.org/ontologies/dspace-metadata-mapping/0.2.0. The newest version of the Schema can be found here: http://digital-repositories.org/ontologies/dspace-metadata-mapping/.
Property:	rdf.metadata.prefixes
Example Value:	rdf.metadata.prefixes = ${dspace.dir}/config/modules/rdf/metadata-prefixes.ttl
Informational Note:	If you want to use prefixes in RDF serializations that support prefixes, you can define these prefixes in the file referenced by this property.
The following properties configure the SimpleDSORelationsConverterPlugin	
Property:	rdf.simplerelations.prefixes
Example Value:	rdf.simplerelations.prefixes = ${dspace.dir}/config/modules/rdf/simple-relations-prefixes.ttl
Informational Note:	If you want to use prefixes in RDF serializations that support prefixes, you can define these prefixes in the file referenced by this property.
Property:	rdf.simplerelations.site2community
Example Value:	rdf.simplerelations.site2community = http://purl.org/dc/terms/hasPart, http://digital-repositories.org/ontologies/dspace/0.1.0#hasCommunity
Informational Note:	Defines the predicates used to link from the data representing the whole repository to the top level communities. Defining multiple predicates separated by commas will result in multiple triples.
Property:	rdf.simplerelations.community2site
Example Value:	rdf.simplerelations.community2site = http://purl.org/dc/terms/isPartOf, http://digital-repositories.org/ontologies/dspace/0.1.0#isPartOfRepository

Informational Note:	Defines the predicates used to link from the top level communities to the data representing the whole repository. Defining multiple predicates separated by commas will result in multiple triples.
Property:	rdf.simplerelations.community2subcommunity
Example Value:	rdf.simplerelations.community2subcommunity = http://purl.org/dc/terms/hasPart, http://digital-repositories.org/ontologies/dspace/0.1.0#hasSubcommunity
Informational Note:	Defines the predicates used to link from communities to their subcommunities. Defining multiple predicates separated by commas will result in multiple triples.
Property:	rdf.simplerelations.subcommunity2community
Example Value:	rdf.simplerelations.subcommunity2community = http://purl.org/dc/terms/isPartOf, http://digital-repositories.org/ontologies/dspace/0.1.0#isSubcommunityOf
Informational Note:	Defines the predicates used to link from subcommunities to the communities they belong to. Defining multiple predicates separated by commas will result in multiple triples.
Property:	rdf.simplerelations.community2collection
Example Value:	rdf.simplerelations.community2collection = http://purl.org/dc/terms/hasPart, http://digital-repositories.org/ontologies/dspace/0.1.0#hasCollection
Informational Note:	Defines the predicates used to link from communities to their collections. Defining multiple predicates separated by commas will result in multiple triples.
Property:	rdf.simplerelations.collection2community
Example Value:	rdf.simplerelations.collection2community = http://purl.org/dc/terms/isPartOf, http://digital-repositories.org/ontologies/dspace/0.1.0#isPartOfCommunity

Informational Note:	Defines the predicates used to link from collections to the communities they belong to. Defining multiple predicates separated by commas will result in multiple triples.
Property:	rdf.simplerelations.collection2item
Example Value:	rdf.simplerelations.collection2item = http://purl.org/dc/terms/hasPart, http://digital-repositories.org/ontologies/dspace/0.1.0#hasItem
Informational Note:	Defines the predicates used to link from collections to their items. Defining multiple predicates separated by commas will result in multiple triples.
Property:	rdf.simplerelations.item2collection
Example Value:	rdf.simplerelations.item2collection = http://purl.org/dc/terms/isPartOf, http://digital-repositories.org/ontologies/dspace/0.1.0#isPartOfCollection
Informational Note:	Defines the predicates used to link from items to the collections they belong to. Defining multiple predicates separated by commas will result in multiple triples.
Property:	rdf.simplerelations.item2bitstream
Example Value:	rdf.simplerelations.item2bitstream = http://purl.org/dc/terms/hasPart, http://digital-repositories.org/ontologies/dspace/0.1.0#hasBitstream
Informational Note:	Defines the predicates used to link from item to their bitstreams. Defining multiple predicates separated by commas will result in multiple triples.

[dspace-source]/dspace/config/modules/rdf/constant-data-*.ttl

As described in the documentation of the configuration file [dspace-source]/dspace/config/modules/rdf.cfg, the constant-data-*.ttl files can be used to add static RDF to the converted data. The data are written in Turtle, but if you change the file suffix (and the path to find the files in rdf.cfg) you can use any other RDF serialization you like to. You can use this, for example, to add a link to the public readable SPARQL endpoint, add a link to the repository homepage, or add a triple to every community or collection defining it as an entity of a specific type like a bibo:collection. The content of the file [dspace-source]/dspace/config/modules/rdf/constant-data-general.ttl will be added to every DSpaceObject that is converted. The content of the file [dspace-source] /dspace/config/modules/rdf/constant-data-community.ttl to every community, the content of the file [dspace-source]/dspace/config/modules/rdf/constant-data-collection.ttl to every collection and the content of the file [dspace-source]/dspace/config/modules/rdf/constant-data-item.ttl to every Item. You can use the file [dspace-source]/dspace/config/modules/rdf/constant-data-site.ttl to specify data representing the whole repository.

[dspace-source]/dspace/config/modules/rdf/metadata-rdf-mapping.ttl

This file should contain several metadata mappings. A metadata mapping defines how to map a specific metadata field within DSpace to a triple that will be added to the converted data. The MetadataConverterPlugin uses these metadata mappings to convert the metadata of a item into RDF. For every metadata field and value it looks if any of the specified mappings matches. If one does, the plugin creates the specified triple and adds it to the converted data. In the file you'll find a lot of examples on how to define such a mapping.

For every mapping a metadata field name has to be specified, e.g. dc.title, dc.identifier.uri. In addition you can specify a condition that is matched against the field's value. The condition is specified as a regular expression (using the syntax of the java class java.util.regex.Pattern). If a condition is defined, the mapping will be used only on fields those values which are matched by the regex defined as condition.

The triple to create by a mapping is specified using reified RDF statements. The DSpace Metadata RDF Mapping Vocabulary defines some placeholders that can be used. The most important placeholder is dm:DSpaceObjectIRI which is replaced by the URI used to identify the entity being converted to RDF. That means if a specific Item is converted the URI used to address this Item in RDF will be used instead of dm:DSpaceObjectIRI. There are three placeholders that allow reuse of the value of a meta data field. dm:DSpaceValue will be replace by the value as it is. dm:LiteralGenerator allows one to specify a regex and replacement string for it (see the syntax of the java classes java.util.regex.Pattern and java.util.regex.Matcher) and creates a Literal out of the field value using the regex and the replacement string. dm:ResourceGenerator does the same as dm:LiteralGenerator but it generates a HTTP (S) URI that is used in place. So you can use the resource generator to generate URIs containing modified field values (e.g. to link to classifications). If you know regular expressions and turtle, the syntax should be quite self explanatory.

[dspace-source]/dspace/config/modules/rdf/fuseki-assembler.ttl

This is a configuration for the triple store Fuseki of the Apache Jena project. You can find more information on the configuration it provides in the section Install a Triple Store above.

[dspace-source]/dspace/config/spring/api/rdf.xml

This file defines which classes are loaded by DSpace to provide the RDF functionality. There are two things you might want to change: the class that is responsible to generate the URIs to be used within the converted data, and the list of Plugins used during conversion. To change the class responsible for the URIs, change the following line:

```
<property name="generator" ref="org.dspace.rdf.storage.LocalURIGenerator"/>
```

This line defines how URIs should be generated, to be used within the converted data. The LocalURIGenerator generates URIs using the ${dspace.url} property. The HandleURIGenerator uses handles in form of HTTP URLs. It uses the property ${handle.canonical.prefix} to convert handles into HTTPS URLs. The class org.dspace.rdf.storage.DOIURIGenerator uses DOIs in the form of HTTP URLs if possible, or local URIs if there are no DOIs. It uses the DOI resolver "http://dx.doi.org" to convert DOIs into HTTP URLs. The class org.dspace.rdf.storage.DOIHandleGenerator does the same but uses Handles as fallback if no DOI exists. The fallbacks are necessary as DOIs are currently used for Items only and not for Communities or Collections.

All plugins that are instantiated within the configuration file will automatically be used during the conversion. Per default the list looks like the following:

```
<!-- configure all plugins the converter should use. If you don't want to
         use a plugin, remove it here. -->
    <bean id="org.dspace.rdf.conversion.SimpleDSORelationsConverterPlugin" class="org.dspace.rdf.conversion.
SimpleDSORelationsConverterPlugin"/>
    <bean id="org.dspace.rdf.conversion.MetadataConverterPlugin" class="org.dspace.rdf.conversion.
MetadataConverterPlugin"/>
    <bean id="org.dspace.rdf.conversion.StaticDSOConverterPlugin" class="org.dspace.rdf.conversion.
StaticDSOConverterPlugin"/>
```

You can remove plugins if you don't want them. If you develop a new conversion plugin, you want to add its class to this list.

Maintenance

As described above you should add `rdf` to the property `event.dispatcher.default.consumers` and in dspace.cfg. This configures DSpace to automatically update the triple store every time the publicly available content of the repository is changed. Nevertheless there is a command line tool that gives you the possibility to update the content of the triple store. As the triple store is used as a cache only, you can delete its content and reindex it every time you think it is necessary of helpful. The command line tool can be started by the following command which will show its online help:

```
[dspace-install]/bin/dspace rdfizer --help
```

The online help should give you all necessary information. There are commands to delete one specific entity; to delete all information stored in the triple store; to convert one item, one collection or community (including all subcommunities, collections and items) or to convert the complete content of your repository. If you start using the Linked Open Data support on a repository that already contains content, you should run `[dspace-install]/bin/dspace rdfizer --convert-all` once.

Every time content of DSpace is converted or Linked Data is requested, DSpace will try to connect to the triple store. So ensure that it is running (as you do with e.g. your sevlet container or relational database).

Rioxx v3 schema compliance

Loading of Entities and Fields

To achieve full Rioxx v3 compliance, there is no need to enable DSpace entitites. However, if entities are enabled, richer metadata such as funding or project related information can be then exposed via the OAI interface. To expose this information, the following steps need to be performed:

First, load the Rioxx Entities model by running the following command:

loading rioxx entity-relationships model

```
[/dspace]/bin/dspace initialize-entities -f [/dspace]/config/entities/rioxx3-relationships.xml
```

(**NOTE**: the openaire4-relationships.xml model can be loaded instead, as the set of relationships needed to support entities are exactly the same as those needed by Rioxx)

To support Rioxx metadata, there is no need to load additional registries, as it uses metadata from Dublin Core and Qualified Dublin Core schemas.

OAI interface

For Rioxx v3, only Publication entities or non-entity item types will be loaded as the main Entity to be processed, and any related entities such as projects or funders (i.e. FundingAgency) will be loaded in the process via virtual metadata.

A Rioxx v3 compliant endpoint is available in a dedicated OAI context, namely **rioxx**, through the URL:

`http://[dspace-server-url]/oai/rioxx?verb=ListRecords&metadataPrefix=rioxx`

To be able to use it, check first that that the oai application is enabled by checking the **oai.cfg** setting:

`oai.enabled = true`
(**NOTE**: when enabling you may need to restart your tomcat service)

To make metadata available from related entities such as projects or funders, you need to enable the relevant virtual metadata file. The Rioxx context makes use of the virtual metadata from openaire4. To enable this, you have to copy the file:

`[/dspace/]config/spring/api/virtual-metadata.xml.openaire4`

into the default virtual-metadata file:

```
cp [/dspace/]config/spring/api/virtual-metadata.xml.openaire4 [/dspace/]config/spring/api/virtual-metadata.xml
```

(**NOTE**: if you do this you should restart your tomcat service container)

This additional virtual metadata will enable to represent something like this in this XML in the **rioxx metadata format**, where you have, for instance, author identifiers, or funding information:

oai datacite:creators example

```
<rioxxterms:creator>
      <rioxxterms:id>https://orcid.org/0000-0001-7656-9453</rioxxterms:id>
      <rioxxterms:name>Pathan, Nazima</rioxxterms:name>
</rioxxterms:creator>
...
<rioxxterms:grant funder_name="NIHR HTA Programme">16/152/01</rioxxterms:grant>
```

Then you may need to run the OAI import from the command line with the cleaning cache parameter to reload all data to OAI:
`[/dspace/]/bin/dspace oai import -c`

Ingesting Content and Metadata

This is a new top level page grouping all documentation concerning all different ways to ingest content and metadata into DSpace

- Submission User Interface
- Configurable Workflow
- Importing and Exporting Content via Packages
- Importing and Exporting Items via Simple Archive Format
- Registering Bitstreams via Simple Archive Format
- Importing Items via basic bibliographic formats (Endnote, BibTex, RIS, CSV, etc) and online services (arXiv, PubMed, CrossRef, CiNii, etc)
- Exporting and Importing Community and Collection Hierarchy
- SWORDv1 Server
- SWORDv2 Server
- Ingesting HTML Archives

The section on **Batch Metadata Editing** also contains information on how to add items through spreadsheet ingest.

Submission User Interface

This page explains various customization and configuration options that are available within DSpace for the Item Submission user interface.

1 Default Submission Process
    1.1 Optional Steps
2 Understanding the Submission Configuration Files
    2.1 The Structure of item-submission.xml
    2.2 Defining Steps (<step>) within the item-submission.xml
        2.2.1 Where to place your <step-definition>
        2.2.2 The ordering of <step> tags matter!
        2.2.3 Structure of the <step-definition> tag
3 Reordering/Removing/Adding Submission Steps
4 Assigning a custom Submission Process to a Collection
    4.1 Getting A Collection's Handle
    4.2 Assigning a default Submission Process per Entity Type
5 Custom Metadata-entry Steps for Submission
    5.1 Introduction
    5.2 Describing Custom Metadata Forms
    5.3 The Structure of submission-forms.xml
        5.3.1 Using a form in a submission process for a Collection
        5.3.2 Adding a Form
            5.3.2.1 Forms and Pages
            5.3.2.2 Composition of a Field
                5.3.2.2.1 Visibility configuration examples
            5.3.2.3 Item type Based Metadata Collection
            5.3.2.4 Composition of a Relation Field
        5.3.3 Configuring Controlled Vocabularies
        5.3.4 Adding Value-Pairs
            5.3.4.1 Example
    5.4 Deploying Your Custom Forms
6 Configuring the File Upload step
    6.1 Basic Settings
    6.2 Modifying metadata form presented for Bitstreams
    6.3 Modifying access conditions (embargo, etc.) presented for Bitstreams
    6.4 Define upload of files mandatory or optional per collection
7 Configuring the Item Access Conditions step
    7.1 Enabling the step
    7.2 Modifying access conditions (embargo, etc.) presented for Items
    7.3 Examples of selecting Item and Bitstream access conditions
8 Configuring the Sherpa Romeo step
    8.1 Enabling the Sherpa Romeo step
9 Configuring the "Identifiers" step
10 Creating new Submission Steps Programmatically.

DSpace Submission Configuration changed in v7.x

The name and structure of the Submission configuration files changed in 7.x. The DSpace 6.x (and below) "item-submission.xml" and "input-forms.xml" configuration files are no longer supported. In 7.x and above, the format of the "item-submission.xml" file has been updated, and the older "input-forms.xml" has been replaced by a new "submission-forms.xml".

You can choose to either start fresh with the new v7 configuration files (see documentation below) and/or use the "./dspace submission-forms-migrate" script to migrate your old configurations into new ones. See the Upgrading DSpace guide (step on "Update your DSpace Configurations") for more information on using the migration script.

Default Submission Process

The DSpace Submission process consists of a series of "steps", where each "step" corresponds to one or "sections" in the Submission UI. By default, the DSpace Submission process includes the following steps/sections, in this order:

1. **"Select Collection"** (id="collection"), appears as dropdown: If not already selected, the user must select a collection to deposit the Item into. As of DSpace 7, you also can change the Collection you are submitting into at any time. However, be aware that there may be some metadata lost if the Collection you switch two uses a *different* submission form & you already began entering metadata in the current submission.
2. **"Describe" sections** (id="traditionalpageone" and "traditionalpagetwo"): This is where the user may enter descriptive metadata about the Item. This step may consist of one or more sections of metadata entry. By default, there are two sections of metadata-entry . For information on modifying the metadata entry pages, please see Custom Metadata-entry Pages for Submission section below.
3. **"Upload" section** (id="upload"): This is where the user may upload one or more files to associate with the Item. As of DSpace 7, you can also drag and drop files anywhere on the page to trigger an upload. For more information on file upload, also see Configuring the File Upload step below.
4. **"License" section** (id="license"): This is where the user **must** agree to the repository distribution license in order to complete the deposit. This repository distribution license is defined in the `[dspace]/config/default.license` file. It can also be customized per-collection from the Collection Edit UI.

5. **"Deposit" button**: Once all required fields/sections are completed, the "Deposit" button becomes enabled. After clicking it, the new Item will either become immediately available or undergo a workflow approval process (depending on the Collection policies). For more information on the workflow approval process see Configurable Workflow

To modify or reorganize these submission steps, just modify the `[dspace]/config/item-submission.xml` file. Please see the section below on Reordering/Removing/Adding Submission Steps.

You can also choose to have different submission processes for different DSpace Collections. For more details, please see the section below on Assigning a custom Submission Process to a Collection.

Optional Steps

DSpace also ships with several optional steps which you may choose to enable if you wish. In no particular order:

- **"Item Access" (or Embargo) section** (id="itemAccessConditions"): *Only available in 7.2 or above.* This step allows the user to (optionally) modify access rights or set an embargo during the deposit of an Item. For more information on this step, and Embargo options in general, please see the Embargo documentation.
- **"CC License" section** (id="cclicense"): This step allows the user to (optionally) assign a Creative Commons license to a particular Item. Please see the Configuring Creative Commons License section of the Configuration documentation for more details.
- **"Extraction" section** (id="extractionstep"): This step will automatically attempt to extract metadata from uploaded files. Currently it only supports bibliographic formats documented in Importing Items via basic bibliographic formats (Endnote, BibTex, RIS, TSV, CSV) and online services (OAI, arXiv, PubMed, CrossRef, CiNii). Any extracted metadata is immediately populated in the submission form (without notifying the user).
    - By default it is disabled, as it populates metadata automatically (without notifying the user). This means it can sometimes result in duplicative metadata in the submission form.
    - The behavior of this step can be more fully configured via the 'config/spring/api/step-processing-listener.xml' configuration
    - NOTE: this action is also only triggered when a request is performed (e.g. when a file is uploaded or the submission form is saved). You can configure the Angular UI to autosave based on a timer in order to force this action to be done more regularly.
- Various Configurable Entities related steps: These steps are "Describe" steps that are specific to different Entity types. They provide a list of metadata fields of specific interest to those Entities.

To enable any of these optional submission steps, just uncomment the step definition within the `[dspace]/config/item-submission.xml` file. Please see the section below on Reordering/Removing/Adding Submission Steps.

You can also choose to enable certain steps only for specific DSpace Collections. For more details, please see the section below on Assigning a custom Submission Process to a Collection.

Understanding the Submission Configuration Files

The `[dspace]/config/item-submission.xml` contains the submission configurations for the DSpace UI. This configuration file contains detailed documentation within the file itself, which should help you better understand how to best utilize it.

The Structure of *item-submission.xml*

The structure of this file changed slightly in DSpace 7

As of DSpace 7, the following structural changes were made to item-submission.xml:

- Step definitions under <step-definitions> now use the <step-definition> tag (previously, in 6.x, the tag was named <step>)
- Every step definition now needs to be defined under <step-definitions> (previously, in 6.x, you could also define them in <submission-process>), and have a unique ID
- Each <step-definition> now only represents a single "section" of the Submission UI. (previously, in 6.x, some steps like Describe represented multiple pages)
- An attribute "mandatory=[true|false]" was added to the <step> element. When true, that section is always displayed to the user. When false, it's not displayed by default, but instead must be activated explicitly by the user by choosing to add the section in the Submission UI.
- The old <workflow-editable> element has been replaced with a <scope> element which defines when/how this <step> should be displayed.

Because this file is in XML format, you should be familiar with XML before editing this file. By default, this file contains the "traditional" Item Submission Process for DSpace, which consists of the following Steps (in this order):

Select Collection -> Describe (two steps) -> Upload -> License -> Complete

If you would like to customize the steps used or the ordering of the steps, you can do so within the <*submission-definition*> section of the *item-submission.xml* .

In addition, you may also specify different Submission Processes for different DSpace Collections. This can be done in the <*submission-map*> section. The *item-submission.xml* file itself documents the syntax required to perform these configuration changes.

Defining Steps (<*step*>) within the *item-submission.xml*

This section describes how Steps of the Submission Process are defined within the *item-submission.xml*.

Where to place your <*step-definition*>

The <*step-definition*> always appear within the <*step-definitions*> section of the *item-submission.xml* configuration file.

- This section allows all <*step*> definitions to be defined globally (i.e. so they may be 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-definition 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"/*>

The ordering of <*step*> tags matter!

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 *Describe* 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 id="license"/>

    <!--Step 2 & 3 will be to ask for metadata-->
    <step id="traditionalpageone"/>
    <step id="traditionalpagetwo"/>

        ...[other steps]...
</submission-process>
```

Structure of the <step-definition> tag

The structure of the <step-definition> tag is as follows:

```
<step-definition id="traditionalpageone" mandatory="true">
    <heading>submit.progressbar.describe.stepone</heading>
    <processing-class>org.dspace.app.rest.submit.step.DescribeStep</processing-class>
    <type>submission-form</type>
    <!-- <scope visibility="hidden" visibilityOutside="hidden">submission</scope> -->
</step-definition>
```

Each *step* contains the following elements/attributes. The required elements are so marked:

- **mandatory** *(attribute)*: [true|false] When true, the step's section is displayed by default to all users in the UI. When false, the step is not displayed and must be activated explicitly by the user by selecting it in the UI or supplying data of interest to the section.
- **heading**: Partial I18N key (defined in the UI's language packs) which corresponds to the text that should be displayed in section header for this step. This partial I18N key is prefixed with "submission.sections.". Therefore, the full i18n key is "submission.sections.[heading]" in the User Interface's language packs (e.g. en.json5 for English)
- **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. 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.*`)
- **type (Required)**: The type of step defined. Most steps are of type "submission-form", which means they directly map to a `<form>` defined in the s`ubmission-forms.xml` configuration file. In this situation, the `<step-definition>` "id" attribute MUST map to a `<form>` "name" attribute defined in submission-forms.xml. Any value is allowed, and only "submission-form" has a special meaning at this time.
- **scope:** Optionally, allows you to limit the "scope" of this particular step, and define whether the step is visible outside that scope. Valid scope values include "submission" (limited to the submission form) and "workflow" (limited to workflow approval process).
    - "visibility" attribute defines the visibility of the step while **within** the given scope. Can be set to "read-only" (in this scope you can see this step but not edit it), or "hidden" (in this scope you cannot see this step).
    - "visibilityOutside" attribute defines the visibility of the step while **outside** the given scope. Can be set to "read-only" (in other scopes you can see this step but not edit it), or "hidden" (in other scopes you cannot see this step).

Reordering/Removing/Adding Submission Steps

The removal of existing steps and reordering of existing steps is a relatively easy process!

Reordering steps

1. Locate the `<submission-process>` tag which defines the Submission Process that you are using. If you are unsure which Submission Process you are using, it's likely the one with `name="traditional"`, since this is the traditional DSpace submission process.
2. Reorder the `<step>` tags within that `<submission-process>` tag. Be sure to move the *entire* `<step>` tag.

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.*
    - *Hint:* You cannot remove the *"collection"* step, as an DSpace Item cannot exist without belonging to a Collection.

Adding one or more optional steps

1. Locate the `<submission-process>` tag which defines the Submission Process that you are using. If you are unsure which Submission Process you are using, it's likely the one with `name="traditional"`, since this is the traditional DSpace submission process.
2. Uncomment (i.e. remove the `<! --` and `-->`) the `<step>` tag(s) which you want to add to that `<submission-process>` tag. Be sure to uncomment the *entire* `<step>` *tag.*

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.

There are three ways to configure this mapping:

1. The traditional way is to use the "collection-handle" attribute to map a submission form to it's Collection. 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*.
    a. 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>
```

2. Another option is to use the "collection-entity-type" attribute to map *all Collections* which use that Entity Type (requires Configurable Entities) to a specific submission definition name (via the *submission-name* attribute, similar to above).
    a. For example, the following fragment shows how to map all Collections which use the out-of-the-box Entity Types to a submission definition of the same name:

```
<submission-map>
    ...
    <name-map collection-entity-type="Publication" submission-name="Publication"/>
    <name-map collection-entity-type="Person" submission-name="Person"/>
    <name-map collection-entity-type="Project" submission-name="Project"/>
    <name-map collection-entity-type="OrgUnit" submission-name="OrgUnit"/>
    <name-map collection-entity-type="Journal" submission-name="Journal"/>
    <name-map collection-entity-type="JournalVolume" submission-name="JournalVolume"/>
    <name-map collection-entity-type="JournalIssue" submission-name="JournalIssue"/>
    ...
<submission-map>
```

3. Finally, it is also possible to use the "community-handle" attribute to map a submission process to all descendant collections that do not have a direct mapping assigned by collection handle or by entity type (also via the submission-name attribute).
    a. For example, the following fragment shows how the descendant collections of the community with handle "12345.6789/38" are assigned the "custom2" submission process:

```
<submission-map>
    <name-map community-handle="12345.6789/38" submission-name="custom2" />
    ...
</submission-map>

<submission-definitions>
    <submission-process name="custom2">
```

```
    ...
</submission-definitions>
```

b. If a collection has multiple parent communities with a defined mapping, the collection will use the submission process mapped for the closest community.

It's a good idea to keep the definition of the *default* name-map, 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 Community or Collection in the DSpace UI. Look for the "Permanent URI" listed near the top of the page. It should look something like:

```
http://myhost.my.edu/handle/12345.6789/42
```

The handle is everything after "handle/" (in the above example it is "12345.6789/42"). It should look familiar to any DSpace administrator. That is what goes in the *collection-handle* attribute of your *name-map* element.

Assigning a default Submission Process per Entity Type

Alternatively to a collection's Handle, Entities Types can be used as an attribute. With these configurations you will enable default submission forms per Entity type. You don't have to specify every collection's handle to use for a particular submission form if you intend to use entities.
In order to do it so, instead of `collection-handle` attribute you need to use `collection-entity-type`. The possible values for this attribute are the ones that you use or that you specified in `relationship-types.xml` file (please check the documentation for more information). In order the submission process to be assigned to an entity type, you need to previously have associated an Entity Type to a Collection (please check: Configurable Entities#3.ConfigureCollectionsforeachEntitytype).

As an example, for every time you need to insert a new person in a Person's collection. You just need to specify the submission form to be used, like: `submission-name="customPerson"` in the example and also the entity type that is associated, like `collection-entity-type="Person"`.

```
<submission-map>
    <name-map collection-entity-type="Person" submission-name="customPerson" />
    ...
</submission-map>

<submission-definitions>
    <submission-process name="customPerson">
    ...
</submission-definitions>
```

If a collection `collection-handle="12345.6789/42"` configuration will prevail over this configuration. Meaning that if a `collection-entity-type` is defined and a collection-handle is also defined and if a collection handle overlaps in both configurations, then, the submission to be considered it will be the one that is defined by `collection-handle` (it will prevail the one with more granularity).

Custom Metadata-entry Steps for Submission

Introduction

This section explains how to customize the Web forms used by submitters and editors to enter and modify the metadata for a new item. These metadata web forms are controlled by the *Describe* step within the Submission Process. However, they are also configurable via their own XML configuration file `[dspace]/config/submission-forms.xml`.

In this configuration you can create alternate metadata forms, which can then be mapped to a "submission-form" step in the "item-submission.xml" (see above).

In creating custom metadata forms, you can choose:

- Which fields appear on each form, and their sequence. (Keep in mind, each "form" represents to a "step" or section)
- Labels, prompts, and other text associated with each field.
- Ability to display smaller fields side-by-side in a single "row"
- List of available choices for each menu-driven field.

All of the custom metadata-entry forms for a DSpace instance are controlled by a single XML file, `submission-forms.xml`, in the *config* subdirectory under the DSpace home, `[dspace]/config/submission-forms.xml`. DSpace comes with a number of sample forms which implement the traditional metadata-entry forms, and also serves as a well-documented example. Some default forms include:

- "bitstream-metadata" - This is a special form which defines the metadata fields available for every uploaded bitstream (file)
- "traditionalpageone" - A sample form which is used by the first "Describe" step defined in item-submission.xml

- "traditionalpagetwo" - A sample form which is used by the second "Describe" step defined in item-submission.xml
- A number of sample forms for various out-of-the-box Configurable Entities. These forms all have a corresponding <step> defined in item-submission.xml. In conjunction to those <step> definitions, these forms may be used to submit new Entities of specific types. Usually this is done by mapping that Entity-specific submission-process (in item-submission.xml) to a Collection which is used for new submissions of that Entity.

The rest of this section explains how to create your own sets of custom forms.

Describing Custom Metadata Forms

The description of a set of fields through which submitters enter their metadata is called a *form* (in the UI, each "form" is displayed in a separate collapsible section). A form is identified by a unique symbolic *name*. In the XML structure, the *form* is broken down into *rows of fields*. This allows you to place smaller fields side-by-side in a single, horizontal row, or alternatively decide to display one field per row.

The Structure of *submission-forms.xml*

The name & structure of this file changed slightly in DSpace 7

As of DSpace 7, the following structural changes were made to this configuration:

- input-forms.xml (v6) was renamed to submission-forms.xml
- <form-map> top-level element was removed. All Collection mappings are now in item-submission.xml
- <page> element under <form> was removed. As described below, <form> element now represent a single section of the submission process.
- <row> element under <form> was added. As described below, multiple fields can now be displayed in one horizontal row.
- A new form named "bitstream-metadata" was introduced to allow you to configure which metadata is requested for a bitstream during submission.

The XML configuration file has a single top-level element, *input-forms*, which contains two elements in a specific order. The outline is as follows:

```
<input-forms>

    <--  Form Set Definitions -->
    <form-definitions>
        <form name="traditionalpageone">
            ...
        </form>
        ...
    </form-definitions>

    <--  Name/Value Pairs used within Multiple Choice Widgets -->
    <form-value-pairs>
        <value-pairs value-pairs-name="common_iso_languages" dc-term="language_iso">
            ...
        </value-pairs>
        ...
    </form-value-pairs>
</input-forms>
```

Using a form in a submission process for a Collection

Keep in mind, the "submission-forms.xml" only defines *forms* and *value-pairs* (used for specific fields like selectboxes). To enable a form requires also updating the "item-submission.xml" configuration to use that form (see also above):

1. In "item-submission.xml", a <step-definition> of type "submission-form" must be created, with an "id" matching the *name* of the *form* (see above for more details on step-definition)
2. In "item-submission.xml", a <submission-process> must be created/updated to use that newly defined "step".
3. Finally, also in "item-submission.xml", a Collection must be setup to use that submission process in the <submission-map> section.

So, if you modify submission-forms.xml, you may need to double check your changes will be used in your item-submission.xml.

Adding a Form

You can add a new form by creating a new *form* element within the *form-definitions* element. It has one attribute, *name*, which as described above must match the "id" of a <step-definition> in "item-submission.xml".

Forms and Pages

The content of the *form* is a sequence of *row* elements. Each of these corresponds to a single, horizontal row, containing metadata input fields. The rows are presented in sequence, with the first row displayed at the top of the form. A form is displayed as a section (or step) within the submission process.

A *form* may contain any number of rows. A row generally only contains one or two input fields (including more than one input field may require the "style" setting, see below). 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.
- **language**: If set to *true* a drop down menu will be shown, containing languages. The selected language will be used as language tag of the metadata field. A compulsory argument *value-pairs-name* must be given containing the name of the value pair that contains all the languages: e. g. `<language value-pairs-name="common_iso_languages">true</language>`.
- **repeatable**: Value is *true* when multiple values of this field are allowed, *false* otherwise. When you mark a field repeatable, the UI will add an "Add more" control to the field, allowing the user to 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. Repeatable fields also support reordering of values.
- **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 (i.e. a normal input textbox)
  - **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'. *(By default, this input type is unused. Author fields now use the "onebox" type to support different types of names.)*
  - **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.
    - Requires that you 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.
    - Similar to the *dropdown* type, *requires* that you include the *value-pairs-name* attribute to specify a menu choice list.
    - Because the "qualdrop_value" dynamically sets the *qualifier* (based on the drop-down menu), the <dc-qualifier> field *MUST* be empty. The <dc-qualifier> element cannot be used with this field type.
  - **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. *(By default, this input type is unused.)*
    - Requires that you include a value for the *value-pairs-name* attribute to specify a list of values from which to choose.
  - **tag** - A free-text field which allows you to add multiple labels/tags as values. An example is the "Subject Keywords" field.
    - Note: A tag field MUST be marked as <repeatable>true</repeatable>.
- **hint** (Required): Content is the text that will appear as a "hint", or instructions, below the input field. Can be left empty, but the tag must be present.
- **required**: When this element is included with any content, it marks the field as a required input. If the user saves the form 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>*
- **vocabulary:** When specified, this field uses a controlled vocabulary defined in `[dspace]/config/controlled-vocabularies/[name].xml`. This setting may be used to provide auto-complete functionality, for example in the "Subject Keywords" field (which uses the "tag" input type). See also the "Configuring Controlled Vocabularies" section below.
- **regex:** When specified, this field will be validated against the Regular Expression, and only successfully validating values will be saved. An example is commented out in the default "Author" field. If the validation fails, the following error message will be shown by default: *"This input is restricted by the current pattern: {{ pattern }}."*. This can be customized, by adding an entry to the internalization files with the key `error.validation.pattern.schema_element_qualifier` and the schema, element and qualifier of the field. For example: *"error.validation.pattern.dc_identifier": "The identifier can only consist of numbers"*. For instructions on how to add custom entries see: Customize UI labels using Internationalization (i18n) files
- **style:** When specified, this provides a CSS style recommendation to the UI for how to style that field. This is primarily used when displaying multiple fields per row, so that you can tell the UI how many columns each field should use in that row. Keep in mind, these styles should follow the Bootstrap Grid System, where the number of columns adds up to 12. An example can be see in the default "Date of Issue" and "Publisher" fields, which are configured to use 4 (col-sm-4) and 8 (col-sm-8) columns respectively.
- **visibility:** the submission scope for which the field should be visible. Values allowed are *submission* or *workflow.* When one of the two options is given the filed will be visible only for the scope provided and it will be hidden otherwise.
- **readonly:** this option can be used only together with the *visibility* element, and it means the field should be a read-only input instead of being hidden out of the scope provided by the *visibility* element. The value allowed is *readonly*, e.g.: *<readonly>readonly</readonly>*

Visibility configuration examples

A field configured to be visible only with *submission* scope, while is hidden with *workflow* scope

```
<field>
    <dc-schema>dc</dc-schema>
    <dc-element>title</dc-element>
    <dc-qualifier>alternative</dc-qualifier>
    <repeatable>true</repeatable>
    <label>Other Titles</label>
    <input-type>onebox</input-type>
    <hint>If the item has any alternative titles, please enter them here.</hint>
    <required></required>
    <visibility>submission</visibility>
</field>
```

A field configured to be visible only with *workflow* scope, while is read-only with *submission* scope

```
<field>
    <dc-schema>dc</dc-schema>
    <dc-element>title</dc-element>
    <dc-qualifier>alternative</dc-qualifier>
    <repeatable>true</repeatable>
    <label>Other Titles</label>
    <input-type>onebox</input-type>
    <hint>If the item has any alternative titles, please enter them here.</hint>
    <required></required>
    <readonly>readonly</readonly>
    <visibility>workflow</visibility>
</field>
```

Item type Based Metadata Collection

Available in 7.3 and later

A field can be made visible depending on the value of *dc.type*. A new field element, <type-bind>, has been introduced to facilitate this. The <type-bind> takes a comma separated list of publication types. If the field is missing or empty, it will always be visible. In this example the field will only be visible if a value of "thesis" or "ebook" has been entered into *dc.type* on an earlier page:

```
<field>
    <dc-schema>dc</dc-schema>
    <dc-element>identifier</dc-element>
    <dc-qualifier>isbn</dc-qualifier>
    <label>ISBN</label>
    <type-bind>thesis,ebook</type-bind>
</field>
```

A field may be configured multiple times in the submission configuration with different values in type-bind. This is useful if a field is required for one type but not another, or should display a different label and hint message depending on the publication type:

```
<field>
    <dc-schema>dc</dc-schema>
    <dc-element>identifier</dc-element>
    <dc-qualifier>isbn</dc-qualifier>
    <label>ISBN</label>
    <type-bind>book,ebook</type-bind>
        <required>You must enter an ISBN for this book</required>
</field>

<field>
    <dc-schema>dc</dc-schema>
    <dc-element>identifier</dc-element>
    <dc-qualifier>isbn</dc-qualifier>
    <label>ISBN of Parent Publication</label>
    <type-bind>thesis,book chapter,letter</type-bind>
        <hint>Enter the ISBN of the book in which this was published</hint>
</field>
```

If a field is *required* but is bound to a type that does not match the submitted publication, the *required* value will be ignored.

Note: When the submitter changes the Type field, other fields (usually just below it) dynamically appear. There's a brief demo of this feature in the 2022-07-13 - DSpace 7 Q&A webinar at time 19:05. The submission process is one page, but it has collapsible sections, each of which corresponds to one of the old "pages".

Composition of a Relation Field

The row tag can contains the `relation-field` element instead than `field` when we want to manage a relationship within the submission. The structure of this element is close to the one of the "normal" field with the following exception

- the element is named `relation-field`
- the following child element are defined
    - relationship-type: contains the name of the relations that you want to manage, from the side of the linked item (i.e. in the publication submission you have `isAuthorOfPublication`)

- search-configuration: the name of a search configuration in the discovery.xml file suitable to search and filter the items of the type expected to be linked by the relation (i.e. `person`)
- externalsources: the list of external sources that should be offered as an import source for the linked items (i.e. orcid to allow import of person). For DSpace 10 this import action can only be used by admin users unless the *submitter.allow.external.entities* is set to "true" (see Configuration Reference#SettingsfortheSubmissionProcess for further information)
- repeatable, label, hint, required: with the same meaning and purpose of the corresponding element in the `field` element (see Composition of a Field)
- linked-metadata-field: this node is used to define a fallback metadata to be used when no items exist to establish the relation, and a plain value (i.e. author name) should be recorded instead than a proper relation to a person item. It contains the following child elements: dc-schema, dc-element, dc-qualifier and input-type (see the description provided in the paragraph related to the Field element)

example of relation-field configuration

```
<relation-field>
        <relationship-type>isAuthorOfPublication</relationship-type>
        <search-configuration>person</search-configuration>
        <repeatable>true</repeatable>
        <label>Author</label>
        <hint>Enter the author's name (Family name, Given names).</hint>
        <linked-metadata-field>
                <dc-schema>dc</dc-schema>
                <dc-element>contributor</dc-element>
                <dc-qualifier>author</dc-qualifier>
                <input-type>onebox</input-type>
        </linked-metadata-field>
        <externalsources>orcid</externalsources>
        <required></required>
</relation-field>
```

Configuring Controlled Vocabularies

DSpace supports controlled vocabularies to confine the set of keywords that users can use while describing items. The need for a limited set of keywords is important since it eliminates the ambiguity of a free description system, consequently simplifying the task of finding specific items of information. The controlled vocabulary allows the user to choose from a defined set of keywords organized in an tree (taxonomy) and then use these keywords to describe items while they are being submitted.

The taxonomies are described in XML following this (very simple) structure:

```
<node id="acmccs98" label="ACMCCS98">
    <isComposedBy>
        <node id="A." label="General Literature">
            <isComposedBy>
                <node id="A.0" label="GENERAL"/>
                <node id="A.1" label="INTRODUCTORY AND SURVEY"/>
                ...
            </isComposedBy>
        </node>
        ...
    </isComposedBy>
</node>
```

You are free to use any application you want to create your controlled vocabularies. A simple text editor should be enough for small projects. Bigger projects will require more complex tools. You may use Protegé to create your taxonomies, save them as OWL and then use a XML Stylesheet (XSLT) to transform your documents to the appropriate format. Future enhancements to this add-on should make it compatible with standard schemas such as OWL or RDF.

New vocabularies should be placed in `[dspace]/config/controlled-vocabularies/` and must be according to the structure described.

Vocabularies need to be associated with the corresponding metadata fields. Edit the file `[dspace]/config/submission-forms.xml` and place a `"vocabulary"` tag under the `"field"` element that you want to control. Set value of the `"vocabulary"` element to the name of the file that contains the vocabulary, leaving out the extension (the add-on will only load files with extension "*.xml"). For example:

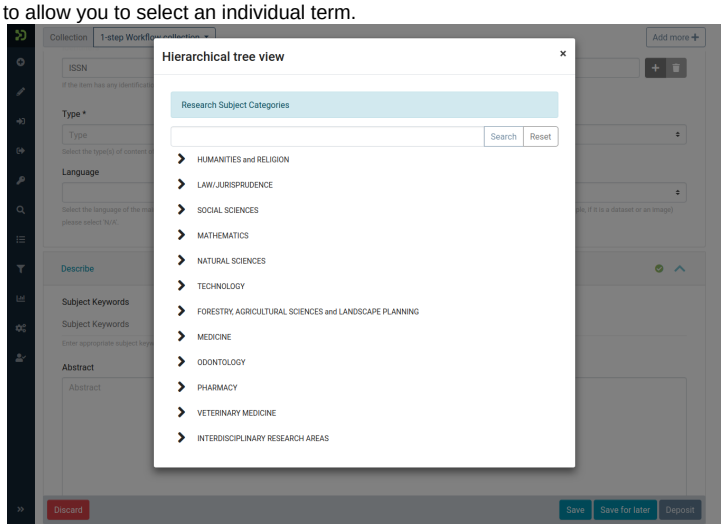
```
<field>
    <dc-schema>dc</dc-schema>
    <dc-element>subject</dc-element>
    <dc-qualifier></dc-qualifier>
    <repeatable>true</repeatable>
    <label>Subject Keywords</label>
```

```
    <input-type>onebox</input-type>
    <hint>Enter appropriate subject keywords or phrases below.</hint>
    <required></required>
    <vocabulary>srsc</vocabulary>
</field>
```

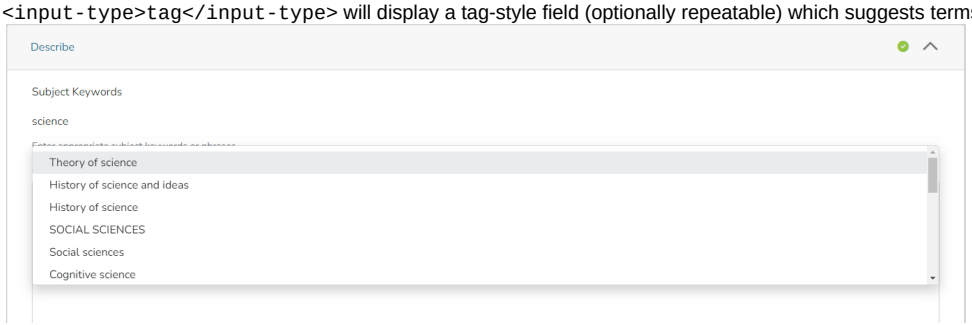
The vocabulary element has an optional boolean attribute closed that can be used to force input only with the Javascript of controlled-vocabulary add-on. The default behaviour (i.e. without this attribute) is as set closed="false". This allow the user also to enter the value in free way.

Controlled vocabularies have two main display types in the submission form:

1. `<input-type>onebox</input-type>` will display a onebox style field (optionally repeatable) which pops up the entire hierarchical vocabulary to allow you to select an individual term.



2. `<input-type>tag</input-type>` will display a tag-style field (optionally repeatable) which suggests terms within the vocabulary as you type.



The following vocabularies are currently available by default:

- **nsi** - *nsi.xml* - The Norwegian Science Index
- **srsc** - *srsc.xml* - Swedish Research Subject Categories

Adding Value-Pairs

Finally, your custom form description needs to define the "value pairs" for any fields with input types that refer to them. Do this by adding a *value-pairs* element to the contents of *form-value-pairs*. It has the following required attributes:

- **value-pairs-name** – Name by which an *input-type* refers to this list.
- **dc-term** – Dublin Core field for which this choice list is selecting a value.

Each *value-pairs* element contains a sequence of *pair* sub-elements, each of which in turn contains two elements:

- **displayed-value** – Name shown (on the web page) for the menu entry.
- **stored-value** – Value stored in the DC element when this entry is chosen. Unlike the HTML *select* tag, there is no way to indicate one of the entries should be the default, so the first entry is always the default choice.

Example

Here is a menu of types of common identifiers:

```
<value-pairs value-pairs-name="common_identifiers" dc-term="identifier">
    <pair>
        <displayed-value>Gov't Doc #</displayed-value>
        <stored-value>govdoc</stored-value>
    </pair>
    <pair>
        <displayed-value>URI</displayed-value>
        <stored-value>uri</stored-value>
    </pair>
    <pair>
        <displayed-value>ISBN</displayed-value>
        <stored-value>isbn</stored-value>
    </pair>
</value-pairs>
```

It generates the following HTML, which results in the menu widget below. (Note that there is no way to indicate a default choice in the custom input XML, so it cannot generate the HTML *SELECTED* attribute to mark one of the options as a pre-selected default.)

```
<select name="identifier_qualifier_0">
    <option VALUE="govdoc">Gov't Doc #</option>
    <option VALUE="uri">URI</option>
    <option VALUE="isbn">ISBN</option>
</select>
```

Deploying Your Custom Forms

The DSpace web application only reads your custom form definitions when it starts up, so it is important to remember:

- *You must always restart Tomcat* (or whatever servlet container you are using) for changes made to the *submission-forms.xml* and/or *item-submission.xml* to 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, may result in errors in the DSpace REST API & UI. The exception message (at the top of the stack trace in the *dspace.log* file) usually has a concise and helpful explanation of what went wrong. Don't forget to stop and restart the servlet container before testing your fix to a bug.

Configuring the File Upload step

Basic Settings

The *Upload* step in the DSpace submission process has a few configuration options which can be set with your *[dspace]/config/local.cfg* configuration file. They are as follows:

- *spring.servlet.multipart.max-file-size (default=512MB)* - Spring Boot's maximum allowable file upload size. For DSpace, we default it to 512MB (in application.properties). But, you may wish to override the default value in your local.cfg. Example values include "512MB", "1GB", or even "-1" (to allow unlimited). See Spring's documentation on this setting: https://spring.io/guides/gs/uploading-files/#_tuning_file_upload_limits
    - NOTE: Increasing this value significantly does NOT guarantee that DSpace will be able to successfully upload files of a very large size via the web. Large uploads depend on many other factors including bandwidth, web server settings, internet connection speed, etc. Therefore, for very large files, you may need to consider importing via command-line tools or similar.
- *spring.servlet.multipart.max-request-size (default=512MB)* - Spring Boot's maximum allowable upload *request* size (i.e. the maximum total upload size for all files in a multi-file upload). For DSpace, we default it to 512MB (in application.properties). But, you may wish to override the default value in your local.cfg. Example values include "512MB", "1GB", or even "-1" (to allow unlimited). See Spring's documentation on this setting: https://spring.io/guides/gs/uploading-files/#_tuning_file_upload_limits
    - NOTE: Increasing this value significantly does NOT guarantee that DSpace will be able to successfully upload files of a very large size via the web. Large uploads depend on many other factors including bandwidth, web server settings, internet connection speed, etc. Therefore, for very large files, you may need to consider importing via command-line tools or similar.
- *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.

Modifying metadata form presented for Bitstreams

After uploading a file (bitstream) in the Submission UI, you can optionally edit that bitstream's metadata. The form displayed on that edit screen is built by the "bitstream-metadata" form defined in submission-forms.xml. You can modify that form to change the fields captured for a Bitstream. *However, the "dc.title" field is REQUIRED in order to store the name of the file.*

```
<form-definitions>
    <!-- Form used for entering in Bitstream/File metadata after uploading a file -->
    <form name="bitstream-metadata">
```

```
        ...
    </form>
</form-definitions>
```

Modifying access conditions (embargo, etc.) presented for Bitstreams

After uploading a file (bitstream) in the Submission UI, you can optionally edit that bitstream's access conditions. This allows you to embargo a bitstream, lease it, or limit it to Administrators only.

These access conditions are defined in a new Spring Bean configuration file `[dspace]/config/spring/api/access-conditions.xml`

- The "uploadConfigurationService" bean maps an existing "UploadConfiguration" bean (default is "uploadConfigurationDefault") to a specific step /section name used in item-submission.xml.

```
<!-- This default configuration says the <step-definition id="upload"> defined in item-submission.xml
uses "uploadConfigurationDefault" -->
<bean id="uploadConfigurationService" class="org.dspace.submit.model.UploadConfigurationService">
    <property name="map">
        <map>
            <entry key="upload" value-ref="uploadConfigurationDefault" />
        </map>
    </property>
</bean>
```

- One or more UploadConfiguration beans may exist, providing different options for different upload sections. An "UploadConfiguration" consists of several properties:
    - **name** (Required): The unique name of this upload configuration
    - **configurationService** (Required through 7.3): reference to the DSpace ConfigurationService (should always be "org.dspace.services.ConfigurationService"). Starting in 7.4 this is no longer required and should not be set.
    - **metadata** (Required): The metadata "form" to use for this upload configuration. The value specified here MUST correspond to a <form> defined in your submission-forms.xml. In the below example, the "bitstream-metadata" form is used by the "uploadConfigurationDefault" bean...meaning that form will be used to capture metadata about the uploaded bitstream.
    - **options** (Required, but can be empty): list of all "AccessConditionOption" beans to enable. This list will be shown to the user to let them select which access restrictions to place on each bitstream. NOTE: *To disable the ability to select bitstream access restrictions, comment out all <ref> tags to create an empty list of options.*
    - **maxSize:** Optionally, you can specify a maximum size of file accepted by this UploadConfiguration. If unspecified, default is to use the maximum file upload limits specified in Spring Boot (see "Basic Settings" above)
    - **required**: Optionally, you can specify if a file upload is required for this UploadConfiguration. If true, upload is required and users cannot complete a submission without uploading at least one file. If false, no upload is required to complete the submission. If unspecified, default is to use "webui.submit.upload.required" configuration in dspace.cfg/local.cfg, which defaults to "true" (file upload required).

```
<bean id="uploadConfigurationDefault" class="org.dspace.submit.model.UploadConfiguration">
    <property name="name" value="upload"></property>
        <property name="configurationService" ref="org.dspace.services.ConfigurationService"/>
        <property name="metadata" value="bitstream-metadata" />
    <property name="options">
        <!-- This is the list of access options which will be displayed on the "bitstream-
metadata" form -->
        <!-- If no <ref> tags appear in this list, then access restrictions will not be allowed on
bitstreams -->
        <list>
            <ref bean="openAccess"/>
            <ref bean="lease"/>
            <ref bean="embargoed" />
            <ref bean="administrator"/>
        </list>
    </property>
</bean>
```

- Any number of "AccessConditionOption" beans may be added for applying different types of access permissions to uploaded files (based on which one the user selects). These beans are easy to add/update, and just require the following
    - **id** (Required): Each defined bean MUST have a unique "id" and have "class=org.dspace.submit.model.AccessConditionOption".
    - **groupName**: Optionally, define a specific DSpace Group which this Access Condition relates to. This group will be saved to the ResourcePolicy when this access condition is applied.
    - **name**: Give a unique name for this Access Condition. This name is stored in the ResourcePolicy "name" when this access condition is applied.
    - **hasStartDate**: If the access condition is time-based, you can decide whether a start date is required. (true = required start date, false = disabled/not required). This start date will be saved to the ResourcePolicy when this access condition is applied.

- **startDateLimit**: If the access condition is time-based, you can optionally set an start date limit (e.g. +36MONTHS). This field is used to set an upper limit to the start date based on the current date. In other words, a value of "+36MONTHS" means that users cannot set a start date which is more than 3 years from today. This setting's value uses Solr's Date Math Syntax, and is always based on today (NOW).
- **hasEndDate**: If the access condition is time-based, you can enable/disable whether an end date is required. (true = required end date, false = disabled/not required). This end date will be saved to the ResourcePolicy when this access condition is applied.
- **endDateLimit**: If the access condition is time-based, you can optionally set an end date limit (e.g. +6MONTHS). This field is used to set an upper limit to the start date based on the current date. In other words, a value of "+6MONTHS" means that users cannot set an end date which is more than 6 months from today. This setting's value use Solr's Date Math Syntax, and is always based on today (NOW).

```
<!-- Example access option named "embargo", which lets users specify a future date
    (not more than 3 years from now) when this file will be available to Anonymous users -->
<bean id="embargoed" class="org.dspace.submit.model.AccessConditionOption">
        <property name="groupName" value="Anonymous"/>
        <property name="name" value="embargo"/>
        <property name="hasStartDate" value="true"/>
        <property name="startDateLimit" value="+36MONTHS"/>
        <property name="hasEndDate" value="false"/>
</bean>
```

- NOTE: It's possible to **test** the Date math syntax via command-line to see what the value would look like starting from today:

```
./dspace dsrun org.dspace.util.DateMathParser +999YEARS
Applied +999YEARS to implicit current time:  Wed Jun 25 19:42:48 UTC 3023
```

- By default, DSpace comes with these out-of-the-box Access Conditions (which you can customize/change based on local requirements)
  - "administrator" - access restricts the bitstream to the Administrator group immediately (after submission completes)
  - "openAccess" - makes the bitstream immediately accessible to Anonymous group (after submission completes)
  - "embargoed" - embargoes the bitstream for a period of time (maximum of 3 years, as defined in startDateLimit default setting), after which it becomes anonymously accessible. See also Embargo for discussion of how embargoes work in DSpace.
  - "lease" - makes the bitstream anonymously accessible immediately (after submission completes), but that access *expires* after a period of time (maximum of 6 months, as defined in endDateLimit default setting). After that date it is no longer accessible (except to Administrators)

Define upload of files mandatory or optional per collection

You will need to make the following changes to the *access-conditions.xml* file

1. Create a new <bean> below “uploadConfigurationDefault” with the following settings, adding only the line `<property name="required" value="false">`

```
<!-- Define a new bean that sets "required=false" for the UploadConfiguration -->
<bean id="uploadConfigurationWithoutMandatory"
      class="org.dspace.submit.model.UploadConfiguration">
    <property name="name" value="upload" />
    <property name="metadata" value="bitstream-metadata" />
    <property name="required" value="false" />
    <property name="options">
        <list>
            <ref bean="openAccess" />
            <ref bean="lease" />
            <ref bean="embargoed" />
            <ref bean="administrator" />
        </list>
     </property>
</bean>
```

2. Create a new <bean> below "uploadConfigurationService" with the following settings:

```
<!-- Define an UploadConfigurationService which makes Bitstream uploading optional -->
<bean id="uploadConfigurationServiceWithoutMandatory"
    class="org.dspace.submit.model.UploadConfigurationService">
    <property name="map">
        <map>
           <entry key="uploadOptional" value-ref="uploadConfigurationWithoutMandatory" />
        </map>
    </property>
</bean>
```

Now, in the *item-submission.xml* file, you need to make the following changes:

1. Create a new <step-definition> below “upload” with the following settings:

```
<!-- Create an uploadOptional step which will use the
     new "uploadConfigurationServiceWithoutMandatory"
     that you defined in your access-conditions.xml -->
<step-definition id="uploadOptional">
    <heading>submit.progressbar.upload</heading>
    <processing-class>org.dspace.app.rest.submit.step.UploadStep</processing-class>
    <type>upload</type>
</step-definition>
```

2. Finally, what you need to do is use “uploadOptional” as a <step>, replacing default upload, for example:

```
<!-- This first example is a submission form that *REQUIRES* file upload
     by using the existing "upload" step. -->
<submission-process name="traditional">
    <step id="collection" />
    <step id="traditionalpageone" />
    <step id="traditionalpagetwo" />
    <step id="upload" />
    <step id="license" />
</submission-process>

<!-- This second example is a submission form that makes file upload *OPTIONAL*
     by using the newly defined "uploadOptional" step. -->
<submission-process name="traditionalUploadOptional">
    <step id="collection" />
    <step id="traditionalpageone" />
    <step id="traditionalpagetwo" />
    <step id="uploadOptional" />
    <step id="license" />
</submission-process>
```

At this point, any Collection which uses the "traditionalUploadOptional" submission-process will no longer require files to be uploaded. But, any Collection which still uses the "traditional" submission-process will still require files to be uploaded.

Configuring the Item Access Conditions step

Enabling the step

By default, the "Item Access Conditions" step is disabled. To enable it, simply update your `item-submission.xml` to include this tag in your <submission-process>:

```
<submission-process name="traditional">
   ...

   <!-- This step enables embargoes and other access restrictions at the Item level -->
   <step id="itemAccessConditions"/>
</submission-process>
```

After making this update, you will need to restart your backend (REST API) for the changes to take effect.

Modifying access conditions (embargo, etc.) presented for Items

The "Item Access Conditions" step uses a similar access condition configuration as the "Upload" step as described in the Modifying access conditions (embargo, etc.) presented for Bitstreams documentation above.

All available Item access conditions are defined in a new Spring Bean configuration file `[dspace]/config/spring/api/access-conditions.xml`

- One or more "AcccessConditionConfiguration" beans may exist, providing different options for different submission forms (only one should be in use in a form at a time). By default an "accessConditionConfigurationDefault" bean is defined. An "AccessConditionConfiguration" consists of several properties:
    - **name** (Required): The unique name of this configuration. It must match the "id" of the step defined in your `item-submission.xml`

    - **canChangeDiscoverable:** Whether this configuration allows users to change the discoverability of an Item. A "discoverable" item is one that is findable through all search/browse interfaces, provided that you have access to see that Item. A "non-discoverable" item is one that will never be findable through search/browse (except by Administrators)... instead a direct link is necessary to view the Item. See also DSpace Item State Definitions. When "canChangeDiscoverable" is "true", the user can modify discoverability in this submission section. When set to "false", the user cannot modify this setting and all submitted Items will be "discoverable".
    - **options** (Required): list of all "AccessConditionOption" beans to enable for this Item access conditions step. This list will be shown to the user to let them select which access restrictions to place on this Item.
- This step uses the same "AccessConditionOption" beans as the "Upload" step, as described in the Modifying access conditions (embargo, etc.) presented for Bitstreams documentation above. You can choose to enable the same options for both Items and Bitstreams, or provide different options for each.
- By default, DSpace comes with these out-of-the-box Access Conditions (which you can customize/change based on local requirements)
    - "administrator" - access restricts the bitstream to the Administrator group immediately (after submission completes)
    - "openAccess" - makes the bitstream immediately accessible to Anonymous group (after submission completes)
    - "embargoed" - embargoes the bitstream for a period of time (maximum of 3 years, as defined in startDateLimit default setting), after which it becomes anonymously accessible. See also Embargo for discussion of how embargoes work in DSpace.
    - "lease" - makes the bitstream anonymously accessible immediately (after submission completes), but that access *expires* after a period of time (maximum of 6 months, as defined in endDateLimit default setting). After that date it is no longer accessible (except to Administrators)

Examples of selecting Item and Bitstream access conditions

What happens when a User selects different access conditions for an Item (via the "Item Access Conditions" step) and its files (via the "Upload" step)? Generally speaking, both

Generally speaking, both access restrictions will be applied. Here's some examples:

- If a user selects "openAccess" in the "Item Access Conditions" step AND "embargo" in the "Upload" step for one Bitstream
    - Then, the Item's metadata will be publicly visible, but that single Bitstream will be embargoed.
- If a user selects "openAccess" in the "Item Access Conditions" step AND "administrator" in the "Upload" step for one Bitstream
    - Then, the Item's metadata will be publicly visible, but that single Bitstream will only be visible to Administrators
- If a user selects "administrator" in the "Item Access Conditions" step AND nothing in the "Upload" step.
    - Then, the Item's metadata and all Bitstreams will only be accessible to administrators.
- If a user selects "embargo" in the "Item Access Conditions" step AND nothing in the "Upload" step.
    - Then, the Item's metadata and all Bitstreams will be embargoed. Nothing will be visible in the system until the embargo data passes.
- If a user selects "embargo" in the "Item Access Conditions" step AND "openAccess" in the "Upload" step for one Bitstream.
    - Then, the Item's metadata will be embargoed (making it impossible to find the Item unless you are an Administrator). HOWEVER, the bitstream will be publicly accessible immediately (but only via a direct link, as it won't be searchable in the system until the embargo date passes).
- (To test other scenarios, submit a test Item with those permissions applied. Then, edit that Item, visit the "Status" tab, and click "Authorizations" to see what access restrictions were applied to the Item and its bitstreams.)

Configuring the Sherpa Romeo step

Enabling the Sherpa Romeo step

By default, the "Sherpa RoMEO Policy" step is disabled. To enable it, simply update your `item-submission.xml` to include this tag in your <submission-process>:

```
<submission-process name="traditional">
   ...
   <!-- This step shows when appropriate publisher policies retrieved from SHERPA/RoMEO -->
   <step id="sherpaPolicies"/>
</submission-process>
```

you must also obtain your `sherpa.romeo.apikey` registering you client application here https://v2.sherpa.ac.uk/api/ and put them in the `local.cfg`

```
sherpa.romeo.apikey = <YOUR-API-KEY>
```

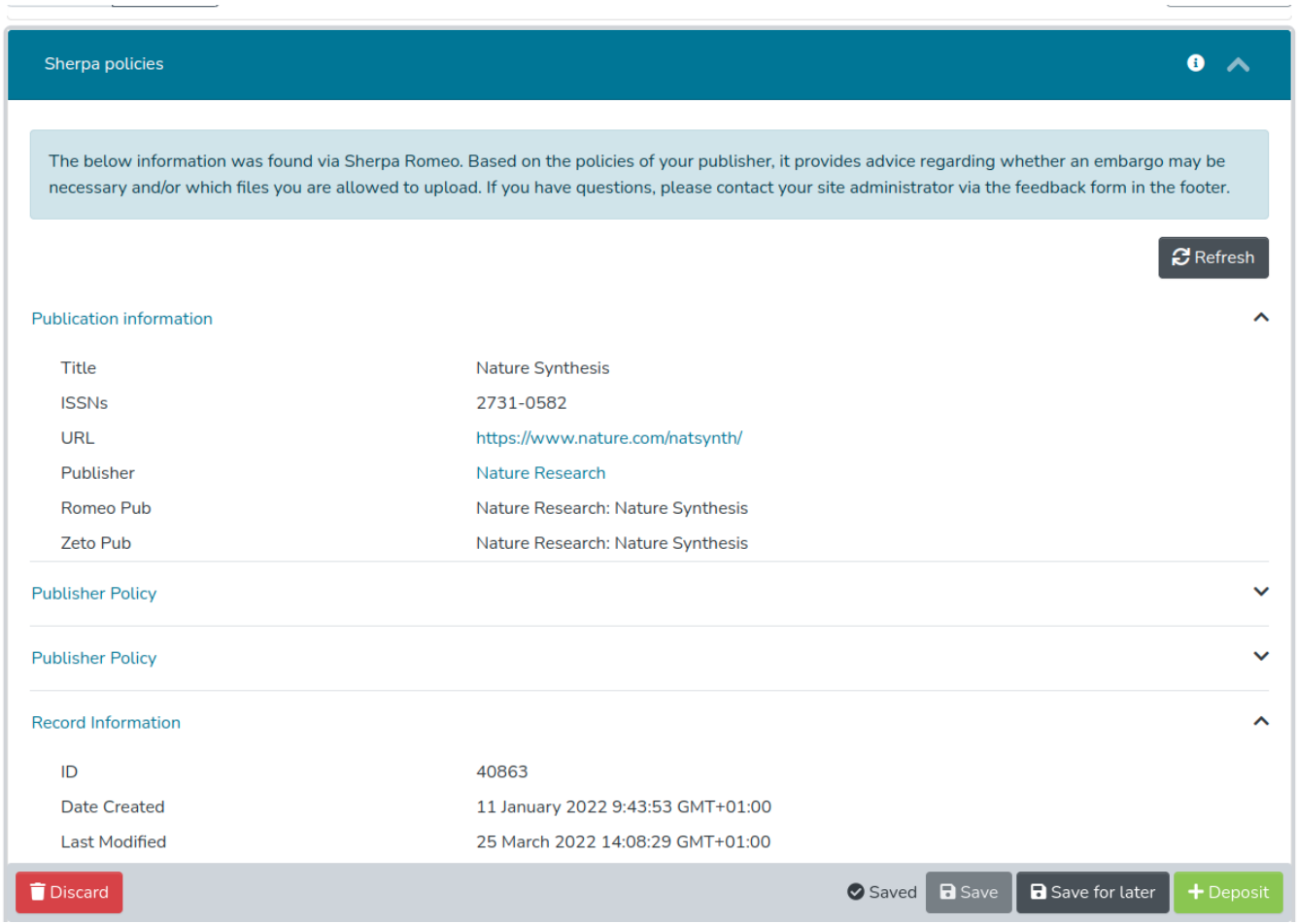
The step needs to extract the ISSN of the Journal where the publication has been submitted/published to query the Sherpa/RoMEO database in order to visualize the publisher policies, this is done by an implementation of the `org.dspace.app.sherpa.submit.ISSNItemExtractor` interface configured in the `org.dspace.app.sherpa.submit.SHERPASubmitConfigurationService`

The configuration is provided by Spring `config/spring/api/sherpa.xml`

```
        <bean class="org.dspace.app.sherpa.submit.SHERPASubmitConfigurationService"
                id="org.dspace.app.sherpa.submit.SHERPASubmitConfigurationService">
                <property name="issnItemExtractors">
                        <list>
                                <bean class="org.dspace.app.sherpa.submit.MetadataValueISSNExtractor">
                                        <property name="metadataList">
                                                <list>
```

```
                                        <value>dc.identifier.issn</value>
                                </list>
                        </property>
                </bean>
                <!-- Uncomment this bean if you have SHERPARoMEOJournalTitle enabled
                <bean class="org.dspace.app.sherpa.submit.MetadataAuthorityISSNExtractor">
                        <property name="metadataList">
                                <list>
                                        <value>dc.title.alternative</value>
                                </list>
                        </property>
                </bean>  -->
        </list>
    </property>
</bean>
```

out-of-box implementations able to extract the ISSN from the metadata value or authority are provided.



Configuring the "Identifiers" step

By default, the "Identifiers" step is disabled. To enable it, update your `item-submission.xml` to include this tag in your <submission-process>:

```
<submission-process name="traditional">

   ...

   <!-- This step shows identifiers already registered for this in-progress item

   <step id="identifiers"/>

   ...

</submission-process>
```

It is recommended to display this step above most others so that the submitter can clearly see any identifiers that will be created while completing their submission.

You must also enable registration of identifiers for workspace and workflow items in `dspace/config/modules/identifiers.cfg` or `local.cfg` (thi s is disabled by default):

```
identifiers.submission.register = true
```

While editing this configuration, pay attention to the filter configuration - logical item filters can be referenced here to apply some conditions as to whether an item qualifies for a DOI or not (eg. based on metadata entered, the type of work, or so on).

Any identifiers registered for the current submission or workflow item will be displayed in a read-only section. If no identifiers are registered, a placeholder “no identifiers” message will be displayed.

If DOI registration is configured for logical item filtering, the DOI will be minted (in a 'pending' state) or deleted as appropriate whenever the in-progress item is saved, depending on whether it passes the filter test.

See `dspace/config/modules/identifiers.cfg`

Creating new Submission Steps Programmatically.

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.*

In most scenarios, this is NOT necessary, as it's much easier to configure a custom Submission Step using DescribeStep or similar.

That being said, at a higher level, creating a new Submission Step requires the following (in this relative order):

1. 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 the UI
2. 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 to the UI*
    - For more information on *<step-definition> tags* within the `item-submission.xml`, see the section above on Defining Steps (*<step>*) within the `item-submission.xml`.
3. For the UI, you will need to..
    - Add a new section type to SectionsType enum matching to the type of step you are creating
    - Create a new Component for this new SectionsType, annotated with "@renderSectionFor()".... see existing section components under src/app/submission/sections for examples.
    - *(Other steps may be necessary... this process has not been fully documented at this time.)*

Basic Duplicate Detection for Submission/Workflow

- Overview
  - Examples of default duplicate display in the DSpace frontend:
- Configuration
  - Configuring Basic Duplicate Detection
  - Configuring Comparison Metadata Fields
    - Field grouping logic
    - Per-field distance configuration
  - Configuring Basic Duplicate Detection in Item Submission
  - Configuring Metadata Value Transformers

Overview

This feature adds basic duplicate detection to DSpace by comparing normalised item metadata fields in Solr with a configurable levenshtein edit distance allowing for fuzzy matching of potential duplicates.

Duplicates can be searched via a submission step, to warn submitters or editors of potential duplicates while they are editing metadata for an in-progress item, and also with a new /duplicates REST item link which will search and retrieve a paged list of potential duplicates for any item.

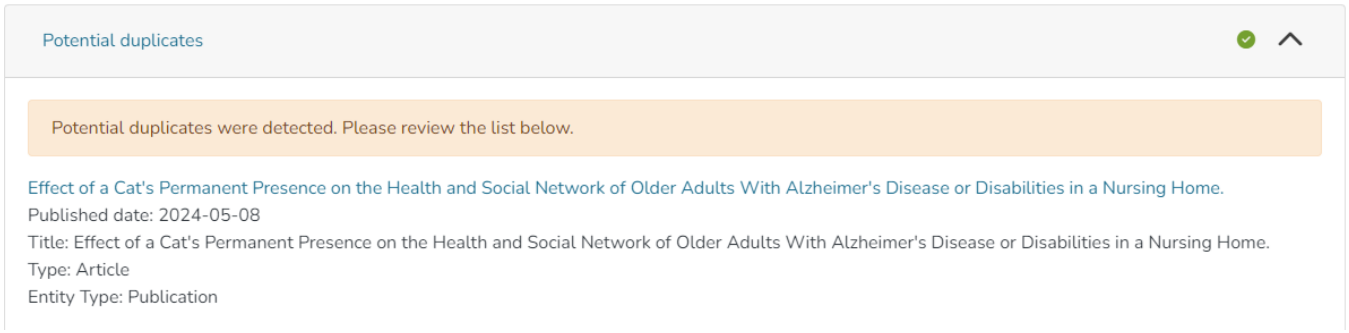
Workflow reviewers / editors will also get a warning for claimed and pooled tasks indicating the total number of potential duplicates.

The feature must be enabled in configuration (see below). It is disabled by default.

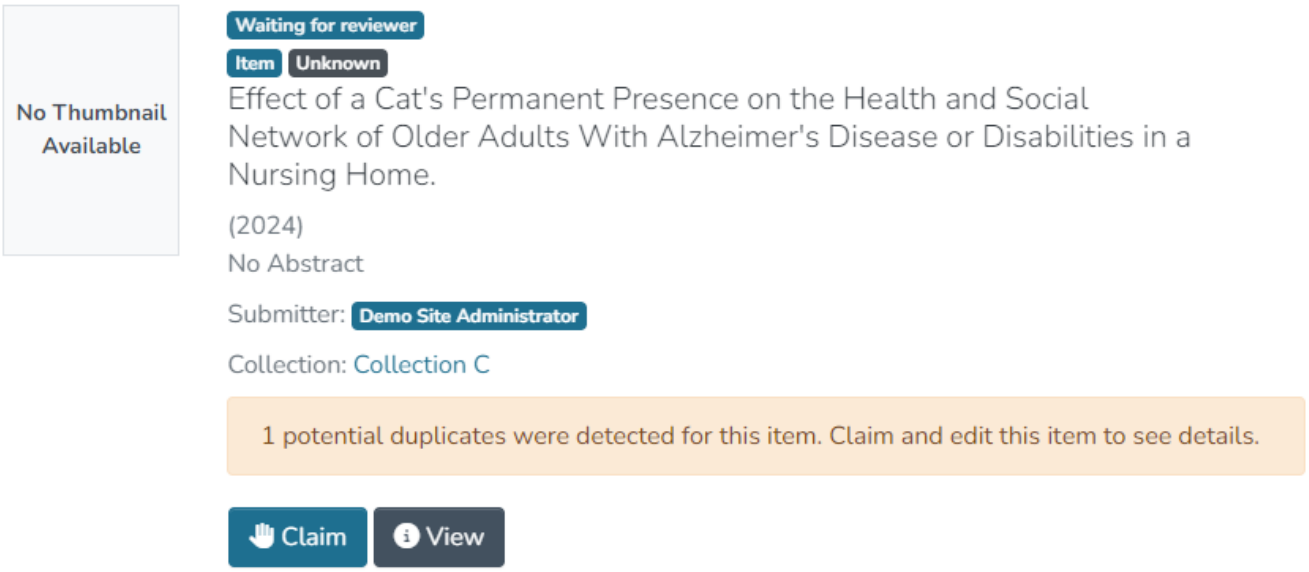
When enabling this feature for the first time, a full discovery reindex must be performed with `${dspace.dir]/bin/dspace index-discovery -b.`

Examples of default duplicate display in the DSpace frontend:

Preview of a potential duplicate in item submission



Warning of 1 potential duplicate for a pooled task.



Configuration

Configuring Basic Duplicate Detection

To enable Basic Duplicate Detection and configure its parameters, edit ${dspace.dir}/config/modules/duplicate-detection.cfg and uncomment or adjust the default properties accordingly.

The default configuration is shown below. The section Configuring Comparison Metadata Fields below covers the comparison field configuration.

Property:	`duplicate.enable`
Example Value:	`duplicate.enable = true`
Informational Note:	This setting enables or disables the entire duplicate detection feature. When changing the value you MUST reindex the site (./dspace index-discovery -b) If the value is not true, any requests to the duplicate detection REST endpoints or section data will be an empty list (the search will not be performed) and item signatures will not be indexed. **Default**: false
Property:	`duplicate.comparison.normalise.lowercase`
Example Value:	`duplicate.comparison.normalise.lowercase` = false
Informational Note:	Specifies whether the metadata used in the fuzzy match for duplicates should be lowercased at index and query time. This is recommended to help keep the edit distance used in fuzzy search predictable and in line with typical user expectations. **Default:** true
Property:	`duplicate.comparison.normalise.whitespace`
Example Value:	`duplicate.comparison.normalise.whitespace` = false
Informational Note:	Specifies whether the metadata used in the fuzzy match for duplicates should have all whitespace stripped at index and query time. This is recommended to help keep the edit distance used in fuzzy search predictable and in line with typical user expectations. **Default:** true
Property:	`duplicate.comparison.distance`
Example Value:	`duplicate.comparison.distance` = 2
Informational Note:	Specifies the maximum edit distance between the two item "signatures" (normalised titles). This value is appended to the Solr term query with the ~ operator. For more information see https://en.wikipedia.org/wiki/Levenshtein_distance A distance of 0 is an exact match (not including any case or whitespace differences as per normalisation rules above) **Default:** 0
Property:	`duplicate.comparison.solr.field.prefix`
Example Value:	`duplicate.comparison.solr.field.prefix` = deduplication
Informational Note:	Specifies the Solr field name prefix to use when indexing the normalised value for duplicate matching. The field name is built using prefix, field name, and suffix - i.e.: **deduplication**_dc_title_*keyword* It is not recommended to change this value. **Default:** deduplication
Property:	`duplicate.comparison.solr.field.suffix`
Example Value:	`duplicate.comparison.solr.field.suffix` = keyword

Informational Note:	Specifies the Solr field name suffix to use when indexing the normalised value for duplicate matching. The field name is built using prefix, field name, and suffix - i.e.: *deduplication*_dc_title_**keyword** It is not recommended to change this value . **Default:** keyword
Property:	`duplicate.preview.metadata.field`
Example Value:	`duplicate.preview.metadata.field` = dc.title `duplicate.preview.metadata.field` = dc.date.issued
Informational Note:	Specifies the item metadata field(s) to include in the duplicate match object, which will be displayed to users. Customise this list of fields to suit your preferences and metadata privacy requirements. **Default:** `duplicate.preview.metadata.field` = dc.title `duplicate.preview.metadata.field` = dc.date.issued `duplicate.preview.metadata.field` = dc.type `duplicate.preview.metadata.field` = dspace.entity.type

To display previews of potential duplicates in item submission, you will need to enable the step as per below

Configuring Comparison Metadata Fields

After changing field configuration, you must fully reindex Solr: `./dspace index-discovery -b`

The metadata fields used for duplicate *comparison* (i.e. the fields used to detect duplicates) are configured separately and support flexible grouping with AND/OR logic.

Each configured field gets its own dedicated Solr index field, which avoids false positives (e.g., preventing an author value from accidentally being matched against a title).

Field grouping logic

Configure fields using `duplicate.comparison.metadata.field`:

- **Multiple fields with the same key** combined with **AND** (all must match)
- **Multiple values within the same field** combined with **OR** (at least one must match)
- **Field groups** combined with **OR** (a match in any group is sufficient)

```
# Example:
# duplicate.comparison.metadata.field   = dc.title
# duplicate.comparison.metadata.field   = dc.contributor.author
# duplicate.comparison.metadata.field.1 = dc.identifier
# duplicate.comparison.metadata.field.1 = dc.subject
#
#  (dc.title AND dc.contributor.author) OR (dc.identifier AND dc.subject)
#
# Default (single field):
duplicate.comparison.metadata.field = dc.title
```

Per-field distance configuration

A field-specific Levenshtein distance can be appended with `:` to override the global `duplicate.comparison.distance` for that field:

```
# Use a distance of 1 for title matching, overriding the global default:
duplicate.comparison.metadata.field = dc.title:1

# Use exact matching (distance 0) for identifiers:
duplicate.comparison.metadata.field.1 = dc.identifier:0
```

Configuring Basic Duplicate Detection in Item Submission

To include a submission section that displays a list of potential duplicates to item submitters and editors,

By default, the "Basic Duplicate Detection" step is disabled. To enable it, simply update your `item-submission.xml` to include this tag in your <submission-process>:

```
<submission-process name="traditional">
   <!-- This step enables preview of potential duplicates for the in-progress item -->

   <step id="duplicates"/>

   ...

</submission-process>
```

After making this update, you will need to restart your backend (REST API) for the changes to take effect.

You will also need to enable the overall Basic Duplicate Detection feature in DSpace configuration as per above.

Configuring Metadata Value Transformers

Before values are used for indexing or querying Solr, they can optionally be transformed on a per-field basis. This is useful when metadata values require normalisation beyond lowercasing/whitespace stripping (e.g. stripping prefixes, normalising identifiers).

Transformers are configured in `core-services.xml` (Spring configuration):

```
<bean class="org.dspace.content.DuplicateDetectionServiceImpl">
    <property name="duplicateComparisonValueTransformers">
        <map>
            <entry key="local.custom">
                <bean class="a.b.c.d.deduplication.CustomValueTransformer"/>
            </entry>
        </map>
    </property>
</bean>
```

The `key` corresponds to the DSpace metadata field name (e.g. `local.custom`). All transformers must implement the `DuplicateComparisonValueTransformer` interface.

A usage example can be found in `dspace-api/src/test/java/org/dspace/content/DuplicateDetectionServiceImplTest.java` (method `testBuildComparisonValueGroups`).

Live Import from external sources

1 General Framework
- 1.1 Introduction
- 1.2 Features
- 1.3 Abstraction of input format
- 1.4 Editing Metadata Mapping
- 1.5 Transformation to DSpace Item
  - 1.5.1 Implementation of an import source for External Sources
  - 1.5.2 Implementation of an import source for files
  - 1.5.3 Mapping raw data to Metadata
  - 1.5.4 Inherited methods
  - 1.5.5 Spring configuration for External Sources
  - 1.5.6 Metadata mapping
  - 1.5.7 Available Metadata Contributor classes
- 1.6 Framework Sources Implementations
  - 1.6.1 PubMed Integration
    - 1.6.1.1 Introduction
    - 1.6.1.2 Publication Lookup URL
    - 1.6.1.3 PubMed Metadata Mapping
    - 1.6.1.4 PubMed specific classes Config
      - 1.6.1.4.1 Metadata mapping classes
      - 1.6.1.4.2 Service classes
  - 1.6.2 ArXiv Integration
    - 1.6.2.1 ArXiv Metadata Mapping
  - 1.6.3 OpenAlex Integration
    - 1.6.3.1 Metadata Mapping
    - 1.6.3.2 Supported Entities
    - 1.6.3.3 Required Configuration
    - 1.6.3.4 Backend Service
    - 1.6.3.5 How to Use It: Publication Import Example

General Framework

Introduction

This framework is used by both the REST API and User Interface to help enhance or enrich submissions. One examples usage is in Importing Items via basic bibliographic formats (Endnote, BibTex, RIS, CSV, etc) and online services (arXiv, PubMed, CrossRef, CiNii, etc)

Features

- lookup publications from remote sources
- Support for multiple implementations

Abstraction of input format

The importer framework does not enforce a specific input format. Each importer implementation defines which input format it expects from a remote source. The import framework uses generics to achieve this. Each importer implementation will have a type set of the record type it receives from the remote source's response. This type set will also be used by the framework to use the correct MetadataFieldMapping for a certain implementation. Read "I mplementation of an import source" below for more information and how to enable the framework.

Editing Metadata Mapping

At a simple level, metadata mapping configurations are all in Spring configs in `[dspace.dir]/config/spring/api/`

In that directory, you'll find a mapping file per import source, e.g. "arxiv-integration.xml", "bibtex-integration.xml", "endnote-integration.xml", "pubmed-integration.xml", etc.

There are two different mapping types.

1. First, mapping from a file-based import (e.g. bibtex, endnote, ris, etc) to a DSpace metadata field.
   a. The list of all of the enabled mappings can be found in a "MetadataFieldConfig" <util:map>, usually at the top of the config file.

```
<util:map id="bibtexMetadataFieldMap" key-type="org.dspace.importer.external.metadatamapping.
MetadataFieldConfig"
             value-type="org.dspace.importer.external.metadatamapping.contributor.
MetadataContributor">
    <description>Defines which metadatum is mapped on which metadatum. Note that while the key
must be unique it
        only matters here for postprocessing of the value. The mapped MetadatumContributor has
full control over
           what metadatafield is generated.
    </description>
    <!-- These entry tags are the enabled mappings. The "value-ref" must map to a <bean> ID. -->
    <entry key-ref="dcTitle" value-ref="bibtexTitleContrib" />
    <entry key-ref="dcAuthors" value-ref="bibtexAuthorsContrib" />
    <entry key-ref="dcJournal" value-ref="bibtexJournalContrib" />
    <entry key-ref="dcIssued" value-ref="bibtexIssuedContrib" />
    <entry key-ref="dcJissn" value-ref="bibtexJissnContrib" />
</util:map>
```

b. Each field in the file is mapped to a DSpace metadata field in a "SimpleMetadataContributor" bean definition. *NOTE: a large number of DSpace defined metadata fields are already configured as MetadataFieldConfig beans in the "dublincore-metadata-mapper.xml" Spring Config in the same directory. These may be reused in other configurations.*

```
<!-- This example bean for BibTex says the "title" key in the BibTex" file should be mapped to the
DSpace metadata field
     defined in the "dcTitle" bean.  This "dcTitle" bean is found in "dublincore-metadata-mapper.
xml" and obviously maps to "dc.title" -->
<bean id="bibtexTitleContrib" class="org.dspace.importer.external.metadatamapping.contributor.
SimpleMetadataContributor">
    <property name="field" ref="dcTitle"/>
    <property name="key" value="title" />
</bean>
```

2. Second, mapping from an external API query import (e.g. arxiv, pubmed, etc) to a DSpace metadata field.
   a. Similar to above, The list of all of the enabled mappings can be found in a "MetadataFieldConfig" <util:map>, usually at the top of the config file.

```
<util:map id="arxivMetadataFieldMap" key-type="org.dspace.importer.external.metadatamapping.
MetadataFieldConfig"
             value-type="org.dspace.importer.external.metadatamapping.contributor.
MetadataContributor">
    <description>Defines which metadatum is mapped on which metadatum. Note that while the key
must be unique it
        only matters here for postprocessing of the value. The mapped MetadatumContributor has
full control over
        what metadatafield is generated.
    </description>
    <!-- These entry tags are the enabled mappings. The "value-ref" must map to a <bean> ID. -->
    <entry key-ref="arxiv.title" value-ref="arxivTitleContrib"/>
    <entry key-ref="arxiv.summary" value-ref="arxivSummaryContrib"/>
    <entry key-ref="arxiv.published" value-ref="arxivPublishedContrib"/>
    <entry key-ref="arxiv.arxiv.doi" value-ref="arxivDoiContrib"/>
    <entry key-ref="arxiv.arxiv.journal_ref" value-ref="arxivJournalContrib"/>
    <entry key-ref="arxiv.category.term" value-ref="arxivCategoryTermContrib"/>
    <entry key-ref="arxiv.author.name" value-ref="arxivAuthorContrib"/>
    <entry key-ref="arxiv.identifier.other" value-ref="arxivOtherContrib"/>
</util:map>
```

   b. Each field in the file is mapped to a DSpace metadata field, usually in a "SimpleXPathMetadatumContributor" bean definition which also uses a "MetadataFieldConfig" bean. *NOTE: a large number of DSpace defined metadata fields are already configured as MetadataFieldConfig beans in the "dublincore-metadata-mapper.xml" Spring Config in the same directory. These may be reused in other configurations.*

```
<!-- This first bean define an XPath query ("ns:title") to map to a field (ID="arxiv.title") in
DSpace -->
<bean id="arxivTitleContrib" class="org.dspace.importer.external.metadatamapping.contributor.
SimpleXpathMetadatumContributor">
    <property name="field" ref="arxiv.title"/>
    <property name="query" value="ns:title"/>
    <property name="prefixToNamespaceMapping" ref="arxivBasePrefixToNamespaceMapping"/>
</bean>
<!-- This second bean then defines which DSpace field to use when "arxiv.title" is references. In
other words, between these two beans,
     the "ns:title" XPath query value is saved to "dc.title". -->
<bean id="arxiv.title" class="org.dspace.importer.external.metadatamapping.MetadataFieldConfig">
    <constructor-arg value="dc.title"/>
</bean>
```

Transformation to DSpace Item

The framework produces an 'ImportRecord' that is completely decoupled from DSpace. It contains a set of metadata DTO's that contain the notion of schema,element and qualifier. The specific implementation is responsible for populating this set. It is then very simple to create a DSpace item from this list.

Implementation of an import source for External Sources

Each external source/API importer implementation must at least implements `org.dspace.importer.external.service.components.QuerySource`, which provides the query method used by the framework to retrieve data from the remote source (e.g. Pubmed, ArXiv, etc). Each external source importer must implements, according to the provider APIs, the declared methods.
An useful abstract for remote sources is `org.dspace.importer.external.service.components.AbstractRemoteMetadataSource`.This class contains functionality to handle request timeout and to retry requests. Using this abstract, the query method must implements java.util.concurrent.Callable.

Implementation of an import source for files

Each file importer implementation must at least implements `org.dspace.importer.external.service.components.FileSource`, which provides the basic methods used by the framework to parse and load data from the file (e.g. CSV, Endnote, etc).

Each importer must implements the method:

```
public List<ImportRecord> getRecords(InputStream inputStream) throws FileSourceException;
```

This method is responsible to transform the input data into an ImportRecord list, which will then managed by the top layer of the framework.

The conversion from raw data to an ImportRecord could be done using the framework too, using the metadata mapping structure (see below).

File sources needs to know which file extensions they have to supports. This is done by the default method `isValidSourceForFile` in `FileSource`, and is controlled by the entries in the list returned by declared method `public List<String> getSupportedExtensions();`

An useful abstract for file source is `org.dspace.importer.external.service.components.AbstractPlainMetadataSource`. It should be used whenever it is possible to model the data in the file as a list of key-value lists (e.g. for CSV files, any row is a key value list).

Mapping raw data to Metadata

The framework core is a mid-layer component which allow the conversion of raw data into metadata (ImportRecord) using xml configurable spring beans.

The core of this approach is `org.dspace.importer.externa.service.AbstractImportMetadataSourceService`. Any service that wants to generate metadata from raw data should go through this abstract.

Our service then should extends AbstractImportMetadataSourceService, and use `transformSourceRecords` to transform raw data into ImportRecords.

The most relevant concept in the framework is `private MetadataFieldMapping<RecordType, MetadataContributor<RecordType>> metadataFieldMapping`. This is the place where the framework take the mapping between row data and the associated metadatum. This map must be injected in the service, and will be used by `transformSourceRecords` to convert the data.

`RecordType` is a generic type, which rapresent a single entry of the list of data, and will be mapped to a single ImportRecord. Any metadatum will be mapped to a specific field in the RecordType using a Contributor as described in Metadata mapping.

Inherited methods

Method getImportSource() should return a unique identifier. Importer implementations should not be called directly, but class *org.dspace.importer.external.service.ImportService* should be called instead. This class contains the same methods as the importer implementations, but with an extra parameter 'url'. This url parameter should contain the same identifier that is returned by the getImportSource() method of the importer implementation you want to use.

The other inherited methods are used to query the remote source.

Spring configuration for External Sources

In order to make the live import providers available, them must be mapped as spring beans into `dspace-api/src/main/resources/spring/spring-dspace-addon-import-services.xml.`

This is an example of a provider which allow to import both files and remote source.

```
<bean id="PubmedImportService"
      class="org.dspace.importer.external.pubmed.service.PubmedImportMetadataSourceServiceImpl" scope="
singleton">
    <property name="metadataFieldMapping" ref="PubmedMetadataFieldMapping"/>
    <property name="supportedExtensions">
        <list>
            <value>xml</value>
        </list>
    </property>
    ...
</bean>
```

Here is defined the service responsible to fetch and transform the data `PubmedImportMetadataSourceServiceImpl`, which is an extension of `AbstractImportMetadataSourceService` as described above.

The field `metadataFieldMapping` is an instance of `Map<MetadataFieldConfig,MetadataContributor>` and contains the effective mapping.

`supportedExtensions` is the file extension this provider supports.

To expose this provider as Live Import provider, we need to construct a bean of type `org.dspace.external.provider.impl.LiveImportDataProvider` in the following way

```
<bean id="pubmedLiveImportDataProvider" class="org.dspace.external.provider.impl.LiveImportDataProvider">
    <property name="metadataSource" ref="PubmedImportService"/>
    <property name="sourceIdentifier" value="pubmed"/>
    <property name="recordIdMetadata" value="dc.identifier.other"/>
</bean>
```

where `metadataSource` is the bean referencing to live import service as described in “Metadata mapping”, `sourceIdentifier` the name of the provider in the live import framework and `recordIdMetadata` the metadatum used as id of the ImportRecord.

Metadata mapping

When using an implementation of `AbstractImportSourceService`, a mapping of remote record fields to DSpace metadata fields can be created.

first create an implementation of class `AbstractMetadataFieldMapping` with the same type set used for the importer implementation.

Then create a spring configuration file in `[dspace.dir]/config/spring/api`.

Each DSpace metadata field that will be used for the mapping must first be configured as a spring bean of class*org.dspace.importer.external.metadatamapping.MetadataFieldConfig*.

```
   <bean id="dc.title" class="org.dspace.importer.external.metadatamapping.MetadataFieldConfig">
       <constructor-arg value="dc.title"/>
   </bean>
```

NOTE: A large number of these MetadataFieldConfig definitions are already provided out-of-the-box in `[dspace.dir]/config/spring/api/dublincore-metadata-mapper.xml` This allows most service-specific Spring configurations to just reuse those existing MetadataFieldConfig definitions

Now this metadata field can be used to create a mapping. To add a mapping for the "dc.title" field declared above, a new spring bean configuration of a class class *org.dspace.importer.external.metadatamapping.contributor.MetadataContributor* needs to be added. This interface contains a type argument. The type needs to match the type used in the implementation of AbstractImportSourceService. The responsibility of each MetadataContributor implementation is to generate a set of metadata from the retrieved document. How it does that is completely opaque to the AbstractImportSourceService but it is assumed that only one entity (i.e. item) is fed to the metadatum contributor.

For example `java SimpleXpathMetadatumContributor implements MetadataContributor<OMElement>` can parse a fragment of xml and generate one or more metadata values.

This bean expects 2 property values:

- field: A reference to the configured spring bean of the DSpace metadata field. e.g. the "dc.title" bean declared above.
- query: The xpath expression used to select the record value returned by the remote source.

```
    <bean id="titleContrib" class="org.dspace.importer.external.metadatamapping.contributor.
SimpleXpathMetadatumContributor">
        <property name="field" ref="dc.title"/>
        <property name="query" value="dc:title"/>
    </bean>
```

Multiple record fields can also be combined into one value. To implement a combined mapping first create a *SimpleXpathMetadatumContributor* as explained above for each part of the field.

```
    <bean id="lastNameContrib" class="org.dspace.importer.external.metadatamapping.contributor.
SimpleXpathMetadatumContributor">
        <property name="field" ref="dc.contributor.author"/>
        <property name="query" value="x:authors/x:author/x:surname"/>
    </bean>
    <bean id="firstNameContrib" class="org.dspace.importer.external.metadatamapping.contributor.
SimpleXpathMetadatumContributor">
        <property name="field" ref="dc.contributor.author"/>
        <property name="query" value="x:authors/x:author/x:given-name"/>
    </bean>
```

Note that namespace prefixes used in the xpath queries are configured in bean "FullprefixMapping" in the same spring file.

```
    <util:map id="FullprefixMapping" key-type="java.lang.String" value-type="java.lang.String">
        <description>Defines the namespace mappin for the SimpleXpathMetadatum contributors</description>
        <entry key="http://purl.org/dc/elements/1.1/" value="dc"/>
        <entry key="http://www.w3.org/2005/Atom" value="x"/>
    </util:map>
```

Then create a new list in the spring configuration containing references to all *SimpleXpathMetadatumContributor* beans that need to be combined.

```
   <util:list id="combinedauthorList" value-type="org.dspace.importer.external.metadatamapping.contributor.
MetadataContributor" list-class="java.util.LinkedList">

        <ref bean="lastNameContrib"/>
        <ref bean="firstNameContrib"/>
    </util:list>
```

Finally create a spring bean configuration of class*org.dspace.importer.external.metadatamapping.contributor.CombinedMetadatumContributor*. This bean expects 3 values:

- field: A reference to the configured spring bean of the DSpace metadata field. e.g. the "dc.title" bean declared above.
- metadatumContributors: A reference to the list containing all the single record field mappings that need to be combined.
- separator: These characters will be added between each record field value when they are combined into one field.

```
    <bean id="authorContrib" class="org.dspace.importer.external.metadatamapping.contributor.
CombinedMetadatumContributor">
        <property name="separator" value=", "/>
        <property name="metadatumContributors" ref="combinedauthorList"/>
        <property name="field" ref="dc.contributor.author"/>
    </bean>
```

Each contributor must also be added to the "MetadataFieldMap" used by the *MetadataFieldMapping* implementation. Each entry of this map maps a metadata field bean to a contributor. For the contributors created above this results in the following configuration:

```
    <util:map id="org.dspace.importer.external.metadatamapping.MetadataFieldConfig"
              value-type="org.dspace.importer.external.metadatamapping.contributor.MetadataContributor">
        <entry key-ref="dc.title" value-ref="titleContrib"/>
        <entry key-ref="dc.contributor.author" value-ref="authorContrib"/>
    </util:map>
```

Note that the single field mappings used for the combined author mapping are not added to this list.

Available Metadata Contributor classes

Class	Description
SimpleXpathMetadatumContributor	Use an XPath expression to map the XPath result to a metadatum
SimpleMetadataContributor	This contributor is used in plain metadata as exposed above. Mapping is easy because it is based on the key used in the DTO.
CombinedMetadatumContributor	Use a LinkedList of MetadataContributor to combine into the value the resulting value for each contributor.

Framework Sources Implementations

PubMed Integration

Introduction

First read the base documentation on external importing (see above). This documentation explains the implementation of the importer framework using PubMed (http://www.ncbi.nlm.nih.gov/pubmed) as an example.

Publication Lookup URL

To be able to do the lookup for our configured import-service, we need to be able to know what URL to use to check for publications. This URL the `publication-lookup.url` setting defined within the `[dspace.dir]/config/modules/publication-lookup.cfg`. You may choose to modify this setting or override it within your local.cfg.

This setting can be modified in one of two ways:

- You can choose to specific a single, specific URL. This will tell the lookup service to only use one location to lookup publication information. Valid URLs are any that are defined as a `baseAddress` for beans within the `[src]/dspace-api/src/main/resources/spring/spring-dspace-addon-import-services.xml` Spring config file.
    - For example, this setting will ONLY use PubMed for lookups: `publication-lookup.url=http://eutils.ncbi.nlm.nih.gov/entrez/eutils/`
- By default, `publication-lookup.url` is set to an asterisk ('*'). This default value will attempt to lookup the publication using ALL configured importServices in the `[src]/dspace-api/src/main/resources/spring/spring-dspace-addon-import-services.xml` Spring config file

PubMed Metadata Mapping

The PubMed metadata mappings are defined in the `[dspace.dir]/config/spring/api/pubmed-integration.xml` Spring configuration file. These metadata mappings can be tweaked as desired. The format of this file is described in the "Metadata mapping" section above

PubMed specific classes Config

These classes are simply implementations based of the base classes defined in importer/external. They add characteristic behavior for services/mapping for the PubMed specific data.

Metadata mapping classes

- "PubmedFieldMapping". An implementation of AbstractMetadataFieldMapping, linking to the bean that serves as the entry point of other metadata mapping
- "PubmedDateMetadatumContributor"/"PubmedLanguageMetadatumContributor". Pubmed specific implementations of the "MetadataContributor" interface

Service classes

- "GeneratePubmedQueryService". Generates the pubmed query which is used to retrieve the records. This is based on a given item.
- "PubmedImportMetadataSourceServiceImpl". Child class of "AbstractImportMetadataSourceService", retrieving the records from pubmed.

ArXiv Integration

ArXiv Metadata Mapping

The ArXiv metadata mappings are defined in the `[dspace.dir]/config/spring/api/arxiv-integration.xml` Spring configuration file. These metadata mappings can be tweaked as desired. The format of this file is described in the "Metadata mapping" section above.

OpenAlex Integration

DSpace now includes live import functionality using OpenAlex, allowing users to fetch and prefill metadata in real time from the OpenAlex API during the submission process.

Metadata Mapping

The metadata mappings for OpenAlex are defined in the Spring configuration file:

```
[dspace.dir]/config/spring/api/openalex-integration.xml
```

OpenAlex has been registered as a `LiveImportDataProvider`, making it available in the import dropdown when submitting supported entities (e.g., Publications, Persons).

To convert OpenAlex API responses into `ImportRecord` objects compatible with DSpace, the integration introduces:

- `OpenAlexPublicationFieldMapping`
- `OpenAlexPersonFieldMapping`

These mapping classes handle the conversion logic for metadata fields used in the submission process.

Supported Entities

The following OpenAlex entities are currently mapped to DSpace entities:

OpenAlex Entity	DSpace Entity	Notes
`works`	Publication	—
`authors`	Person	—
`sources`	Journal	Only if `type = "journal"`
`institutions`	OrgUnit	—
`publishers`	OrgUnit	—
`funders`	OrgUnit	—

Required Configuration

To enable the integration, you must add the following properties in your `dspace.cfg` or configuration files:

```
openalex.url.works = https://api.openalex.org/works
openalex.url.authors = https://api.openalex.org/authors
openalex.url.sources = https://api.openalex.org/sources
openalex.url.institutions = https://api.openalex.org/institutions
openalex.url.publishers = https://api.openalex.org/publishers
openalex.url.funders = https://api.openalex.org/funders
```

These properties define the OpenAlex API endpoints for each supported entity type.

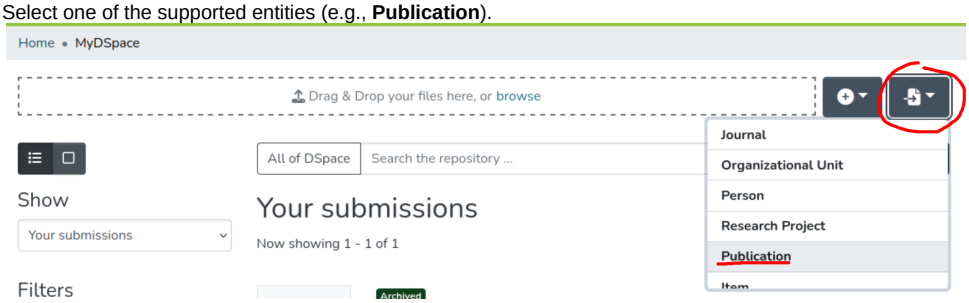
Backend Service

The class `OpenAlexImportMetadataSourceServiceImpl` is responsible for fetching metadata from OpenAlex and returning results used in the import process. It is part of the backend infrastructure that powers both the UI import and any REST-based interactions.

How to Use It: Publication Import Example

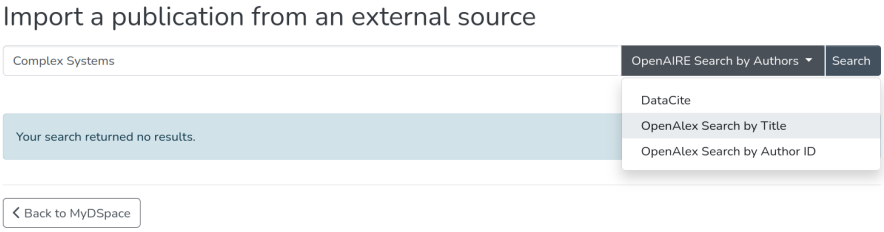
1. **Log in** to DSpace and go to **MyDSpace**.

2. Select one of the supported entities (e.g., **Publication**).

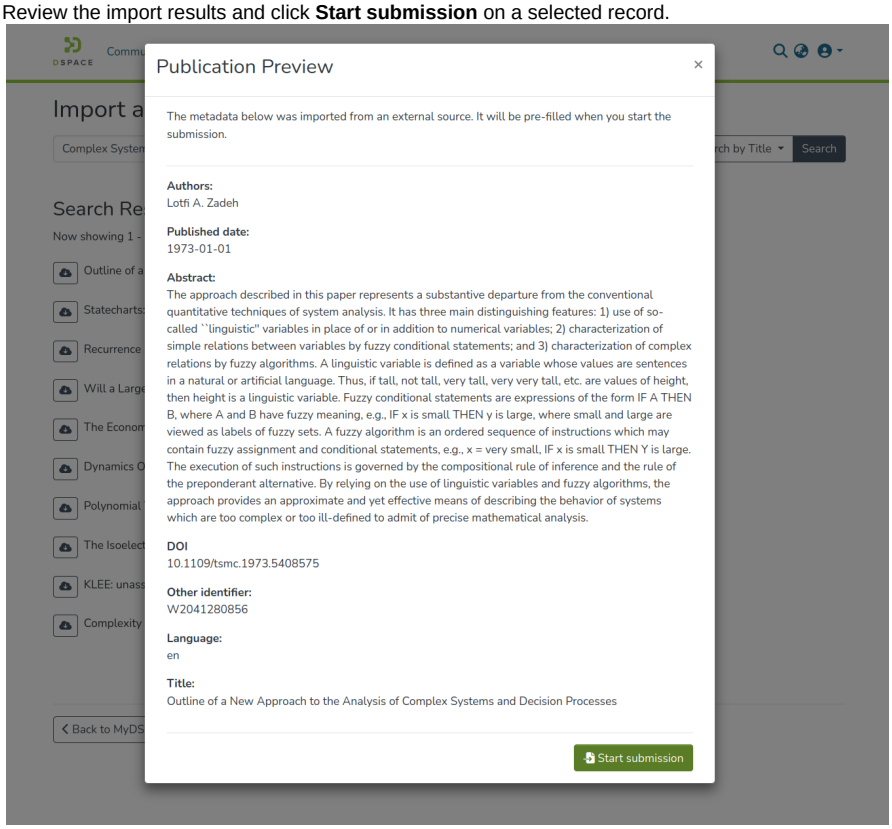


3. Choose **OpenAlex** as the import source from the dropdown.

   a. Search by publication title or OpenAlex author ID (e.g., `A1969205037`).

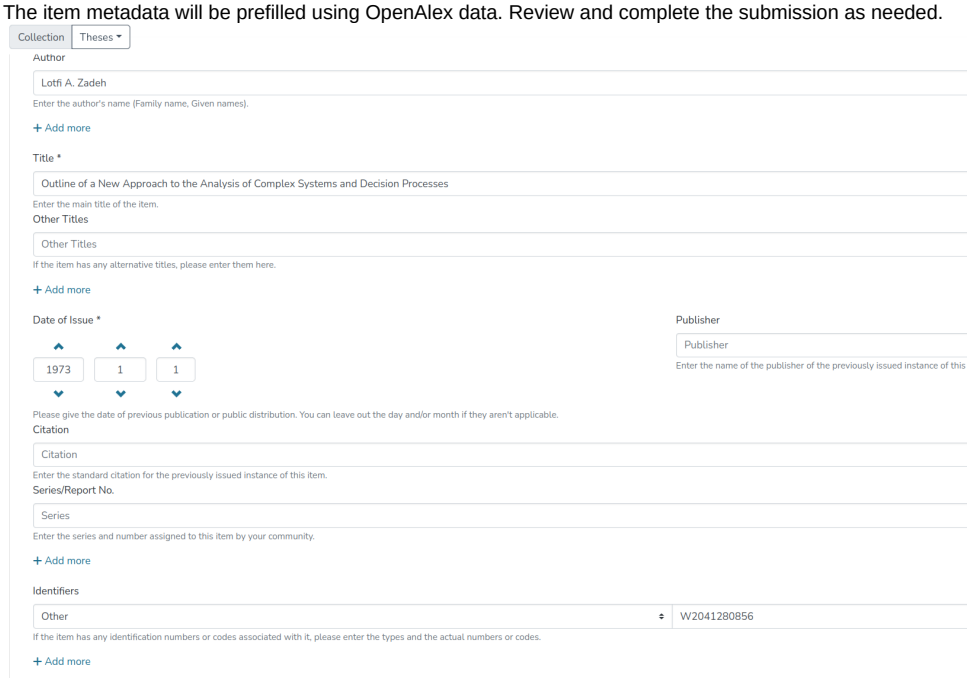


4. Review the import results and click **Start submission** on a selected record.



5. Select the collection where the item should be deposited.

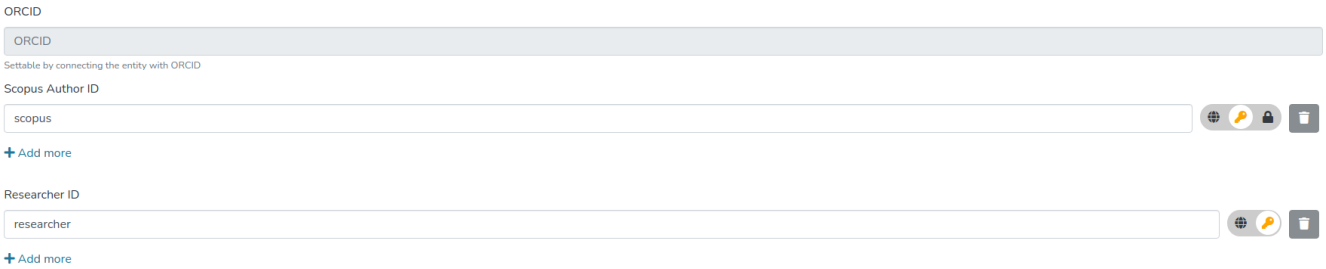
6. The item metadata will be prefilled using OpenAlex data. Review and complete the submission as needed.



Metadata security configuration

Purpose

Allows item submitters/editors to set visibility levels on individual metadata values. When set, a metadata value is only returned via REST/UI if the requesting user meets the required security level.



This mechanism is overridden by **Hide Item Metadata Fields**, so if you set *metadata.hide.person.email = true,* the configuration below will not work for the *person.email* metadata.

Configuration Key Format

```
metadatavalue.visibility[.EntityType][.metadataField].settings = [level1 level2 ...]
```

Lookup Hierarchy (most specific wins)

1. Field-level: metadatavalue.visibility.Person.person.email.settings

2. Entity-level: metadatavalue.visibility.Person.settings

3. Global fallback: metadatavalue.visibility.settings

An empty [] means no security levels are offered (metadata follows standard visibility). A null/missing value falls through to the next level.

Default Security Levels

Defined in *spring-dspace-security-metadata.xml*:

Level	Class	Meaning
0	`MetadataPublicAccess`	Visible to everyone (including Anonymous)
1	`MetadataGroupBasedAccess`	Visible only to members of the "Trusted" group
2	`MetadataAdministratorAndOwnerAccess`	Visible only to Admins or the entity owner

How Values Work

The configured array defines which levels the submitter/editor can choose from, not the visibility itself. For example:

- [0 1 2] – submitter can pick Public, Trusted, or Admin/Owner
- [1 2] – submitter can only pick Trusted or Admin/Owner
- If only one value > 0 is set, the choice is between level 0 and that level

Configuration

Inside *metadata-security.cfg*:

```
metadatavalue.visibility.settings = [0 1 2]
metadatavalue.visibility.Person.settings = [0 1]
metadatavalue.visibility.Person.dc.date.available.settings = [0 1]
```

Custom Security Levels

To add custom levels:

1. Implement the org.dspace.content.service.MetadataSecurityEvaluation interface
2. Register the implementation in *dspace/config/spring/api/spring-dspace-security-metadata.xml* with a new level number:

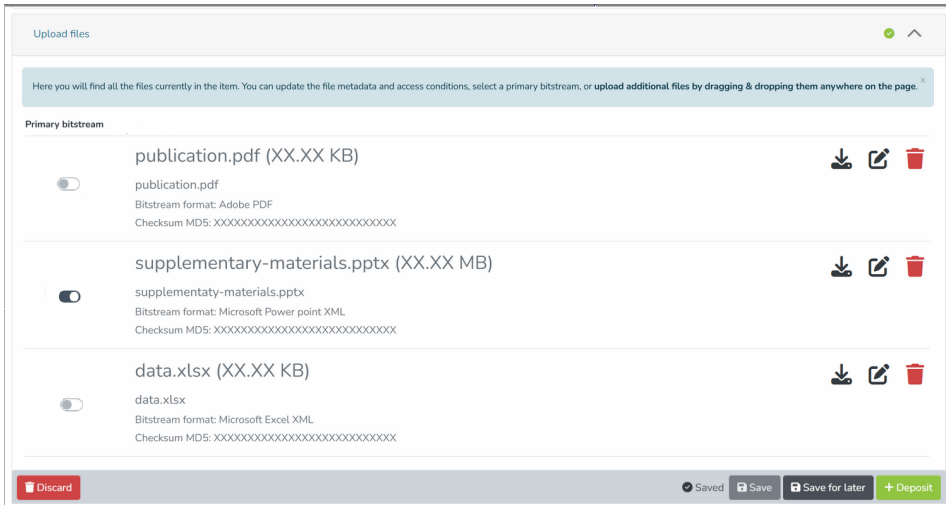
```
<util:map id="securityLevelsMap">
<entry key="0" value-ref="level0Security"/>
<entry key="1" value-ref="level1Security"/>
<entry key="2" value-ref="level2Security"/>
<entry key="3" value-ref="level3Security"/> <!-- your custom level -->
</util:map>
<bean id="level3Security" class="org.dspace.content.YourCustomSecurityAccess"/>
```

3. Add the new level number to the appropriate configuration keys in *metadata-security.cfg*

Set a bitstream as primary

When uploading multiple files during a submission, it is possible to define which is the primary bitstream.

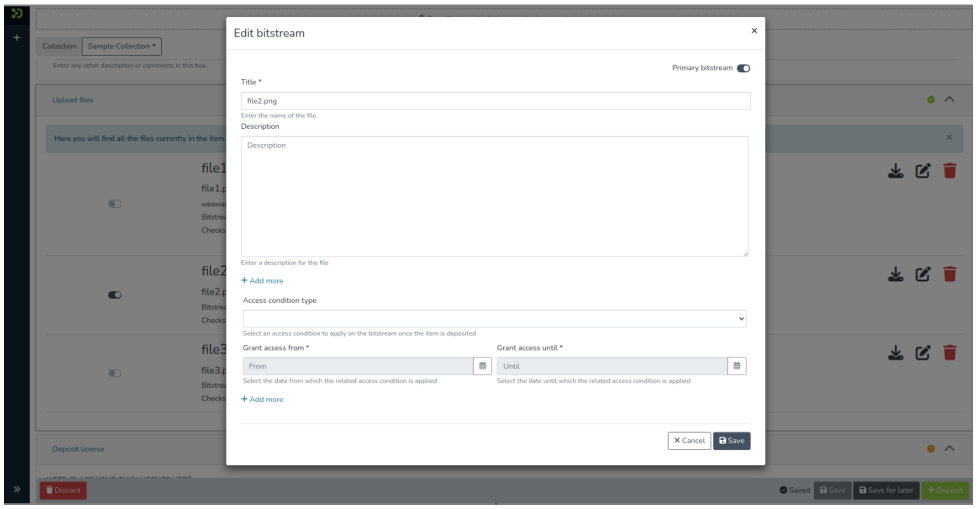
In the list of the uploaded files, a toggle in the **Primary bitstream** column lets choose which of the bitstreams is the primary. The changes are automatically saved.



To choose another uploaded file as primary bitstream, is enough to activate the related toggle. The previous primary bitstream toggle turns off automatically and the changes are saved.

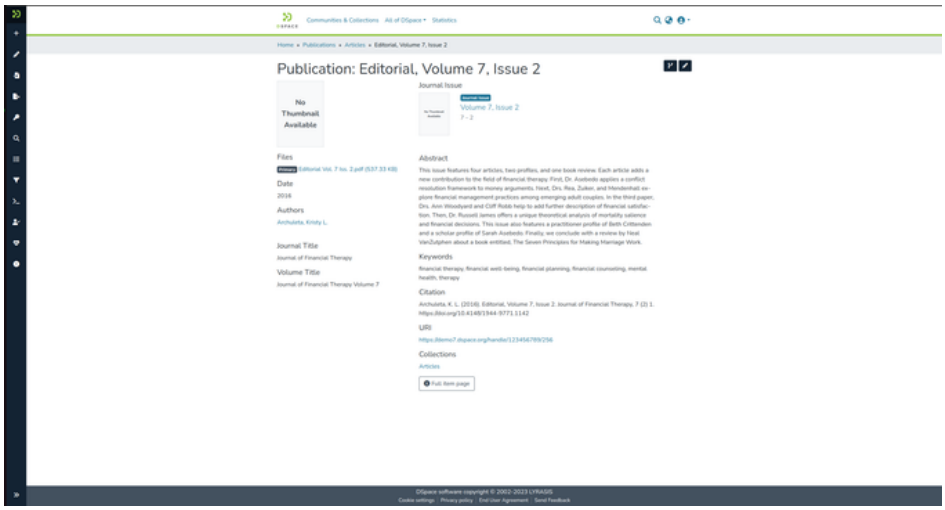
If a bitstream set as a primary gets deleted, a new primary bitstream needs to be defined manually.

The Primary bitstream toggle is also present in the **Edit bitstream** window. Clicking on *Save*, the bitstream appears as primary.

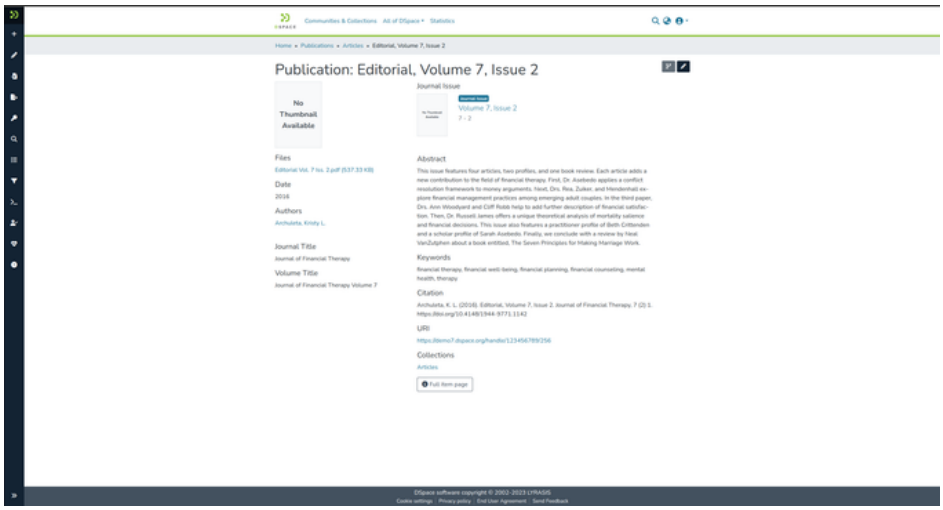


If another bitstream was set as primary before, and a new bitstream is set as primary in its edit window, the toggle on the previous file turns off automatically, and the new bitstream becomes primary. The system then saves the changes.

After depositing the changes, the item page will show the **Primary badge** next to the item's title.



If no bitstream can be defined as primary, the toggles can stay off and the item can be deposited. On the item page, no **Primary badge** will be shown.



Simple HTML Fragment Markup

A few features of the user interface, such as the deposit license text & some metadata fields, can be marked up using a subset of HTML. This HTML subset is defined by Angular, as we use Angular's "[innerHtml]" property to display these HTML-based fields.

Angular automatically sanitizes any HTML passed to "[innerHtml]" in order to avoid XSS attacks. See Angular docs at https://angular.io/guide/security#preventing-cross-site-scripting-xss

At this time, Angular does NOT have a formal reference of elements/attributes which are allowed, but we've compiled a list below of currently known acceptable elements. This list may change in later releases of Angular, but is currently maintained in Angular's "html_sanitizer.ts": https://github.com/angular/angular/blob/main/packages/core/src/sanitization/html_sanitizer.ts

As of the writing of this page, these HTML5 elements may be used:

- h1, h2, h3, h4, h5, h6
- p, div
- a
- img
- audio
- video
- map
- table-based elements (table, td, th, tr, etc)
- list-based elements (ol, ul, li, etc)
- other formatting elements (b, i, u, br, hr, small, font, etc)

Not all DSpace fields support HTML, but the User Interface should make it clear which fields do. When adding HTML to a field, you should not create a complete HTML document (surrounded with "<html>" tags). Just add an HTML fragment.

Supervision Orders

Available in 7.5 or later

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:

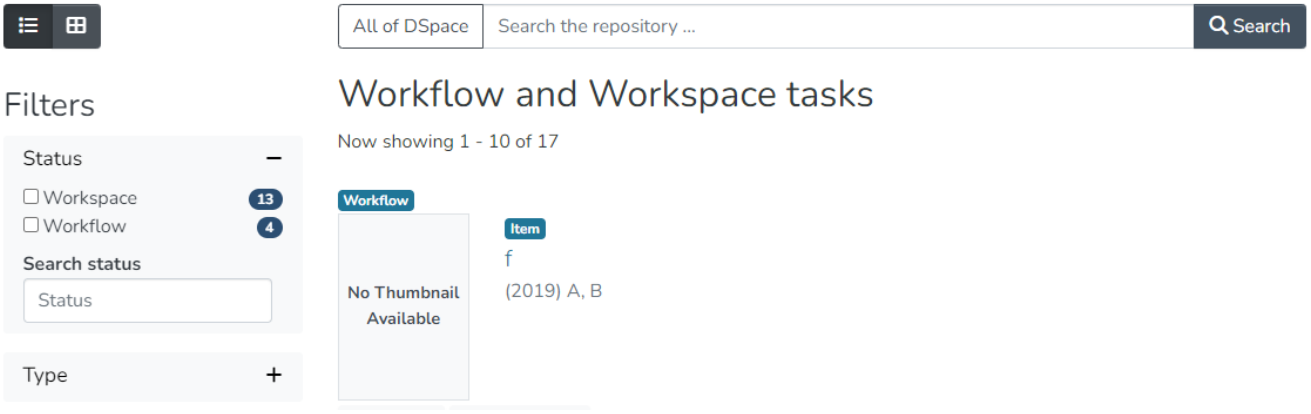
- Full editorial control (EDITOR)
- View item contents (OBSERVER)

Once the default set has been applied, a system administrator may modify them as they would any other policy set in DSpace

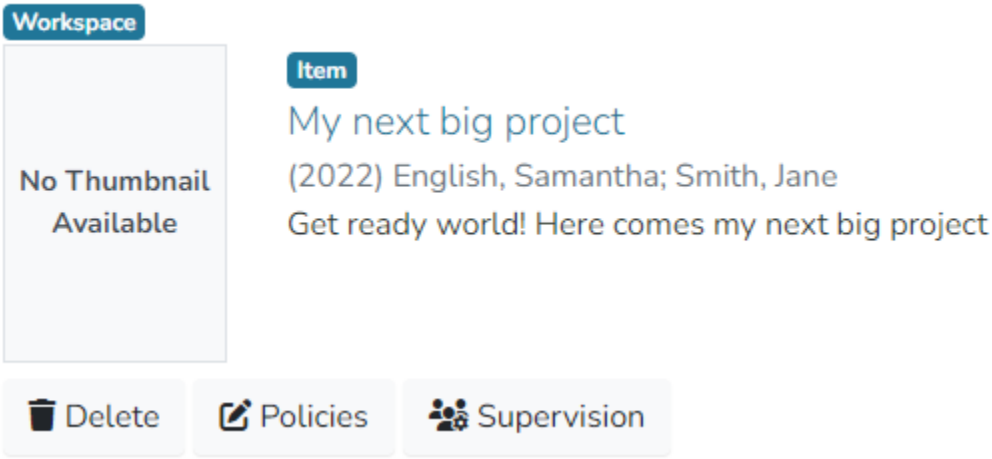
This functionality could also be used in situations where researchers wish to collaborate on a particular submission, although there is no particular collaborative workspace functionality.

Creating a Supervision Order

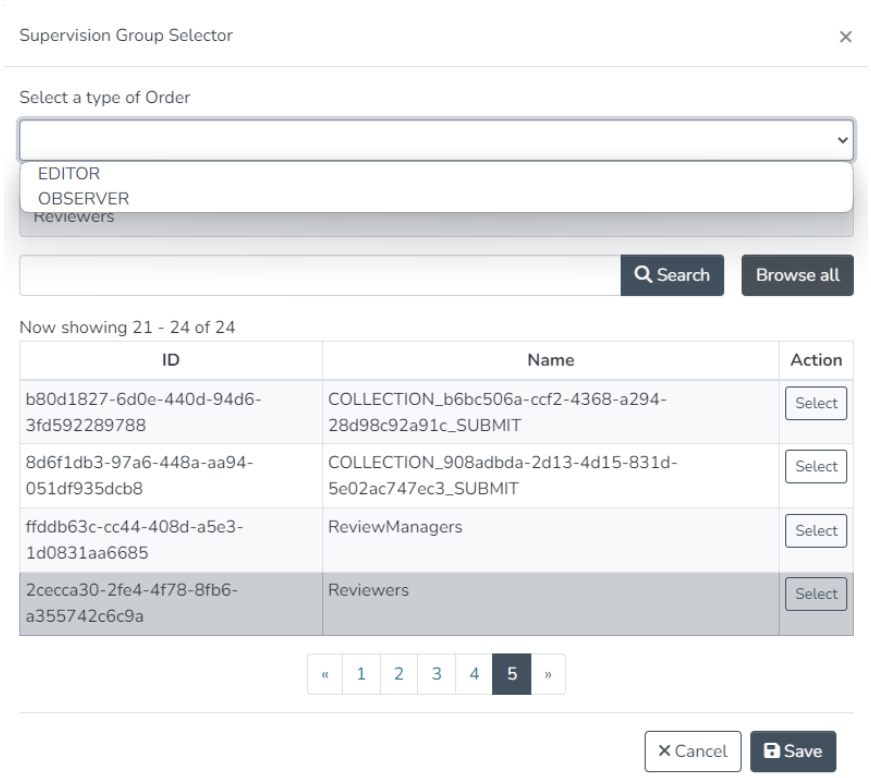
Login as an Administrator, and visit the "Administer Workflow" sidebar menu. From this screen you can see all Items that are either in the "Workspace" (pre-submission) or "Workflow" (workflow approval process) status.



For Items that are in the "Workspace", it is possible to create a supervision order by clicking on the "Supervision" button.



After clicking "Supervision", you'll be able to create a Supervision order by selecting the "Type of Order" (EDITOR or OBSERVER) and assigning those permissions to an existing DSpace Group.



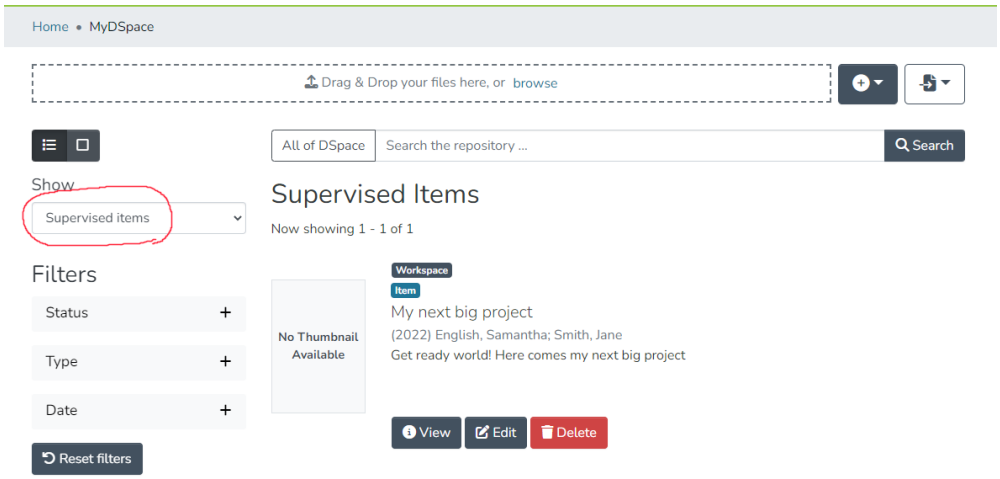
In DSpace, there are currently two Types of Orders:

- EDITOR - The supervising group is given ADD, WRITE, and READ access to the item (but not any bundles or bitstreams that already exist). Any new bundles or bitstreams inherit the supervising group's policy to permit ADD, WRITE and READ operations.
    - *NOTE:* Keep in mind, this does NOT give the supervising group REMOVE policies on the Bundle or DELETE on the Item. This means the supervising group is only able to edit the metadata & add additional bitstreams. They are NOT able to remove existing bitstreams or delete the Item unless additional policies are manually added.
- OBSERVER - The supervising group is given READ access to the item (but not to any bundles or bitstreams that already exist). Any new bundles or bitstreams inherit the supervision group's policy to permit READ operations.
    - NOTE: At this time, there is a known issue where the OBSERVER group will still see the "Edit" and "Delete" buttons, but are unable to perform those actions. https://github.com/DSpace/dspace-angular/issues/2094

Keep in mind, *you can adjust the permissions defined to any order after creating the order!* Simply click on the "Policies" button on the "Administer Workflow" page to adjust the default policies for that supervising group!

Supervising a Submission

Once a Supervision Order is created (see above step), all group members for the supervising group will see that Item in their "Supervised Items" list on their MyDSpace page:

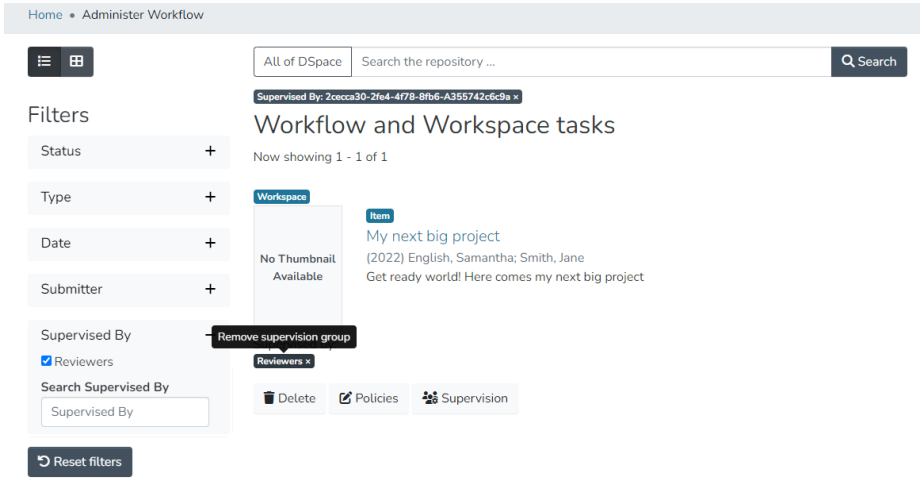


Based on the type of Submission Order (or additional permissions provided), all members of the supervising group will be able to view and/or edit that in-progress submission.

Managing Supervision Orders

At any time, an Administrator can remove or recreate Supervision Orders. This is also done from the "Administer Workflow" page.

On that page, a "Supervised By" filter exists, allowing you to locate all currently supervised items by the assigned group:



You can click on the "Supervised by" label under the supervised item to **remove** the existing supervision order. New orders can be added by clicking the "Supervision" button. You can also adjust any supervising group permissions by editing the policies directly by clicking on the "Policies" button.

Configurable Workflow

1 Introduction
2 How to Configure your Workflows
    2.1 WORKFLOWS
    2.2 STEPS
    2.3 ROLES
    2.4 ACTIONS
    2.5 CURATION
    2.6 HOW IT WORKS
3 Data Migration
    3.1 Workflowitem conversion/migration scripts
        3.1.1 Automatic migration
        3.1.2 Java based migration
4 Configuration
    4.1 Main workflow configuration
        4.1.1 workflowFactory bean (org.dspace.xmlworkflow.XmlWorkflowFactoryImpl)
        4.1.2 workflow beans (org.dspace.xmlworkflow.state.Workflow)
        4.1.3 role beans (org.dspace.xmlworkflow.Role)
        4.1.4 step beans (org.dspace.xmlworkflow.state.Step)
    4.2 Workflow actions configuration
        4.2.1 API configuration
            4.2.1.1 User Selection Action
            4.2.1.2 Processing Action
5 Authorizations
6 Database
    6.1 cwf_workflowitem
    6.2 cwf_collectionrole
    6.3 cwf_workflowitemrole
    6.4 cwf_pooltask
    6.5 cwf_claimtask
    6.6 cwf_in_progress_user
7 Additional workflow steps/actions and features
    7.1 Optional workflow steps: Select single reviewer workflow
    7.2 Optional workflow steps: Score review workflow
    7.3 Workflow overview features

Introduction

Workflows can be used to define how documents should be reviewed or edited after being submitted and/or imported into DSpace. 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. Each workflow can be compared to an action that is performed on an item between its submission to the repository and the moment it is archived and published in the repository. The concept behind this approach was modeled on the configurable submission system already present in DSpace.

How to Configure your Workflows

Every submission to DSpace goes through a workflow before it is published in the repository. A workflow consists of a series of steps, each of which is an opportunity for a reviewer, editor or collection administrator to modify and/or approve/reject the submission. A step may also begin by running a {Curation Task} over the submission.

Each collection is associated with a workflow. If no explicit association is made, the collection is assigned the default workflow. These associations are configured in `config/spring/api/workflow.xml` using the workflowMapping property of the XmlWorkflowFactory bean. To make an explicit association, add an entry to the list with the collection's Handle as the 'key' and the 'name' of a Workflow bean as the 'value-ref'.

Each step in a workflow is associated with a "role" which defines who can perform that step. Role members will be notified when a new submission needs their attention. Roles are defined by DSpace user groups. If you wish to have reviewers interact with incoming submissions, you must create and fill the necessary groups. See below for details.

WORKFLOWS

To create a new workflow, add another bean with the 'class' 'org.dspace.xmlworkflow.state.Workflow' and a unique 'name'. Give it a 'steps' property containing a list of the steps that should be entered in sequence, and a 'firstStep' property which names the step to be entered first. See the default workflow for an example. An existing step may be re-used if appropriate, or you can create one to suit.

STEPS

Aside from its name, a step has a "user selection method", a "role", "actions" and "outcomes".

A step's 'userSelectionMethod' is the name of an "action" of the user-selection type. A step may, for example, let itself be claimed (for a given submission) by a single user, or it may combine the actions of multiple users. A step has exactly one 'userSelectionMethod'. See more on actions below.

A step's role defines the set of users who may perform actions on a submission that has entered that step. See more on roles below.

A step's actions are the types of work that are done in the step. See more on actions below. More than one action may be listed.

A step's outcomes connect the role members' decisions with the next step to be performed. For example, this allows a role member to accept a submission and skip subsequent steps by going directly to the final step in the workflow.

To create a new step, add a bean with 'class' org.dspace.xmlworkflow.state.Step and the necessary properties, as discussed above. See the existing steps in workflow.xml for examples.

ROLES

You may re-use existing roles, or add your own. A role has a 'name', a 'scope', and optionally a 'description'. There are three kinds of roles:

- A COLLECTION role refers to a user group associated with a specific collection. It will be named {collectionID}_{roleName}. For example, a role 'editor' with COLLECTION scope, applied to collection 123, will refer to the user group named 'editor_123', while the same role applied to collection 456 will refer to the user group 'editor_456'.
- A REPOSITORY role refers to a fixed user group, whose name is the role's name. A REPOSITORY role named 'fred' will always refer to the user group 'fred'.
- An ITEM role is assigned by a previous action in the workflow. [NEEDS MORE EXPLANATION]

To create a new role, add a bean with 'class' org.dspace.xmlworkflow.Role, the appropriate 'scope', and a unique 'name'. Be sure that the related groups exist.

ACTIONS

Actions are defined separately in 'config/spring/api/workflow-actions.xml'.

A number of actions are already defined, and these should serve most needs. Actions are implemented in Java code, so if you need a new one then you will need to write some Java in addition to configuring it here.

There are two kinds of actions: user assignment and processing. A user assignment action selects one or more role members to execute a step. A processing action modifies the state of the submission.

To configure a new Action, create a bean with a unique 'id', 'class' equal to the fully qualified name of the Java class which implements the action, and 'scope' "prototype". Add properties, constructor arguments, etc. as required by the code.

CURATION

To attach a Curation Task to a workflow step, see Curation System. Tasks are executed at the beginning of a step, before role members are notified.

HOW IT WORKS

For details of how these concepts are implemented (for example, to create new actions) see the Workflow page under DSpace Development.

Data Migration

As of DSpace 7, Configurable Workflow is the only workflow system available in DSpace. It has fully replaced the older "traditional/basic workflow" system. One major difference is that Configurable Workflow is dynamic – if a user is added to a workflow approval task *after* a workflow has already begun, they will immediately get access to any existing items in workflow. Previously, this was not possible in the "traditional" workflow system.

Workflowitem conversion/migration scripts

Depending on the workflow that is used by a DSpace installation, different scripts can be used when migrating to the new workflow.

Automatic migration

As part of the upgrade to DSpace 7 or above, all your old policies, roles, tasks and workflowitems will be automatically updated from the original workflow to the Configurable Workflow framework. This is done via this command:

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

The "ignored" parameter will tell DSpace to run any previously-ignored migrations on your database. As the Configurable Workflow migrations have existed in the DSpace codebase for some time, this is the only way to force them to be run.

For more information on the "database migrate" command, please see Database Utilities.

Java based migration

In case your DSpace installation uses a customized version of the workflow, the migration script might not work properly and a different approach is recommended. Therefore, an additional Java based script has been created that restarts the workflow for all the workflowitems that exist in the original workflow framework. The script will take all the existing workflowitems and place them in the first step of the 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 configurable workflow.

To execute the script, run the following CLI command:

```
[dspace]/bin/dspace dsrun org.dspace.xmlworkflow.migration.RestartWorkflow -e admin@myrespository.org
```

The following arguments can be specified when running the script:

- -e: specifies the username of an administrator user
- -n: if sending submissions through the workflow, send notification emails
- -p: the provenance description to be added to the item
- -h: help

Configuration

Main workflow configuration

As of DSpace 7, the `workflow.xml` configuration file has been migrated to use Spring Bean syntax (instead of a custom XML format). The structure of this XML has changed. If you need help migrating your old `workflow.xml` file (which started with a `<wf-config>` tag) to the new format (using `<bean>` tags), an XSLT script is available: workflow-migration.xsl

The workflow main configuration can be found in the workflow.xml file, located in **`[dspace]/config/spring/api/workflow.xml`** . An example of this workflow configuration file can be found below.

```
<beans>
    <bean class="org.dspace.xmlworkflow.XmlWorkflowFactoryImpl">
        <property name="workflowMapping">
            <util:map>
                <entry key="defaultWorkflow" value-ref="defaultWorkflow"/>
<!--            <entry key="123456789/4" value-ref="selectSingleReviewer"/>-->
<!--            <entry key="123456789/5" value-ref="scoreReview"/>-->
            </util:map>
        </property>
    </bean>

    <!--Standard DSpace workflow-->
    <bean name="defaultWorkflow" class="org.dspace.xmlworkflow.state.Workflow">
        <property name="firstStep" ref="reviewstep"/>
        <property name="steps">
            <util:list>
                <ref bean="reviewstep"/>
                <ref bean="editstep"/>
                <ref bean="finaleditstep"/>
            </util:list>
        </property>
    </bean>

    <bean id="{workflow.id}"
                class="org.dspace.xmlworkflow.state.Workflow">
            <!-- Another workflow configuration-->
    </bean>

        <!-- Role beans.  See below. -->

        <!-- Step beans.  See below. -->

</beans>
```

workflowFactory bean (org.dspace.xmlworkflow.XmlWorkflowFactoryImpl)

The workflow map contains a mapping between collections in DSpace and a workflow configuration, and is defined by the `workflowMapping` property of the workflow factory. Similar to the configuration of the submission process, the mapping can be done based on the handle of the collection. The mapping with "defaultWorkflow" 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 "`entry`" element with two attributes:

- key: can either be a collection handle or "defaultWorkflow"
- value-ref: the value of this attribute points to one of the workflow configurations defined by the "Workflow" beans

workflow beans (org.dspace.xmlworkflow.state.Workflow)

The workflow bean is a repeatable XML element and represents one workflow process. It requires the following:

- "`name`" attribute: a unique name used for the identification of the workflow and used in the workflow to collection mapping
- "`firstStep`" property: the identifier of the first step of the workflow. This step 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 "`firstStep`" property will he the first step the item will go through.
- "`steps`" property: a list of all steps within this workflow (in the order they will be processed).

role beans (org.dspace.xmlworkflow.Role)

Each workflow step has defined "role" property. A role represents one or more existing 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" bean and has the following:

- "`id`" attribute: a unique identifier (in one workflow process) for the role
- "`description`" property: optional attribute to describe the role
- "`scope`" property: optional attribute that is used to find our group and must have one of the following values, which are defined as constant fields of `org.dspace.xmlworkflow.Role.Scope`:
  - 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`" property: The name specified in the name attribute of a role will be used to lookup an eperson group 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.

```
<bean id="reviewer" class="org.dspace.xmlworkflow.Role">
    <property name="scope" value="#{ T(org.dspace.xmlworkflow.Role.Scope).COLLECTION}"/>
    <property name="name" value="Reviewer"/>
    <property name="description" value="The people responsible for this step are able to edit the metadata of
incoming submissions, and then accept or reject them."/>
</bean>
```

step beans (org.dspace.xmlworkflow.state.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 in order to further configure it:

- "`name`" attribute: The name attribute specifies a unique identifier for the step. This identifier 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.
- "`userSelectionMethod`" property: This attribute defines the `UserSelectionAction` that will be used to determine how to attach users to this step for a workflow-item. The value of this attribute must refer to the identifier of an action bean in the workflow-actions.xml. Examples of the user attachment to a step are the currently used system of a task pool or as an alternative directly assigning a user to a task.
- "`role`" property: 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

```
<bean name="reviewstep" class="org.dspace.xmlworkflow.state.Step">
    <property name="userSelectionMethod" ref="claimaction"/>
    <property name="role" ref="reviewer"/>
    <property name="outcomes">
        <util:map>
            <entry key="#{ T(org.dspace.xmlworkflow.state.actions.ActionResult).OUTCOME_COMPLETE}"
                   value-ref="editstep"/>
        </util:map>
    </property>
    <property name="actions">
        <util:list>
            <ref bean="reviewaction"/>
        </util:list>
    </property>
</bean>
```

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 "`outcomes`". 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 "`outcomes`" 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.

```
<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 configuration refers to a bean definition in the `workflow-actions.xml` file. In order to define a new user selection action, 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 of the task. This is the same as the `id` attribute of the bean and is used by the workflow configuration 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 similarly 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.

Authorizations

Currently, the authorizations are always granted and revoked based on the tasks that are available for certain users and groups. The types of authorization policies that is granted for each of these is always the same:

- READ
- WRITE
- ADD
- DELETE

Database

The workflow uses a separate metadata schema named `workflow`. The fields this schema contains can be found in the `[dspace]/config/registries` directory and in the file `workflow-types.xml`. This schema is only used when using the score reviewing system at the moment, but one could always use this schema if metadata is required for custom workflow steps.

The following tables have been added to the DSpace database. All tables are prefixed with 'cwf_' to avoid any confusion with the existing workflow related database tables:

cwf_workflowitem

The cwf_workflowitem table contains the different workflowitems in the workflow. This table has the following columns:

- workflowitem_id: The identifier of the workflowitem and primary key of this table
- item_id: The identifier of the DSpace item to which this workflowitem refers.
- collection_id: The collection to which this workflowitem is submitted.
- multiple_titles: Specifies whether the submission has multiple titles (important for submission steps)
- published_before: Specifies whether the submission has been published before (important for submission steps)
- multiple_files: Specifies whether the submission has multiple files attached (important for submission steps)

cwf_collectionrole

The cwf_collectionrole table represents a workflow role for one collection. This type of role is the same as the roles that existed in the original workflow meaning that for each collection a separate group is defined to 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

Additional workflow steps/actions and features

These optional steps are only supported in 7.5 and later.

These optional workflow steps are pre-defined in the "workflow.xml" but are not used by default.

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.

- selectReviewerStep: During this step, 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.
    - The available users to select from are defined in the "action.selectrevieweraction.group" setting in workflow.cfg. This setting must list the name of a group of reviewers to select from (default value = "Reviewers" group).
- singleUserReviewStep: 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.
    - In workflow.cfg, there's an option to allow these reviewers to be able to edit files & metadata. When "workflow.reviewer.file-edit=true", the selected user will be allowed to edit the workflow item. By default they cannot.

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. Depending on the configuration, a different number of users can be required to execute the task (default is requiredusers=2). 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.
    - In workflow.cfg, there's an option to allow these reviewers to be able to edit files & metadata. When "workflow.reviewer.file-edit=true", the selected user will be allowed to edit the workflow item. By default they cannot.
- evaluationStep: During the evaluationstep, no user interface is required. The workflow system will automatically execute the step that evaluates the different scores (which corresponds to a rating from 1-5). In case the average score is greater than the average "minimumAcceptanceScore", the item is approved, otherwise it is rejected. (The minimum average score is set by adjusting the `minimumAcceptanceScore` property passed to `evaluationactionAPI` in `config/spring/api/workflow-actions.xml`.)

Workflow overview features

The DSpace UI also provides a feature to allow Administrators to see & administer all active workflows (workitems). This feature is provided in the "Administer Workflow" menu option. Currently, the Administrator has the ability to permanently delete the workflowitem, or to send it back to the original submitter.

Importing and Exporting Content via Packages

1 Package Importer and Exporter
- 1.1 Supported Package Formats
- 1.2 Ingesting
  - 1.2.1 Ingestion Modes & Options
    - 1.2.1.1 Ingesting a Single Package
    - 1.2.1.2 Ingesting Multiple Packages at Once
  - 1.2.2 Restoring/Replacing using Packages
    - 1.2.2.1 Default Restore Mode
    - 1.2.2.2 Restore, Keep Existing Mode
    - 1.2.2.3 Force Replace Mode
- 1.3 Disseminating
  - 1.3.1 Disseminating a Single Object
  - 1.3.2 Disseminating Multiple Objects at Once
- 1.4 Archival Information Packages (AIPs)
- 1.5 METS packages

Package Importer and Exporter

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

To see all the options, invoke it as:

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

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

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

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

Supported Package Formats

DSpace comes with several pre-configured package ingestion and dissemination plugins, which allow you to import/export content in a variety of formats.

Pre-Configured Submission Package (SIP) Types

- AIP - Ingests content which is in the DSpace Archival Information Package (AIP) format. This is used as part of the DSpace AIP Backup and Restore process
- DSPACE-ROLES - Ingests DSpace users/groups in the DSPACE-ROLES XML Schema. This is primarily used by the DSpace AIP Backup and Restore process to ingest/replace DSpace Users & Groups.
- METS - Ingests content which is in the DSpace METS SIP format
- PDF - Ingests a single PDF file (where basic metadata is extracted from the file properties in the PDF Document).

Pre-Configured Dissemination Package (DIP) Types

- AIP - Exports content which is in the DSpace Archival Information Package (AIP) format. This is used as part of the DSpace AIP Backup and Restore process
- DSPACE-ROLES - Exports DSpace users/groups in the DSPACE-ROLES XML Schema. This is primarily used by the DSpace AIP Backup and Restore process to export DSpace Users & Groups.
- METS - Exports content in the DSpace METS SIP format

For a list of all package ingestion and dissemination plugins that are currently installed in your DSpace, you can execute:

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

Some packages ingestion and dissemination plugins also have custom options/parameters. For example, to see a listing of the custom options for the "METS" plugin, you can execute:

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

Ingesting

Ingestion Modes & Options

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

1. Submit/Ingest Mode (`-s` option, default) – submit package to DSpace in order to create a new object(s)
2. Restore Mode (`-r` option) – restore pre-existing object(s) in DSpace based on package(s). This also attempts to restore all handles and relationships (parent/child objects). This is a specialized type of "submit", where the object is created with a known Handle and known relationships.
3. Replace Mode (`-r -f` option) – replace existing object(s) in DSpace based on package(s). This also attempts to restore all handles and relationships (parent/child objects). This is a specialized type of "restore" where the contents of existing object(s) is replaced by the contents in the AIP(s). By default, if a normal "restore" finds the object already exists, it will back out (i.e. rollback all changes) and report which object already exists.

Ingesting a Single Package

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

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

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

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

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

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

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

Ingesting Multiple Packages at Once

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

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

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

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

for example:

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

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

Not All Packagers Support Bulk Ingest

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

- METS Packager Plugin
- AIP Packager Plugin

Restoring/Replacing using Packages

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

There are currently three restore modes:

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

Default Restore Mode

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

Use this 'packager' command template:

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

For example:

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

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

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

Restore, Keep Existing Mode

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

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

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

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

For example:

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

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

Force Replace Mode

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

Potential for Data Loss

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

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

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

For example:

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

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

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

Disseminating

Disseminating a Single Object

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

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

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

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

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

Disseminating Multiple Objects at Once

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

For example, use this 'packager' command template:

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

for example:

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

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

Archival Information Packages (AIPs)

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

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

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

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

METS packages

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

The DSpace METS SIP profile is available at: DSpaceMETSSIPProfile

Importing and Exporting Items via Simple Archive Format

1 Item Importer and Exporter
- 1.1 DSpace Simple Archive Format
  - 1.1.1 dublin_core.xml or metadata_[prefix].xml
  - 1.1.2 contents file
  - 1.1.3 relationships file
- 1.2 Configuring metadata_[prefix].xml for a Different Schema
- 1.3 Importing Items
  - 1.3.1 Adding Items to a Collection from a directory
  - 1.3.2 Adding Items to a Collection from a zipfile
  - 1.3.3 Replacing Items in a Collection
  - 1.3.4 Deleting or Unimporting Items in a Collection
  - 1.3.5 Other Options
  - 1.3.6 UI Batch Import
- 1.4 Exporting Items
  - 1.4.1 UI Batch Export

Item Importer and Exporter

DSpace has a set of command line tools for importing and exporting items in batches, using the DSpace Simple Archive Format. Apart from the offered functionality, these tools serve as an example for users who aim to implement their own item importer.

DSpace Simple Archive Format

The basic concept behind the DSpace's Simple Archive Format is to create an archive, which is a directory containing one subdirectory per item. Each item directory contains a file for the item's descriptive metadata, and the files that make up the item.

```
archive_directory/
    item_000/
        dublin_core.xml         -- qualified Dublin Core metadata for metadata fields belonging to the 'dc'
schema.
        metadata_[prefix].xml   -- metadata in another schema.  The prefix is the name of the schema as
registered with the metadata registry.
        contents                -- text file containing one line per filename.
                collections                                 -- (Optional) text file that contains the handles of
the collections the item will belong to. Each handle in a row.
                                                                -- Collection in first line will be the owning
collection.
                handle                                      -- contains the handle assigned/to be assigned to
this resource
        relationships           -- (Optional) If importing Entities, you can specify one or more relationships
to create on import
        file_1.doc              -- files to be added as bitstreams to the item.
        file_2.pdf
    item_001/
        dublin_core.xml
        contents
        file_1.png
        ...
```

dublin_core.xml or metadata_[prefix].xml

The `dublin_core.xml` or `metadata_[prefix].xml` file has the following format, where each metadata element has its own entry within a `<dcvalue>` tagset. There are currently three tag attributes available in the `<dcvalue>` tagset:

- `element` - the Dublin Core element
- `qualifier` - the element's qualifier
- `language` - (optional) ISO language code for element

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

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

When providing urls as values for fields that contain the ampersand (&) symbol, the ampersands in these urls have to be encoded as **&**

Every metadata field used, must be registered via the metadata registry of the DSpace instance first. See Metadata and Bitstream Format Registries.

Recommended Metadata

It is recommended to minimally provide "dc.title" and, where applicable, "dc.date.issued". Obviously you can (and should) provide much more detailed metadata about the Item. For more information see: Metadata Recommendations.

contents file

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.

Supported in 7.2 or above for 'import' only

The IIIF metadata feature was added in 7.2 and is only supported on *import* ('add' mode) of an SAF package.

For IIIF enabled items, the bitstream name may optionally be followed by any of the following:

- `\tiiif-label:IIIFLABEL`
- `\tiiif-toc:IIIFTOC`
- `\tiiif-width:IIIFWIDTH`
- `\tiiif-height:IIIFHEIGHT`

Where:

'IIIFLABEL' is the label that will be used for the image in the viewer.

'IIIFTOC' is the label that will be used for a table of contents entry in the viewer.

'IIIFWIDTH' is the image width that will be used for the IIIF canvas.

'IIIFHEIGHT' is the image height that will be used for the IIIF canvas.

relationships file

Supported in 7.1 or above for 'import' only.

This feature was added in 7.1. Currently the 'relationships' file is only supported on *import* ('add' mode) of an SAF package. See note at bottom of this section about using the "metadata_relation.xml" if you wish to export & update relationships.

The optional `relationships` file enumerates the relationships of this Entity to other Entities (either already in the system, or also specified in your SAF import batch). This allows entities to be linked to new or existing entities during import. Entities can be linked to other entities in this import set by referring to their import subfolder name. Because relationships can only be created for Entities, it can only be used when importing Configurable Entities.

Each line in the file contains a relationship type key and an item identifier in the following format:

```
relation.<relation_key> <handle|uuid|folderName:import_item_folder|schema.element[.qualifier]:value>
```

The input_item_folder should refer the folder name of another item in this import batch. Example:

```
relation.isAuthorOfPublication 5dace143-1238-4b4f-affb-ed559f9254bb
relation.isAuthorOfPublication 123456789/1123
relation.isOrgUnitOfPublication folderName:item_001
relation.isProjectOfPublication project.identifier.id:123
relation.isProjectOfPublication project.identifier.name:A Name with Spaces
```

During initial import, new items are stored in a map keyed by the item folder name. Once the initial import is complete, a second pass checks for a 'relationships' manifest file in each folder and creates a relationship of the specified type to the specified item.

Don't forget Entities require a "dspace.entity.type" metadata field

Remember, if you are creating *new* Entities via an SAF package, those Entities MUST specify a "dspace.entity.type" metadata field. Because this metadata field is in the "dspace" schema, it MUST be specified in a "metadata_dspace.xml", similar to:

metadata_dspace.xml

```
<dublin_core schema="dspace">
  <dcvalue element="entity" qualifier="type">Publication</dcvalue>
</dublin_core>
```

Relationships to existing Entities can also be created via metadata_relation.xml

If you already know the UUID of an existing Entity that you want to relate to, you can also create/update the "metadata_relation.xml" file to add/update the relationship, similar to:

metadata_relation.xml

```
<dublin_core schema="relation">
  <dcvalue element="isAuthorOfPublication">5dace143-1238-4b4f-affb-ed559f9254bb</dcvalue>
</dublin_core>
```

The "relationships" file is primarily for creating relationships between Entities in the *same import batch.* Of course, you can also choose to use the "relationships" file to create new relationships to existing Entities instead of creating/updating the "metadata_relation.xml" file. The main advantage of the "metadata_relation.xml" file is that it is used both on export and import, while the "relationships" file is only used on import at this time.

Configuring `metadata_[prefix].xml` for a Different Schema

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

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

```
<dublin_core schema="etd">
     <dcvalue element="degree" qualifier="department">Computer Science</dcvalue>
     <dcvalue element="degree" qualifier="level">Masters</dcvalue>
     <dcvalue element="degree" qualifier="grantor">Michigan Institute of Technology</dcvalue>
</dublin_core>
```

Importing Items

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

Command used:	`[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 its 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
`-v or --validate`	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
`-z or --zip`	Name of zipfile

‡ These are mutually exclusive.

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

Adding Items to a Collection from a directory

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

- eperson
- Collection ID (either Handle (e.g. 123456789/14) or UUID
- 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.** You can use it for replacing or deleting (unimporting) the mapped items.

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

Adding Items to a Collection from a zipfile

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

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

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

or by using the short form:

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

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

Testing. You can add `--validate` (or `-v`) 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 a Collection

Replacing existing items is relatively easy. Remember that mapfile you saved above? 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
```

If you wish to replace content using a Zipfile, that's also possible. The command is similar. But, in this situation "-s" refers to the directory of the zip file, and "-z" gives the name of the zipfile:

```
[dspace]/bin/dspace import -r -e joe@user.com -c collectionID -s zipfile_dir -z filename.zip -m 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 saved above? The command is (in short form):

```
[dspace]/bin/dspace import -e joe@user.com -d -m mapfile
```

In long form:

```
[dspace]/bin/dspace import --eperson=joe@user.com --delete --mapfile mapfile
```

Other Options

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

  If you omit the -c flag, which is otherwise mandatory, the ItemImporter searches for a file named "collections" in each item directory. This file should contain a list of collections, one per line, specified either by their handle, or by their internal db id. The ItemImporter then will put the item in each of the specified collections. The owning collection is the collection specified in the first line of the collections file.

  If both the -c flag is specified and the collections file exists in the item directory, the ItemImporter will ignore the collections file and will put the item in the collection specified on the command line.

  Since the collections file can differ between item directories, this gives you more fine-grained control of the process of batch adding items to collections.

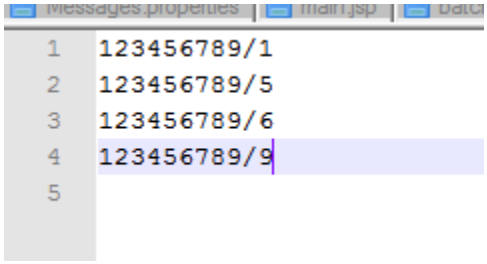
UI Batch Import

Available in DSpace 7.4 and above.

Batch import can also take place via the Administrator's UI. The steps to follow are:

A. Prepare the data

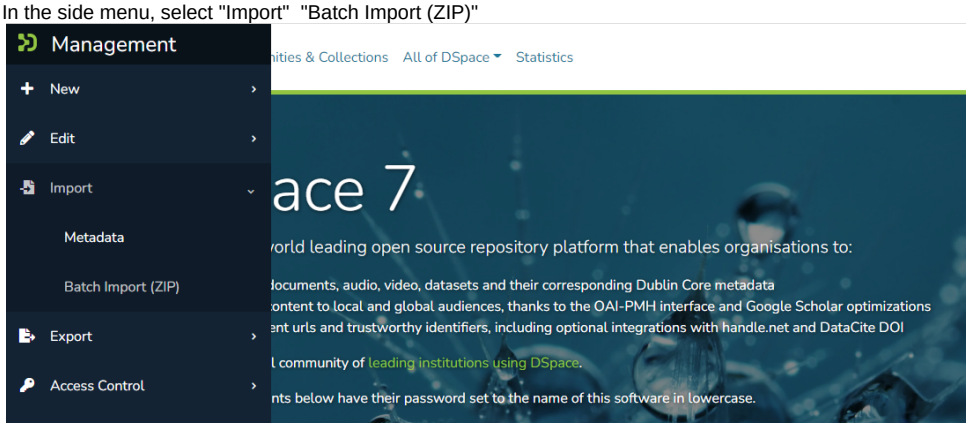
1. Items, i.e. the metadata and their bitstreams, must be in the Simple Archive Format described earlier in this chapter. Thus, for each item there must be a separate directory that contains the corresponding files of the specific item.
2. Moreover, in each item directory, there can be another file that describes the collection or the collections that this item will be added to. The name of this file must be "collections" and it is optional. It has the following format:



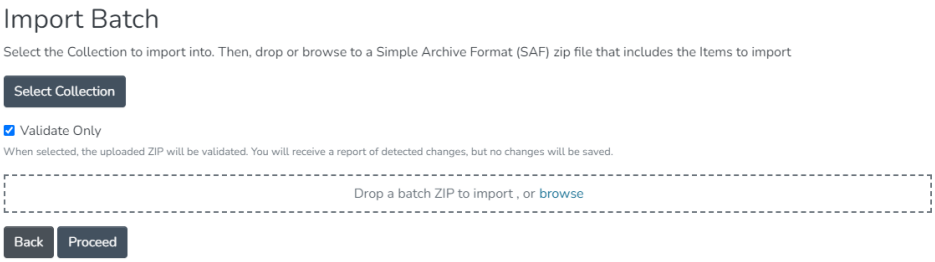
   Each line contains the handle of the collection. The collection in the first line is the owning collection while the rest are the other collections that the item should belong to.
3. Compress the item directories into a ZIP file. Please note that you need to zip the actual item directories and not just the directory that contains the item directorie s. Thus, the final zip file must directly contain the item directorie s.

B. Import the items via the UI

1. Login as an Administrator.
2. In the side menu, select "Import" "Batch Import (ZIP)"



3. From the "Import Batch" page:
   a. Select the Collection you are importing into.
   b. Drag & drop the ZIP file into the drop box (or browse to it on your filesystem).
   c. Choose whether you want to "Validate Only" or not.
      i. When selected, DSpace will **test** the batch import process, but no content will be batch imported. This allows you to validate the results of the import process before doing the import.
      ii. When deselected, DSpace will do the batch import.



4. Clicking "Proceed" will start the Batch Import. This creates a new "Process" which begins the upload of the batch. *Depending on the size of the batch, this process may take some time to complete.* You can refresh to page to see the current status, or go back to the list of processes ("Processes" menu in sidebar) to check on its status. Once the process is COMPLETED, you will see a log of the results and a mapfile (which can be used to make later updates).
5. All prior imports will be listed in the "Processes" menu, until their corresponding process entry is deleted. Once you are satisfied with the import and have no need to see the logs or mapfile, you may wish to delete that process entry in order to free up storage space (as your uploaded ZIP will be retained in DSpace until the process is deleted). A "process-cleaner" script can also be started from the "Processes" page which can be used to bulk delete old processes.

It is also possible to start an "import" directly from the "Processes" menu. This allows you to specify additional options/flags which are normally only available to the command-line "import" tool (see documentation above).

Exporting Items

The item exporter can export a single item or a collection of items, and creates a DSpace simple archive in the aforementioned format for each exported item. The items are exported in a sequential order in which they are retrieved from the database. As a consequence, the sequence numbers of the item subdirectories (item_000, item_001) are not related to DSpace handle or item IDs.

Command used:	`[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 path where you want the file of items to be placed.
`-n` or `--number`	Sequence number to begin with. Whatever number you give, this will be the name of the first directory created for your export. The layout of the export directory is the same as the layout used for import.
`-m` or `--migrate`	Export the item/collection for migration. This will remove the handle and any other metadata that will be re-created in the new instance of DSpace.
-x or --exclude-bitstreams	Do not export bitstreams. See the usage scenario below.
`-h` or `--help`	Brief Help.

Exporting a Collection

The CLI command to export the items of a collection:

```
[dspace]/bin/dspace export --type=COLLECTION --id=collectionID_or_handle --dest=/path/to/destination --
number=seq_num
```

Short form:

```
[dspace]/bin/dspace export -t COLLECTION -i collectionID_or_handle -d /path/to/destination -n seq_num
```

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.

Exporting a Single Item

To export a single item use the keyword *ITEM* and give the item ID as an argument:

```
[dspace]/bin/dspace export --type=ITEM --id=itemID_or_handle --dest=/path/to/destination --number=seq_num
```

Short form:

```
[dspace]/bin/dspace export -t ITEM -i itemID_or_handle -d /path/to/destination -n seq_num
```

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

The `-m` Argument

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

The -x Argument

Using the -x argument will do the standard export except for the bitstreams which will not be exported. If you have full SAF without bitstreams and you have the bitstreams archive (which might have been imported into DSpace earlier) somewhere near, you could symlink original archive files into SAF directories and have an exported collection which almost doesn't occupy any space but otherwise is identical to the exported collection (i.e. could be imported into DSpace). In case of huge collections -x mode might be substantially faster than full export.

UI Batch Export

Available in DSpace 7.4 and above.

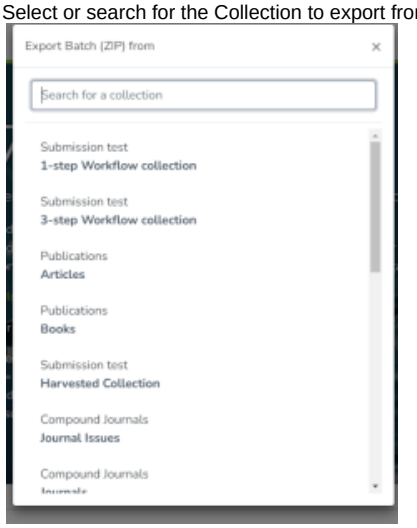
Batch export can also take place via the Administrator's UI. The default file size upload limit is 512MB, and is being configured in the spring boot application.properties file.

The steps to follow are:

1. Login as an Administrator.
2. In the side menu, select "Import" "Batch Export (ZIP)"



3. Select or search for the Collection to export from:



4. Clicking "Export" will start the Batch Export. This creates a new "Process" which begins export process. *Depending on the size of the export, this process may take some time to complete.* You can refresh to page to see the current status, or go back to the list of processes ("Processes" menu in sidebar) to check on its status. Once the process is COMPLETED, you will see a log of the results and an exported ZIP file which you can download for the results.
5. All prior exports will be listed in the "Processes" menu, until their corresponding process entry is deleted. Once you are satisfied with the export and have downloaded the ZIP, you may wish to delete that process entry in order to free up storage space (as your exported ZIP will be retained in DSpace until the process is deleted). A "process-cleaner" script can also be started from the "Processes" page which can be used to bulk delete old processes.

It is also possible to start an "export" directly from the "Processes" menu. This allows you to specify additional options/flags which are normally only available from the command-line "export" tool (see documentation above). It also allows you to export a single Item.

Registering Bitstreams via Simple Archive Format

1 Overview
- 1.1 Accessible Storage
- 1.2 Registering Items Using the Item Importer
- 1.3 Internal Identification and Retrieval of Registered Items
- 1.4 Exporting Registered Items
- 1.5 Deleting Registered Items

Registering is not Importing

The procedures below will **not import** the actual bitstreams into DSpace. They will merely inform DSpace of an existing location where these Bitstreams can be found. Please refer to Importing and Exporting Items via Simple Archive Format for information on importing metadata and bitstreams.

Overview

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

Accessible Storage

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

Registering Items Using the Item Importer

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

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

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

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

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

where

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

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

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

(or by using the long form)

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

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

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

The `--replace` flag will function as described in Importing Items but care should be taken to consider different cases and implications. With old items and new items being registered or ingested normally, there are four combinations or cases to consider. Foremost, an old registered item deleted from DSpace using `--replace` will not be removed from the storage. See Deleting Registered Items. where 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.

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

Exporting Registered Items

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

Deleting Registered Items

If a registered item is deleted from DSpace, (either interactively or by using the `--delete` or `--replace` flags described in Importing and Exporting Items via Simple Archive Format) the item will disappear from DSpace but its registered content files will remain in place just as they were prior to registration. Bitstreams not registered but added by DSpace as part of registration, such as `license.txt` files, will be deleted.

Importing Items via basic bibliographic formats (Endnote, BibTex, RIS, CSV, etc) and online services (arXiv, PubMed, CrossRef, CiNii, etc)

1 Introduction
2 Supported External Sources
3 Disabling an External source
4 Submitting starting from external sources
5 Submitting starting from bibliographic file
6 More Information

In DSpace 7.0, the Biblio-Transformation-Engine (BTE) was removed in favor of Live Import from external sources. All online services and bibliographic formats previously supported by BTE have been moved or are being moved to the External Sources framework.

Introduction

This documentation explains the features and the usage of the importer framework. The importer framework is built into both the REST API and User Interface. Currently supported formats include:

- Drag & drop of Endnote, BibTex, RIS, TSV, CSV, arXiv, PubMed. From the MyDSpace page, dragging & dropping one of these files will start a new submission, extracting the metadata from the file.
- Import via ORCID, PubMed, Sherpa Journals, Sherpa Publishers. From the MyDSpace page, you can select to start a new submission by searching an external source.

Supported External Sources

DSpace supports importing metadata from a variety of online services. Some of these services can ONLY support out-of-the-box Configurable Entities.

Supported online services are all configured on the backend in the `[dspace]/config/spring/api/external-services.xml` file. To disable a service, simply comment it out in that file.

Online providers available out of the box include:

- NASA Astrophysics Data System (ADS) lookup (Supported for creating new Items, or "Publication" Entities). Can be configured via "ads.*" settings in external-providers.cfg. *REQUIRES an API key to function, signup at https://ui.adsabs.harvard.edu/help/api/*
- arXiv lookup (Supported for creating new Items, or "Publication" Entities).
- CiNii lookup (Supported for creating new Items, or "Publication" Entities). Can be configured via "cinii.*" settings in external-providers.cfg. *REQUIRES an API key to function, signup at https://support.nii.ac.jp/en/cinii/api/developer*
- CrossRef lookup (Supported for creating new Items, or "Publication" Entities). Can be configured via "crossref.*" settings in external-providers.cfg
- European Patent Office (EPO) lookup (Supported for creating new Items, or "Publication" Entities). Can be configured via "epo.*" settings in external-providers.cfg. *REQUIRES an API key to function, signup at https://developers.epo.org/*
- OpenAlex lookup (Supported for creating new Items or Publication Entities). See also Live Import from external sources#OpenAlexIntegration
- ORCID
  - ORCID author lookup (Only supported for creating new "Person" Entities). Can be configured via "orcid.*" settings in orcid.cfg.
  - ORCID publication lookup (Supported for creating new Items, or "Publication" Entities). Allows you to lookup a publication based on an author's ORCID. Can be configured via "orcid.*" settings in orcid.cfg.
- PubMed
  - Search PubMed (Supported for creating new Items, or "Publication" Entities). Can be configured via "pubmed.*" settings in external-providers.cfg
  - Search PubMed Europe (Supported for creating new Items, or "Publication" Entities). Can be configured via "pubmedeurope.*" settings in external-providers.cfg
- ROR lookup (Supported for creating new "Organization" Entites). It's enabled via "`ror.*`" properties inside the "`external-providers.cfg`". Other specific configuration for the "data-provider" and the medatada mapping can be found respectively inside "`external-services.xml`" and " `ror-integration.xml`" files. The API is publicly available, for some other implementation details you can refer to these PRs:
  - `DSpace`
    - ROR Integration - Live Import
    - ROR Integration - OAI PMH & Orcid
  - `dspace-angular`
    - ROR Integration - Identifier Visualization
- SciELO lookup (Supported for creating new Items, or "Publication" Entities). Can be configured via "scielo.*" settings in external-providers.cfg.
- Scopus lookup (Supported for creating new Items, or "Publication" Entities). Can be configured via "scopus.*" settings in external-providers.cfg. *REQUIRES an API key to function, signup at https://dev.elsevier.com*
- Sherpa Romeo
  - Sherpa Journals by ISSN (Only supported for creating new "Journal" Entities)
  - Sherpa Journals (Only supported for creating new "Journal" Entities) - supports looking up a Journal by title
  - Sherpa Publishers (Only supported for creating new "OrgUnit" Entities)
- VuFind lookup (Supported for creating new Items, or "Publication" Entities). Can be configured via "vufind.*" settings in external-providers.cfg
- Web of Science lookup (Supported for creating new Items, or "Publication" Entities). Can be configured via "wos.*" settings in external-providers.cfg. *REQUIRES a paid API key to function, signup at https://developer.clarivate.com/apis/wos*
  - Currently this WOS integration requires a paid license and does NOT yet support the WOS Starter API. See this issue ticket for more information: https://github.com/DSpace/DSpace/issues/8695

Disabling an External source

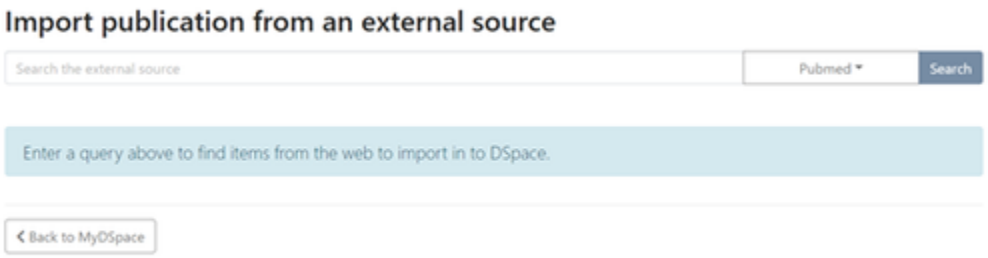
By default, DSpace has all external sources enabled in the `[dspace]/config/spring/api/external-services.xml` file. However, because some external services may require a **paid subscription** , sites may wish to disable any external sources that they do not want to use.

To disable an external source, simply comment out its "`<bean>`" tag in the `external-services.xml` file. Comment it out using XML comment tags (<!-- and -->).

For example, this will disable the Scopus external service (which is one that requires a paid subscription):

```
<!--
<bean id="scopusLiveImportDataProvider" class="org.dspace.external.provider.impl.LiveImportDataProvider">
    <property name="metadataSource" ref="ScopusImportService"/>
    <property name="sourceIdentifier" value="scopus"/>
    <property name="recordIdMetadata" value="dc.identifier.scopus"/>
    <property name="supportedEntityTypes">
        <list>
            <value>Publication</value>
        </list>
    </property>
</bean>
-->
```

Submitting starting from external sources



1. From the MyDSpace page a new submission can be started not only using the submission form but also automatically populating metadata, importing them from several online services.
2. After choosing the external source to import from and inserting a term in search bar, the system will show the list of matching results.
3. When selecting an item, the system will display the metadata to be imported, according to the configured mapping.
4. Clicking on “Start submission” the system will display the submission forms filled with the imported metadata.

Submitting starting from bibliographic file

1. From the MyDSpace page, drag & drop the bibliographic file (e.g. bibtex, endnote, etc) onto the file dropbox
2. Select the collection to submit to
3. Submission will be created, with the metadata parsed out of that bibliographic file. The bibliographic file will also be attached as a Bitstream (in case some fields could not be successfully parsed). You can choose to keep it or delete it.

More Information

More information on configuring metadata mappings for various import formats / services can be found in the Live Import from external sources documentation. See the "Editing Metadata Mapping" section.

Exporting and Importing Community and Collection Hierarchy

1 Community and Collection Structure Importer
    1.1 Usage
    1.2 XML Import Format
2 Community and Collection Structure Exporter
    2.1 Usage

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. The presence of this argument engages import mode.
-o	Output xml file. **Required.** A copy of the input augmented with the Handles assigned to each new Community or Collection.
-e	Email of DSpace Administrator. **Required.**

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.

Community and Collection Structure Exporter

This command-line tool writes the current Community and Collection structure into an XML document in the format which is read by the importer. See above for format details.

Usage

```
[dspace]/bin/dspace structure-builder [-h] [-?] -x -e <eperson> -o <output>
```

Argument: short and long (if available) forms:	**`Description of the argument`**
-x, --export	Export the current structure as XML. The presence of this argument engages export mode.
-e, --eperson *email or netid*	User who is manipulating the repository's structure. **Required.** This user's rights determine access to communities and collections.
-o, --output *file path*	The exported structure is written here. **Required.**
-h or -?	Help

SWORDv1 Server

SWORD (Simple Web-service Offering Repository Deposit) is a protocol that allows the remote deposit of items into repositories. DSpace implements the SWORD protocol via the 'sword' web application. The version of SWORD v1 currently supported by DSpace is 1.3. The specification and further information can be found at http://swordapp.org.

SWORD is based on the Atom Publish Protocol and allows service documents to be requested which describe the structure of the repository, and packages to be deposited.

1 Enabling SWORD Server
2 Configuring SWORD Server
3 Deposit to SWORD Server
4 DSpace Demo Server: Service Documents

Enabling SWORD Server

To enable DSpace's SWORD v1 server, set the following in your local.cfg:

```
sword-server.enabled = true
# Optionally, if you wish to change its path
sword-server.path = sword
```

Keep in mind, modifying these settings will require restarting your Servlet Container (usually Tomcat).

Once enabled, the SWORD v1 module will be available at ${dspace.server.url}/${sword-server.path}. For example, if "dspace.server.url=http://localhost:8080/server", then (by default) it will be available at http://localhost:8080/server/sword/

Configuring SWORD Server

These are the SWORD (v1) configurations. They may be edited directly or overridden in your local.cfg config (see Configuration Reference).

Configuration File:	**`[dspace]/config/modules/sword-server.cfg`**
Property:	`sword-server.enabled`
Example Value:	`sword-server.enabled = true  (default is false)`
Informational Note:	Whether SWORDv1 module is enabled or disabled (disabled by default). Modifying this setting will require restarting your Servlet Container (usually Tomcat).
Property:	`sword-server.path`
Example Value:	`sword-server.path = sword`
Informational Note:	When enabled, this is the subpath where the SWORDv1 module is deployed. This path is relative to `${dspace.server.url}`. Modifying this setting will require restarting your Servlet Container (usually Tomcat).
Property:	`sword-server.mets-ingester.package-ingester`
Example Value:	`sword-server.mets-ingester.package-ingester = METS`
Informational Note:	The property key tell the SWORD METS implementation which package ingester to use to install deposited content. This should refer to one of the classes configured for: `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, but others are supported. An example of an EPDCX SWORD package can be found at [dspace-src]/dspace-sword/example/example.zip. Additional metadata types can be added to this list by just defining new configurations. For example, you can map a new "mdtype" MYFORMAT to a custom crosswalk named MYFORMAT: `mets.submission.crosswalk.MYFORMAT = MYFORMAT` You'd also want to map your new custom crosswalk to a stylesheet using the next configuration (crosswalk.submission.*.stylesheet).
Property:	`crosswalk.submission.EPDCX.stylesheet` (NOTE: This configuration is in the `dspace.cfg` file)
Example Value:	`crosswalk.submission.EPDCX.stylesheet = crosswalks/sword-swap-ingest.xsl`
Informational Note:	Define the stylesheet which will be used by the self-named XSLTIngestionCrosswalk class when asked to load the SWORD configuration (as specified above). This will use the specified stylesheet to crosswalk the incoming SWAP metadata to the DIM format for ingestion. Additional crosswalk types can be added to this list by just defining new configurations. For example, you can map a custom crosswalk named MYFORMAT to use a specific "my-crosswalk.xsl" stylesheet: `crosswalk.submission.MYFORMAT.stylesheet = crosswalks/my-crosswalk.xsl` Keep in mind, you'll need to also ensure MYFORMAT crosswalk is defined by the previous configuration (mets.submission.crosswalk.*).
Property:	`sword-server.deposit.url`
Example Value:	`sword-server.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 `${dspace.server.url}/${sword-server.path}/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 shown in the example value.
Property:	`sword-server.servicedocument.url`
Example Value:	`sword-server.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 `${dspace.server.url}/${sword-server.path}/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 shown in the example value.
Property:	`sword-server.media-link.url`
Example Value:	`sword-server.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 the media link URLs for items which are deposited via sword. The default is `${dspace.server.url}/${sword-server.path}/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 shown in the example value.
Property:	`sword-server.generator.url`
Example Value:	`sword-server.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 own version. If you are using the standard 'dspace-sword' module you will not, in general, need to change this setting.
Property:	`sword-server.updated.field`

Example Value:	`sword-server.updated.field = dc.date.updated`
Informational Note:	The metadata field in which to store the updated date for items deposited via SWORD.
Property:	`sword-server.slug.field`
Example Value:	`sword-server.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:	`sword-server.accept-packaging.METSDSpaceSIP.identifier` `sword-server.accept-packaging.METSDSpaceSIP.q`
Example Value:	`sword-server.accept-packaging.METSDSpaceSIP.identifier = http://purl.org/net/sword-types/METSDSpaceSIP` `sword-server.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:	`sword-server.accepts`
Example Value:	`sword-server.accepts = application/zip, foo/bar`
Informational Note:	A comma separated list of MIME types that SWORD will accept.
Properties:	`sword-server.accept-packaging.[handle].METSDSpaceSIP.identifier` `sword-server.accept-packaging.[handle].METSDSpaceSIP.q`
Example Value:	`sword-server.accept-packaging.[handle].METSDSpaceSIP.identifier = http://purl.org/net/sword-types/METSDSpaceSIP` `sword-server.accept-packaging.[handle].METSDSpaceSIP.q = 1.0`
Informational Note:	Collection Specific settings: these will be used on the collections with the given handles.
Property:	`sword-server.expose-items`
Example Value:	`sword-server.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 items, which will not be forthcoming for a short while.
Property:	`sword-server.expose-communities`
Example Value:	`sword-server.expose-communities = false`
Informational Note:	Should the server offer as the default the list of all Communities to a Service Document request. If false, the server will offer the list of all collections, which is the default and recommended behavior at this stage. **NOTE:** a service document for Communities will not offer any viable deposit targets, and the client will need to request the list of Collections in the target before deposit can continue.
Property:	`sword-server.max-upload-size`
Example Value:	`sword-server.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, the sword service will default to no limit.
Property:	`sword-server.keep-original-package`
Example Value:	`sword-server.keep-original-package = true`

Informational Note:	Whether or not DSpace should store a copy of the original sword deposit package. **NOTE:** this will cause the deposit process to run slightly slower, and will accelerate the rate at which the repository consumes disk space. BUT, it will also mean that the deposited packages are recoverable in their original form. It is strongly recommended, therefore, to leave this option turned on. When set to "true", this requires that the configuration option `upload.temp.dir` (in `dspace.cfg`) is set to a valid location.
Property:	`sword-server.bundle.name`
Example Value:	`sword-server.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:	`sword-server.keep-package-on-fail` `sword-server.failed-package.dir`
Example Value:	`sword-server.keep-package-on-fail=true` `sword-server.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:	`sword-server.identify-version`
Example Value:	`sword-server.identify-version = true`
Informational Note:	Should the server identify the sword version in a deposit response. It is recommended to leave this unchanged.
Property:	`sword-server.on-behalf-of.enable`
Example Value:	`sword-server.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:	`sword-server.restore-mode.enable`
Example Value:	`sword-server.restore-mode.enable = true`
Informational Note:	Should the sword server enable restore-mode when ingesting new packages. If this is enabled the item will be treated as a previously deleted item from the repository. If the item had previously been assigned a handle then that same handle will be restored to activity. If that item had not been previously assign a handle, then a new handle will be assigned.
Property:	`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 of the format specified above in: `sword-server.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.

Deposit to SWORD Server

If you'd like to deposit content to your repository via the installed SWORD Server, you'll need to select a SWORD Client to do so.

- A variety of SWORDv1 Clients (in various languages/tools) are available off of http://swordapp.org/sword-v1/
- DSpacealso comes with an optional SWORDv1 Client which can be enabled to deposit content from one DSpace to another.
- An example SWORDv1 ZIP package is available in the DSpace Codebase at: https://github.com/DSpace/DSpace/tree/main/dspace-sword/example
- Finally, it's also possible to simply deposit a valid SWORD Zip package via common Linux commandline tools (e.g. curl). For example:

```
# Deposit a SWORD Zip package named "sword-package.zip" into a DSpace Collection (Handle 123456789/2) as
user "test@dspace.org"
# (Please note that you WILL need to obviously modify the Collection location, user/password and name of
the SWORD package)

curl -i --data-binary "@sword-package.zip" -H "Content-Disposition:filename=sword-package.zip" -H
"Content-Type:application/zip" -H "X-Packaging:http://purl.org/net/sword-types/METSDSpaceSIP" -u
test@dspace.org:[password] http://[dspace.url]/sword/deposit/123456789/2

# Template 'curl' command:
#curl -i --data-binary "@[zip-package-name]" -H "Content-Disposition:filename=[zip-package-name]" -H
"Content-Type:application/zip" -H "X-Packaging:http://purl.org/net/sword-types/METSDSpaceSIP" -u [user]:
[password] http://[dspace.url]/sword/deposit/[collection-handle
```

DSpace Demo Server: Service Documents

In DSpace, the SWORD interfaces are on the backend, so the correct URLs are

- https://demo.dspace.org/server/sword/servicedocument
- https://demo.dspace.org/server/swordv2/servicedocument

SWORDv2 Server

SWORD (Simple Web-service Offering Repository Deposit) is a protocol that allows the remote deposit of items into repositories. DSpace implements the SWORD protocol via the 'sword' web application. The specification and further information can be found at http://swordapp.org/.

SWORD is based on the Atom Publish Protocol and allows service documents to be requested which describe the structure of the repository, and packages to be deposited.

1 Enabling SWORD v2 Server
2 Configuring SWORD v2 Server
3 Deposit to SWORDv2 Server
    3.1 Other example SWORDv2 commands
4 Troubleshooting
    4.1 Missing expression of encoding in XML header
5 DSpace Demo Server: Service Documents

Enabling SWORD v2 Server

To enable DSpace's SWORD v2 server, set the following in your local.cfg:

```
swordv2-server.enabled = true
# Optionally, if you wish to change its path
swordv2-server.path = swordv2
```

Keep in mind, modifying these settings will require restarting your Servlet Container (usually Tomcat).

Once enabled, the SWORDv2 module will be available at ${dspace.server.url}/${swordv2-server.path}. For example, if "dspace.server.url=http://localhost:8080/server", then (by default) it will be available at http://localhost:8080/server/swordv2/

Configuring SWORD v2 Server

These are the SWORD (v2) configurations. They may be edited directly or overridden in your local.cfg config (see Configuration Reference).

Configuration File:	**`[dspace]/config/modules/swordv2-server.cfg`**
Property:	`swordv2-server.enabled`
Example Value:	`swordv2-server.enabled = true  (default is false)`
Informational Note:	Whether SWORDv2 module is enabled or disabled (disabled by default). Modifying this setting will require restarting your Servlet Container (usually Tomcat).
Property:	`swordv2-server.path`
Example Value:	`swordv2-server.path = swordv2`
Informational Note:	When enabled, this is the subpath where the SWORDv2 module is deployed. This path is relative to `${dspace.server.url}`. Modifying this setting will require restarting your Servlet Container (usually Tomcat).
Property:	`swordv2-server.url`
Example Value:	`swordv2-server.url = ${dspace.server.url}/${swordv2-server.path}`
Informational Note:	The base url of the SWORD 2.0 system. This defaults to `${dspace.server.url}/${swordv2-server.path}`
Property:	`swordv2-server.collection.url`
Example Value:	`swordv2-server.collection.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 `${dspace.server.url}/${swordv2-server.path}/collection`
Property:	`swordv2-server.servicedocument.url`
Example Value:	`swordv2-server.servicedocument.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 `${dspace.server.url}/${swordv2-server.path}/servicedocument`

Property:	`swordv2-server.accept-packaging.collection`

Example Value:	`swordv2-server.accept-packaging.collection.METSDSpaceSIP = http://purl.org/net/sword/package/METSDSpaceSIP`
	`swordv2-server.accept-packaging.collection.SimpleZip = http://purl.org/net/sword/package/SimpleZip`
	`swordv2-server.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. Package format information • METSDSpaceSIP: zipfile containing mets.xml file describing the resources packed together with it in the root of the zipfile. • Binary: Binary resource that should be taken in as-is, not unpacked • SimpleZip: Zip file that should be unpacked and each file in the zip should be ingested separately. No metadata provided /ingested.
Property:	`swordv2-server.accept-packaging.item`

Example Value:	`swordv2-server.accept-packaging.item.METSDSpaceSIP = http://purl.org/net/sword/package/METSDSpaceSIP`
	`swordv2-server.accept-packaging.item.SimpleZip = http://purl.org/net/sword/package/SimpleZip`
	`swordv2-server.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 collections by adding the handle of the collection to the setting, for example `swordv2-server.accept-packaging.collection.[handle].METSDSpaceSIP = http://purl.org/net/sword-types/METSDSpaceSIP` Package format information • METSDSpaceSIP: zipfile containing mets.xml file describing the resources packed together with it in the root of the zipfile. • Binary: Binary resource that should be taken in as-is, not unpacked • SimpleZip: Zip file that should be unpacked and each file in the zip should be ingested separately. No metadata provided /ingested.
Property:	`swordv2-server.accepts`
Example Value:	`swordv2-server.accepts = application/zip, image/jpeg`
Informational Note:	A comma-separated list of MIME types that SWORD will accept. To accept all mimetypes, the value can be set to "*/*"
Property:	`swordv2-server.expose-communities`
Example Value:	`swordv2-server.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:	`swordv2-server.max-upload-size`
Example Value:	`swordv2-server.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:	`swordv2-server.keep-original-package`

Example Value:	`swordv2-server.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:	`swordv2-server.bundle.name`
Example Value:	`swordv2-server.bundle.name = SWORD`
Informational Note:	The bundle name that SWORD should store incoming packages within if `swordv2-server.keep-original-package` is set to true.
Property:	`swordv2-server.bundle.deleted`
Example Value:	`swordv2-server.bundle.deleted = DELETED`
Informational Note:	The bundle name that SWORD should use to store deleted bitstreams if `swordv2-server.versions.keep` is set to true. This will be used in the case that individual files are updated or removed via SWORD. If the entire Media Resource (files in the ORIGINAL bundle) is removed this will be backed up in its entirety in a bundle of its own
Property:	`swordv2-server.keep-package-on-fail`
Example Value:	`swordv2-server.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 `swordv2-server.failed-package-dir` setting.
Property:	`swordv2-server.failed-package-dir`
Example Value:	`swordv2-server.failed-package-dir = /dspace/upload`
Informational Note:	If `swordv2-server.keep-package-on-fail` is set to true, this is the location where the package would be stored.
Property:	`swordv2-server.on-behalf-of.enable`
Example Value:	`swordv2-server.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:	`swordv2-server.on-behalf-of.update.mediators`
Example Value:	`swordv2-server.on-behalf-of.update.mediators = admin@mydspace.edu, mediator@mydspace.edu`
Informational Note:	Which user accounts are allowed to do updates on items which already exist in DSpace, on-behalf-of other users? If this is left blank, or omitted, then all accounts can mediate updates to items, which could be a security risk, as there is no implicit checking that the authenticated user is a "legitimate" mediator
Property:	`swordv2-server.verbose-description.receipt.enable`

Example Value:	`swordv2-server.verbose-description.receipt.enable = false`
Informational Note:	Should the deposit receipt include a verbose description of the deposit? For use by developers - recommend to set to "false" for production systems
Property:	`swordv2-server.verbose-description.error.enable`
Example Value:	`swordv2-server.verbose-description.error.enable = true`
Informational Note:	should the error document include a verbose description of the error? For use by developers, although you may also wish to leave this set to "true" for production systems
Property:	`swordv2-server.error.alternate.url`
Example Value:	`swordv2-server.error.alternate.url = http://mydspace.edu/contact`
Informational Note:	The error document can contain an alternate url, which the client can use to follow up any issues. For example, this could point to the Contact-Us page
Property:	`swordv2-server.error.alternate.content-type`
Example Value:	`swordv2-server.error.alternate.content-type = text/html`
Informational Note:	The `swordv2-server.error.alternate.url` may have an associated content type, such as `text/html` if it points to a web page. This is used to indicate to the client what content type it can expect if it follows that url.
Property:	`swordv2-server.generator.url`
Example Value:	`swordv2-server.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:	`swordv2-server.generator.version`
Example Value:	`swordv2-server.generator.version = 2.0`
Informational Note:	The version of the SWORD interface.
Property:	`swordv2-server.auth-type`
Example Value:	`swordv2-server.auth-type = Basic`
Informational Note:	Which form of authentication to use. Normally this is set to `Basic` in order to use HTTP Basic.
Property:	`swordv2-server.upload.tempdir`
Example Value:	`swordv2-server.upload.tempd = /dspace/upload`
Informational Note:	The location where uploaded files and packages are stored while being processed.

Property:	`swordv2-server.updated.field`
Example Value:	`swordv2-server.updated.field = dc.date.updated`
Informational Note:	The metadata field in which to store the updated date for items deposited via SWORD.
Property:	`swordv2-server.slug.field`
Example Value:	`swordv2-server.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:	`swordv2-server.author.field`
Example Value:	`swordv2-server.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:	`swordv2-server.title.field`
Example Value:	`swordv2-server.title.field = dc.title`
Informational Note:	The metadata field in which to store the value of the atom entry title if it supplied.
Property:	`swordv2-server.disseminate-packaging`
Example Value:	`swordv2-server.disseminate-packaging.METSDSpaceSIP = http://purl.org/net/sword/package/METSDSpaceSIP` `swordv2-server.disseminate-packaging.SimpleZip = http://purl.org/net/sword/package/SimpleZip`
Informational Note:	Supported packaging formats for the dissemination of packages.
Property:	`swordv2-server.statement.bundles`
Example Value:	`swordv2-server.statement.bundles = ORIGINAL, SWORD, LICENSE`
Informational Note:	Which bundles should the Statement include in its list of aggregated resources? The Statement will automatically mark any bitstreams which are in the bundle identified by the `${bundle.name}` property, provided that bundle is also listed here (i.e. if you want Original Deposits to be listed in the Statement then you should add the SWORD bundle to this list)
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:	`swordv2-server.workflowmanagerdefault.always-update-metadata`
Example Value	`swordv2-server.workflowmanagerdefault.always-update-metadata = true`

Informational Note	Should the WorkflowManagerDefault plugin allow updates to the item's metadata to take place on items which are in states other than the workspace (e.g. in the workflow, archive, or withdrawn) ?
Property:	`swordv2-server.workflowmanagerdefault.file-replace.enable`
Example Value	`swordv2-server.workflowmanagerdefault.file-replace.enable = false`
Informational Note	Should the server allow PUT to individual files? If this is enabled, then DSpace may be used with the DepositMO SWORD extensions, BUT the caveat is that DSpace does not formally support Bitstream replace, so this is equivalent to a DELETE and then a POST, which violates the RESTfulness of the server. The resulting file DOES NOT have the same identifier as the file it was replacing. As such it is STRONGLY RECOMMENDED to leave this option turned off unless working explicitly with DepositMO enabled client environments
Property:	`swordv2-server.mets-ingester.package-ingester`
Example Value:	`swordv2-server.mets-ingester.package-ingester = METS`
Informational Note:	Which package ingester to use for METS packages.
Property:	`swordv2-server.restore-mode.enable`
Example Value:	`swordv2-server.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:	`swordv2-server.simpledc.*`
Example Value:	`swordv2-server.simpledc.abstract = dc.description.abstractsimpledc.date = dc.datesimpledc.rights = dc.rights`
Informational Note:	Configuration of metadata field mapping used by the SimpleDCEntryIngester, SimpleDCEntryDisseminator and FeedContentDisseminator
Property:	`swordv2-server.atom.*`
Example Value	`swordv2-server.atom.author = dc.contributor.author`
Informational Note:	Configuration of metadata field mapping used by the SimpleDCEntryIngester, SimpleDCEntryDisseminator and FeedContentDisseminator
Property:	`swordv2-server.metadata.replaceable`
Example Value	`swordv2-server.metadata.replaceable = dc.description.abstract, dc.rights, dc.title.alternative, dc.identifier.citation`
Informational Note	Used by SimpleDCEntryIngester: Which metadata fields can be replaced during a PUT to the Item of an Atom Entry document? Fields listed here are the ones which will be removed when a new PUT comes through (irrespective of whether there is a new incoming value to replace them)
Property:	`swordv2-server.multipart.entry-first`
Example Value:	`swordv2-server.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:	`swordv2-server.workflow.notify`

Example Value:	`swordv2-server.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:	`swordv2-server.versions.keep`
Example Value:	`swordv2-server.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:	`swordv2-server.state.*`

Example Value:	`swordv2-server.state.workspace.uri = http://dspace.org/state/inprogress`
	`swordv2-server.state.workspace.description = The item is in the user workspace`
	`swordv2-server.state.workflow.uri = http://dspace.org/state/inreview`
	`swordv2-server.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`.
Property:	`swordv2-server.workspace.url-template`
Example Value	`swordv2-server.workspace.url-template = http://mydspace.edu/workspaceitems/#wsid#/edit`
Informational Note	URL template for links to items in the workspace (items in the archive will use the handle). The `#wsid#` url parameter will be replaced with the workspace id of the item. The example above shows how to construct this URL for the UI.

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

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
```

Deposit to SWORDv2 Server

If you'd like to deposit content to your repository via the installed SWORD Server, you'll need to select a SWORD Client to do so.

- Some SWORDv2 Clients are available at http://swordapp.org/sword-v2/
- It's possible to simply deposit a valid SWORDv2 Zip package via common Linux commandline tools (e.g. curl). For example:

```
# Deposit a SWORD Zip package named "sword-package.zip" into a DSpace Collection (Handle 123456789/2) as
user "test@dspace.org"
# (Please note that you WILL need to obviously modify the Collection location, user/password and name of
the SWORD package)

curl -i --data-binary "@sword-package.zip" -H "Content-Disposition:attachment; filename=sword-package.
zip" -H "Content-Type:application/zip" -H "Packaging:http://purl.org/net/sword/package/METSDSpaceSIP" -u
test@dspace.org:[password] -X POST http://[dspace.url]/swordv2/collection/123456789/2

# Template 'curl' command:
#curl -i --data-binary "@[zip-package-name]" -H "Content-Disposition:attachment; filename=[zip-package-
name]" -H "Content-Type:application/zip" -H "Packaging:http://purl.org/net/sword/package/METSDSpaceSIP" -
u [user]:[password] -X POST http://[dspace.url]/swordv2/collection/[collection-handle]
```

  NOTE: `-H "Packaging:http://purl.org/net/sword/package/METSDSpaceSIP"` is required for SWORDv2 and `-H "X-Packaging:http://purl.org/net/sword-types/METSDSpaceSIP"` is required for SWORD. X-Packaging/Packaging and the URLs are different.
- A sample SWORD Zip package (which works for both SWORDv1 or SWORDv2) is available in the codebase at https://github.com/DSpace/DSpace/tree/main/dspace-sword/example. Look for the file named example.zip

Other example SWORDv2 commands

```
# Example of retrieving Item information via "edit-media" path in ATOM format (can be run on any item within
DSpace, but requires authentication)
# NOTE: Accept header is required, and must be a format supported by a SwordContentDisseminator plugin (see
configuration above)
curl -i -H "Accept:application/atom+xml" -u test@dspace.org:[password] -X GET http://[dspace.url]/swordv2/edit-
media/[internal-item-identifier]
```

Troubleshooting

Missing expression of encoding in XML header

If your SWORD Deposit requests are unsuccessful, please check that the XML in your initial metadata deposit correctly specifies the encoding.

If you use:

```
<?xml version="1.0"?>
```

DSpace will default to UTF-32.

So to successfully deposit an XML in UTF-8, make sure you use:

```
<?xml version="1.0" encoding="utf-8" ?>
```

DSpace Demo Server: Service Documents

In DSpace, the SWORD interfaces are on the backend, so the correct URLs are

- https://demo.dspace.org/server/sword/servicedocument
- https://demo.dspace.org/server/swordv2/servicedocument

Ingesting HTML Archives

Not yet supported in DSpace 7. See https://github.com/DSpace/DSpace/issues/8635

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. DSpace allows relative links between HTML documents stored together in a single item to work. 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.

Items and Metadata

- Authority Control of Metadata Values
- Batch Metadata Editing
- DOI Digital Object Identifier
- Item Level Versioning
- Mapping/Linking Items to multiple Collections
- Metadata Recommendations
- Moving Items
- PDF Citation Cover Page
- Request Withdrawn and Reinstate of an item
- Updating Items via Simple Archive Format

Authority Control of Metadata Values

1 Introduction
2 Simple choice management for DSpace submission forms
    2.1 Example
    2.2 Use simple choice management to add language tags to metadata fields
3 Hierarchical Taxonomies and Controlled Vocabularies
    3.1 Default Hierarchical Controlled Vocabularies
    3.2 Enabling / Disabling a Hierarchical Controlled Vocabulary
    3.3 Additional configurations for Hierarchical Controlled Vocabularies
    3.4 How to invoke a controlled vocabulary from submission-forms.xml
4 Authority Control: Enhancing DSpace metadata fields with Authority Keys
    4.1 How it works
    4.2 Original source:

Introduction

With DSpace you can describe digital objects such as text files, audio, video or data to facilitate easy retrieval and high quality search results. These descriptions are organized into metadata fields that each have a specific designation. For example: dc.title stores the title of an object, while dc.subject is reserved for subject keywords.

For many of these fields, including title and abstract, free text entry is the proper choice, as the values are likely to be unique. Other fields are likely to have values drawn from controlled sets. Such fields include unique names, subject keywords, document types and other classifications. For those kinds of fields the overall quality of the repository metadata increases if values with the same meaning are normalized across all items. Additional benefits can be gained if unique identifiers are associated as well in addition to canonical text values associated with a particular metadata field.

This page covers features included in the DSpace submission forms that allow repository managers to enforce the usage of normalized terms for those fields where this is required in their institutional use cases. DSpace offers simple and straightforward features, such as definitions of simple text values for dropdowns, as well as more elaborate integrations with external vocabularies such as the Library of Congress Naming Authority.

Simple choice management for DSpace submission forms

The DSpace Submission forms, defined in the submission-forms.xml file, allows the inclusion of value pairs that can be organized in lists in order to populate dropdowns or other multiple choice elements. If you explore the default submission-forms.xml file, you can see that a number of such value pair lists are already pre defined.

Example

```
<value-pairs value-pairs-name="common_identifiers" dc-term="identifier">
    <pair>
        <displayed-value>Gov't Doc #</displayed-value>
        <stored-value>govdoc</stored-value>
    </pair>
    <pair>
        <displayed-value>URI</displayed-value>
        <stored-value>uri</stored-value>
    </pair>
    <pair>
        <displayed-value>ISBN</displayed-value>
        <stored-value>isbn</stored-value>
    </pair>
</value-pairs>
```

It generates the following HTML, which results in the menu widget below.

```
<select name="identifier_qualifier_0">
    <option VALUE="govdoc">Gov't Doc #</option>
    <option VALUE="uri">URI</option>
    <option VALUE="isbn">ISBN</option>
</select>
```

A list of value pairs has following required attributes:

- **value-pairs-name** – Name by which an *input-type* refers to this list.
- **dc-term** – Dublin Core field for which this choice list is selecting a value.

Each *value-pairs* element contains a sequence of *pair* sub-elements, each of which in turn contains two elements:

- **displayed-value** – Name shown (on the web page) for the menu entry.
- **stored-value** – Value stored in the DC element when this entry is chosen. Unlike the HTML *select* tag, there is no way to indicate one of the entries should be the default, so the first entry is always the default choice.

Use simple choice management to add language tags to metadata fields

DSpace uses the simple choice management to provide a controlled list of language tags. Out of the box DSpace comes with a list of ISO language tags. You can add further language lists or use the provided one to let submitters tag languages of metadata fields. Take a look at the part of this documentation about the configuration of the Submission User Interface.

Hierarchical Taxonomies and Controlled Vocabularies

The value pairs system works well for short and flat lists of choices. DSpace offers a second way of structuring and managing more complex, hierarchical controlled vocabularies. In contrast to the value pairs system, these controlled vocabularies are managed in separate XML files in the `[dspace]/config/controlled-vocabularies/` directory instead of being entered straight into submission-forms.xml

The taxonomies are described in XML according to this structure:

```
<node id="acmccs98" label="ACMCCS98">
    <isComposedBy>
        <node id="A." label="General Literature">
            <isComposedBy>
                <node id="A.0" label="GENERAL"/>
                <node id="A.1" label="INTRODUCTORY AND SURVEY"/>
                ...
            </isComposedBy>
        </node>
        ...
    </isComposedBy>
</node>
```

As you can see, each node element has an id and label attribute. It can contain the isComposedBy element, which in its turn, consists of a list of other nodes.

You are free to use any application you want to create your controlled vocabularies. A simple text editor should be enough for small projects. Bigger projects will require more complex tools. You may use Protegé to create your taxonomies, save them as OWL and then use a XML Stylesheet (XSLT) to transform your documents to the appropriate format. Future enhancements to this add-on should make it compatible with standard schemas such as OWL or RDF.

Default Hierarchical Controlled Vocabularies

By default, DSpace includes two out-of-the-box hierarchical controlled vocabularies in the `[dspace]/config/controlled-vocabularies/` directory.

- **nsi** - *nsi.xml* - The Norwegian Science Index (in the Norweigen language)
- **srsc** - *srsc.xml* - Swedish Research Subject Categories (in the English language, with notes in Swedish)

You may create your own hierarchical controlled vocabulary by using either of those as a model. All valid hierarchical vocabularies should align with the "controlledvocabulary.xsd" schedule available in that same directory.

Enabling / Disabling a Hierarchical Controlled Vocabulary

To enable a hierarchical controlled vocabulary, simply configure it's usage in one (or more) of your fields in your "submission-forms.xml" (as documented below).

To disable a hierarchical controlled vocabulary, simply remove it from all your fields in your "submission-forms.xml". You can also disable **all** controlled vocabularies by commenting out the "DSpaceControlledVocabulary" plugin in "authority.cfg":

authority.cfg

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

It's also possible to disable just the "Browse by" menu for a specific vocabulary in the user interface. For details on that, see the "`webui.browse.vocabularies.disabled`" setting in the dspace.cfg documented at Configuration Reference#HierarchicalBrowseIndexes

Additional configurations for Hierarchical Controlled Vocabularies

Each hierarchical controlled vocabulary supports additional configurations in the "authority.cfg" file (or local.cfg). These settings override the default behavior of a Controlled Vocabulary in the "org.dspace.content.authority.DSpaceControlledVocabulary" Java class.

Property:	`vocabulary.plugin.[vocab-name].hierarchy.store`
Example Value:	`vocabulary.plugin.srsc.hierarchy.store = true`
Informational Note:	Store entire hierarchy along with selected value. Default is true
Property:	`vocabulary.plugin.[vocab-name].hierarchy.suggest`
Example Value:	`vocabulary.plugin.srsc.hierarchy.suggest = false`
Informational Note:	Display the entire hierarchy in the suggestion list. Default is false.
Property:	`vocabulary.plugin.[vocab-name].hierarchy.delimiter`
Example Value:	`vocabulary.plugin.srsc.hierarchy.delimiter = "::"`
Informational Note:	Delimiter to use when building hierarchy string. Default is "::" (double colon)
Property:	`vocabulary.plugin.[vocab-name].hierarchy.preloadLevel`
Example Value:	`vocabulary.plugin.srsc.hierarchy.preloadLevel = 1`
Informational Note:	The number of levels of the hierarchy to load at once (e.g. in the "Browse by [vocab-name]" in the user interface). The value "1" means to just load the top level members of the hierarchy. The value "2" means to load the top level members and their children. The value "3" means to load the top level members, their children and their grandchildren. Keep in mind, the "Browse by" options in the User Interface may be slower when you load more levels at once. Default is "1".

How to invoke a controlled vocabulary from submission-forms.xml

Vocabularies need to be associated with the correspondent DC metadata fields. Edit the file `[dspace]/config/submission-forms.xml` and place a "`vocabulary`" tag under the "`field`" element that you want to control. Set value of the "`vocabulary`" element to the name of the file that contains the vocabulary, leaving out the extension (the add-on will only load files with extension "*.xml"). For example:

```
<field>
    <dc-schema>dc</dc-schema>
    <dc-element>subject</dc-element>
    <dc-qualifier></dc-qualifier>
    <repeatable>true</repeatable>
    <label>Subject Keywords</label>
    <input-type>onebox</input-type>
    <hint>Enter appropriate subject keywords or phrases below.</hint>
    <required></required>
    <vocabulary>srsc</vocabulary>
</field>
```

The vocabulary element has an optional boolean attribute `closed` that can be used to force input only with the Javascript of controlled-vocabulary add-on. The default behaviour (i.e. without this attribute) is `closed="false"`. This allows the user to enter values as free text in addition to selecting them from the controlled vocabulary.

Authority Control: Enhancing DSpace metadata fields with Authority Keys

The aforementioned features only deal with text representations of controlled values. DSpace also offers support for adding authority keys and confidence values to a specific text value entered in a metadata field. The following terminology applies in the description of this area of DSpace functionality:

- **Authority** An *authority* is an external source of fixed values for a given domain, each unique value identified by a *key*. For example, the OCLC LC Name Authority Service, ORCID or VIAF.
- **Authority Record** The information associated with one of the values in an authority; may include alternate spellings and equivalent forms of the value, etc.
- **Authority Key** An opaque, hopefully persistent, identifier corresponding to exactly one record in the authority.

The fact that this functionality deals with **external** sources of authority makes it inherently different from the functionality for controlled vocabularies. Another difference is that the authority control is asserted *everywhere* metadata values are changed, including unattended/batch submission, SWORD package submission, and the administrative UI.

How it works

TODO

Original source:

Authority Control of Metadata Values original development proposal for DSpace 1.6

ORCID Authority

1 Introduction
2 Use case and high level benefits
3 Enabling the ORCID authority control
    3.1 Settings to enable in local.cfg
    3.2 Enabling the ORCID beans
4 Importing existing authors & keeping the index up to date
    4.1 Different possible use cases for Index-authority script
        4.1.1 Metadata value WITHOUT authority key in metadata
        4.1.2 Metadata that already has an authority key from an external source (NOT auto-generated by DSpace)
        4.1.3 Metadata that has already a new dspace generated uid authority key
        4.1.4 Processing on records in the authority cache
    4.2 Submission of new DSpace items - Author lookup
    4.3 Admin Edit Item
    4.4 Editing existing items using Batch CSV Editing
    4.5 Storage of related metadata
5 Configuration
6 Adding additional fields under ORCID
7 Integration with other systems beside ORCID
8 FAQ
    8.1 Which information from ORCID is currently indexed in the authority cache?
    8.2 How can I index additional fields in the authority cache?
    8.3 How can I use the information stored in the authority cache?
    8.4 How to add additional metadata fields in the authority cache that are not related to ORCID?
    8.5 What happens to data if another authority control was already present?
    8.6 Where can I find the URL that is used to lookup ORCIDs?
    8.7 How can I export all the authority records that have an ORCID?

ORCID Authority can only pull data from ORCID and link it to a DSpace metadata field

ORCID Authority allows you to link up DSpace metadata fields (added during the submission process) to a person's ORCID identifier. The main use case for this feature is to allow you to link author metadata fields to their ORCID identifier. This is a very basic ORCID integration that has existed since DSpace 5.x.

If you re looking for ORCID Authentication & the ability to synchronize data from DSpace to an ORCID profile, then you should be using the ORCID Integration feature instead.

Introduction

The ORCID integration adds ORCID compatibility to the existing solutions for Authority control in DSpace. String names of authors are still being stored in DSpace metadata. The authority key field is leveraged to store a uniquely generated internal ID that links the author to more extended metadata, including the ORCID ID and alternative author names.

This extended metadata is stored and managed in a dedicated SOLR index, the DSpace authority cache.

Use case and high level benefits

The vision behind this project consists of the following two aspects:

Lowering the threshold to adopt ORCID for the members of the DSpace community

ORCID's API has enabled developers across the globe to build points of integration between ORCID and third party applications. Up until today, this meant that members of the DSpace community were still required to implement front-end and back-end modifications to the DSpace source code in order to leverage these APIs. As DSpace aims to provide turnkey Institutional Repository functionality, the platform is expected to provide more functionality out of the box. Only an elite selection of members in the DSpace community has software development resources readily available to implement this kind of functionality. By contributing a solution directly to the core DSpace codebase, this threshold to adopt ORCID functionality in DSpace repositories is effectively lowered. The ultimate goal is to allow easy adoption of ORCID without customization of the DSpace software, by allowing repository administrators to enable or disable functionality by means of user friendly configuration.

Address generic use cases with appealing end user functionality

This proposal aims to provide user friendly features for both repository administrators as well as non- technical end users of the system. The addition of ORCID functionality to DSpace should not come at the cost of making the system more difficult for administrators and end users to use. Scope With this vision in mind, the project partners wanted to tackle the first phases for repository managers of existing DSpace repositories: ensuring that ORCIDs are properly associated with new works entering the system, as well as providing functionality to efficiently batch-update content already existing in the system, with unambiguous author identity information.

Enabling the ORCID authority control

To enable ORCID authority control requires settings in both local.cfg and updating the registered beans in "orcid-authority-services.xml" as described below.

Settings to enable in local.cfg

If you wish to enable this feature, some changes are required to the `local.cfg` file. The first step is to activate the authority as a valid option for authority control, this is done by adding/setting an additional plugin in the `plugin.named.org.dspace.content.authority.ChoiceAuthority` property. An example of this can be found below.

authority.cfg

```
plugin.named.org.dspace.content.authority.ChoiceAuthority = \
    org.dspace.content.authority.SolrAuthority = SolrAuthorAuthority
```

The feature relies on the following configuration parameters in `authority.cfg, solrauthority.cfg and orcid.cfg`. To activate the default settings it suffices to remove the comment hashes ("#") for the following lines or copy them into your local.cfg. See the section at the bottom of this page what these parameters mean exactly and how you can tweak the configuration.

```
# This setting should already be specified in your solrauthority.cfg
solr.authority.server=${solr.server}/${solr.multicorePrefix}authority

# These settings can be found in your authority.cfg (or could be added to local.cfg)
choices.plugin.dc.contributor.author = SolrAuthorAuthority
choices.presentation.dc.contributor.author = authorLookup
authority.controlled.dc.contributor.author = true
authority.author.indexer.field.1=dc.contributor.author
```

Beginning with DSpace 7, you must specify which ORCID API you wish to use. A Client ID/Secret is also required, but can be obtained for free for the Public API: https://info.orcid.org/documentation/features/public-api/

If you are an ORCID Member Institution, you can use the Member API instead. The Member API is required for additional ORCID Integration features, but is NOT required for this basic ORCID Authority feature.

```
# These ORCID settings are now required for ORCID Authority
# They can be found in your orcid.cfg (or can be added to local.cfg)
orcid.domain-url = https://orcid.org
# You can use either the Public API or Member API in this next setting
orcid.api-url = https://pub.orcid.org/v3.0

# You do NOT need to pay for a Member API ID to use ORCID Authority.
# Instead, you just need a Public API ID from a free ORCID account.
orcid.application-client-id = MYID
orcid.application-client-secret = MYSECRET
```

The final part of configuration is to add the authority consumer in front of the list of event consumers (in dspace.cfg or local.cfg). Add "authority" in front of the list as displayed below.

```
event.dispatcher.default.consumers = authority, versioning, discovery, eperson
```

Enabling the ORCID beans

You must also uncomment the ORCID beans in `config/spring/api/orcid-authority-services.xml`. These are commented out by default as they require setting the "orcid.*" settings described above.

Simply uncomment these settings as-is & restart Tomcat. They will pull their configs from orcid.cfg or your local.cfg.

orcid-authority-services.xml

```
<!-- This bean & alias are commented out by default. Simply uncomment them -->
<alias name="OrcidSource" alias="AuthoritySource"/>
<bean name="OrcidSource" class="org.dspace.authority.orcid.Orcidv3SolrAuthorityImpl">
    <property name="clientId" value="${orcid.application-client-id}" />
    <property name="clientSecret" value="${orcid.application-client-secret}" />
    <property name="OAUTHUrl" value="${orcid.token-url}" />
    <property name="orcidRestConnector" ref="orcidRestConnector"/>
</bean>

<!-- Also uncomment Orcidv3AuthorityValue in the list of supported types below.
     The other settings in this AuthorityTypes bean can be left as-is. -->
<bean name="AuthorityTypes" class="org.dspace.authority.AuthorityTypes">
        <property name="types">
            <list>
                <bean class="org.dspace.authority.orcid.Orcidv3AuthorityValue"/>
                <bean class="org.dspace.authority.PersonAuthorityValue"/>
            </list>
        </property>
...
</bean>
```

Importing existing authors & keeping the index up to date

When first enabled the authority index will be empty, to populate the authority index run the following script:

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

This will iterate over every metadata under authority control and create records of them in the authority index. The metadata without an authority key will each be updated with an auto generated authority key. These will not be matched in any way with other existing records. The metadata with an authority key that does not already exist in the index will be indexed with those authority keys. The metadata with an authority key that already exist in the index will be re-indexed the same way. These records remain unchanged.

Different possible use cases for Index-authority script

Metadata value WITHOUT authority key in metadata

“Luyten, Bram” is present in the metadata without any authority key.
GOAL: “Luyten, Bram” gets added in the cache ONCE

All occurences of “Luyten, Bram” in the DSpace item metadata will become linked with the same generated uid.

Metadata that already has an authority key from an external source (NOT auto-generated by DSpace)

“Snyers, Antoine” is present with authority key “u12345”

The old authority key needs to be preserved in the item metadata and duplicated in the cache.
“u12345” will be copied to the authority cache and used as the authority key there.

Metadata that has already a new dspace generated uid authority key

Item metadata already contains an author with name “Haak, Danielle” and a uid in the authority field 3dda2571-6be8-4102-a47b-5748531ae286

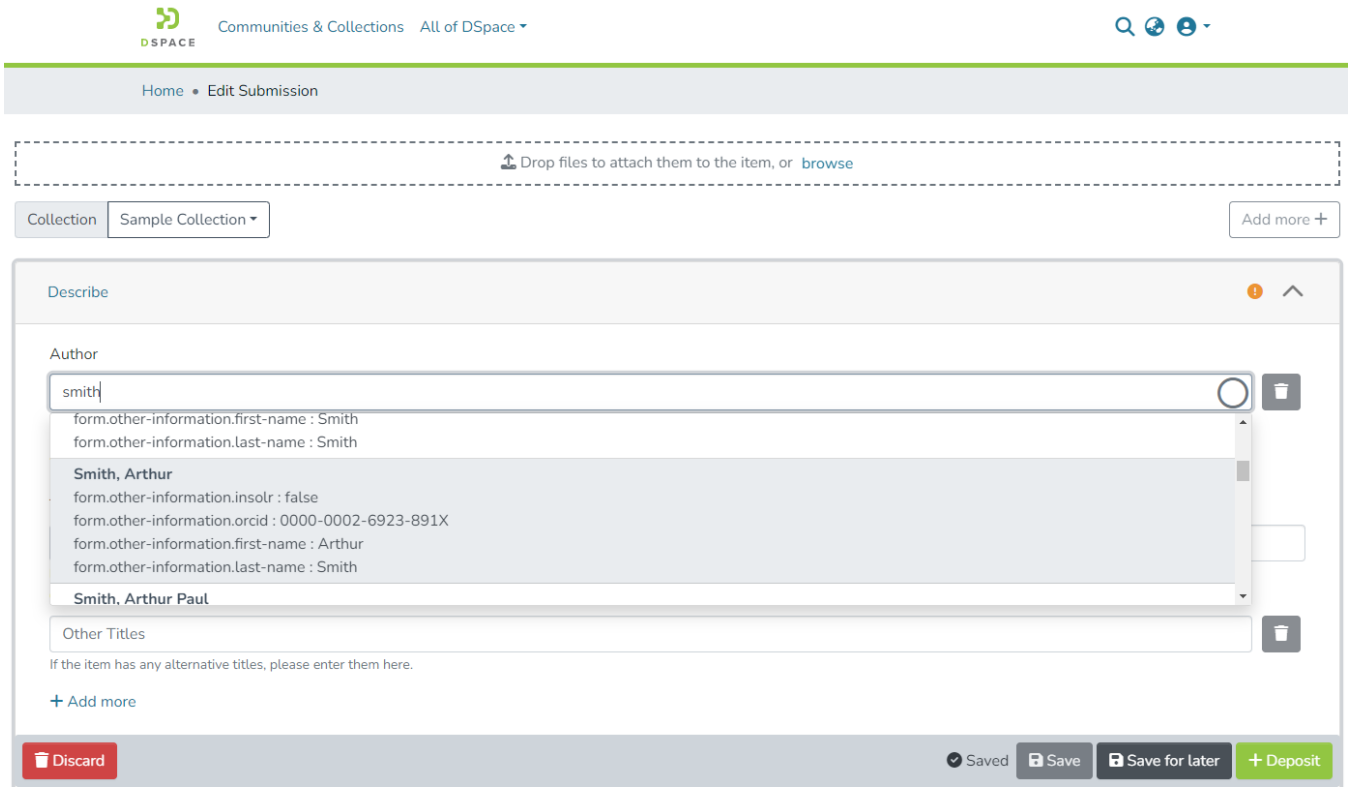
This uid is preserved and no new record is being created in the authority index.

Processing on records in the authority cache

Running this script again will update the index and keep the index clean. For example if an author occurs in a single item and that item is deleted the script will need to be run again to remove it from the index. When run again it will remove all records that no longer have a link to existing authors in the database.

Submission of new DSpace items - Author lookup

When ORCID Authority is enabled, the Author field can be used to search entries in ORCID. Simply type in an Author name to search your locally indexed authors and authors in ORCID.



Select an author entry from the list to add that Author. The List of authors is updated as you type.

Authors that already appear somewhere in the repository are differentiated from the authors that have been retrieved from ORCID.

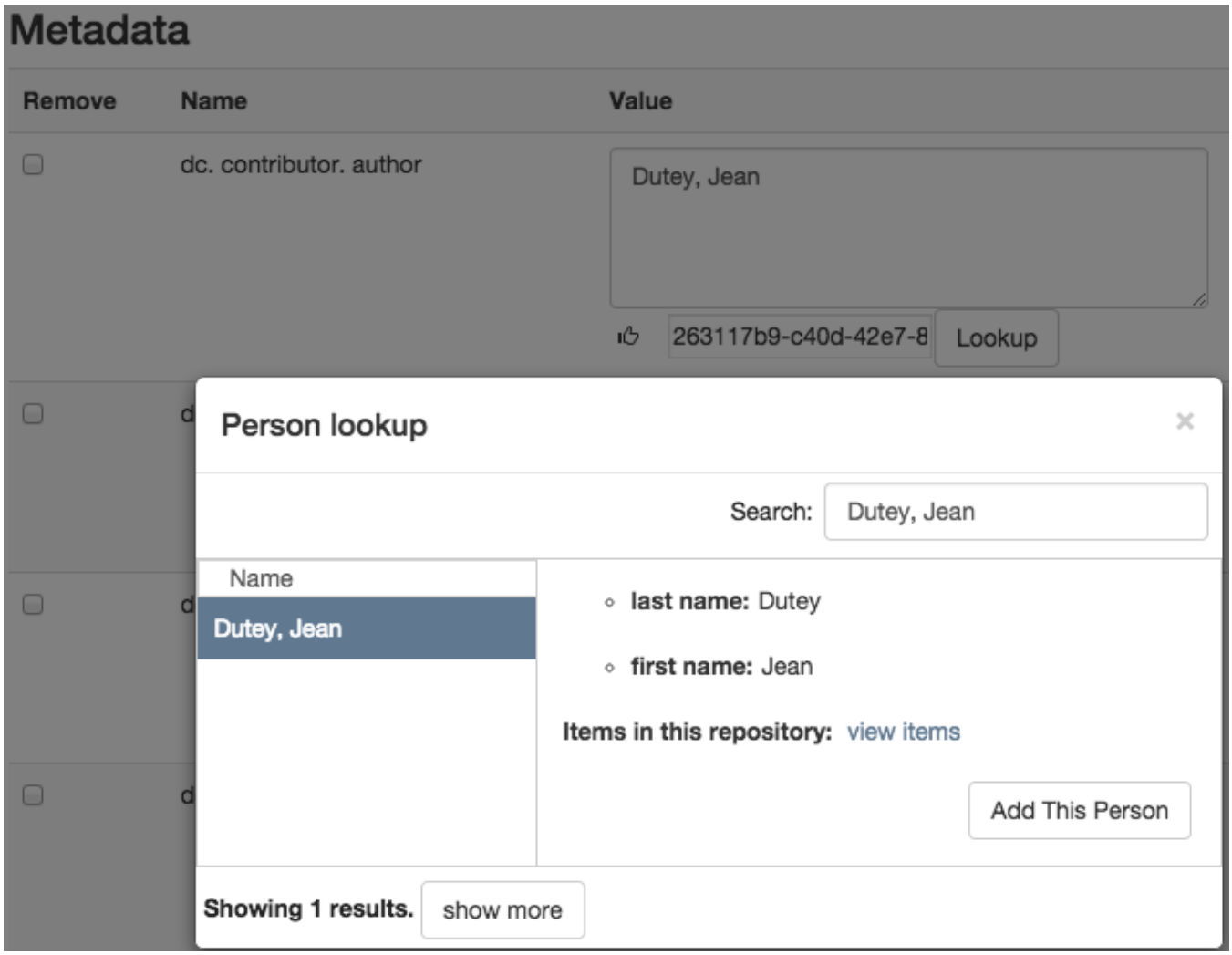
Admin Edit Item

Not yet available in DSpace 7.x. The below screenshots are from 6.x

In the edit metadata page, under the values for the dc.contributor.author fields, an extra line shows the author ID together with a lock icon. The input field

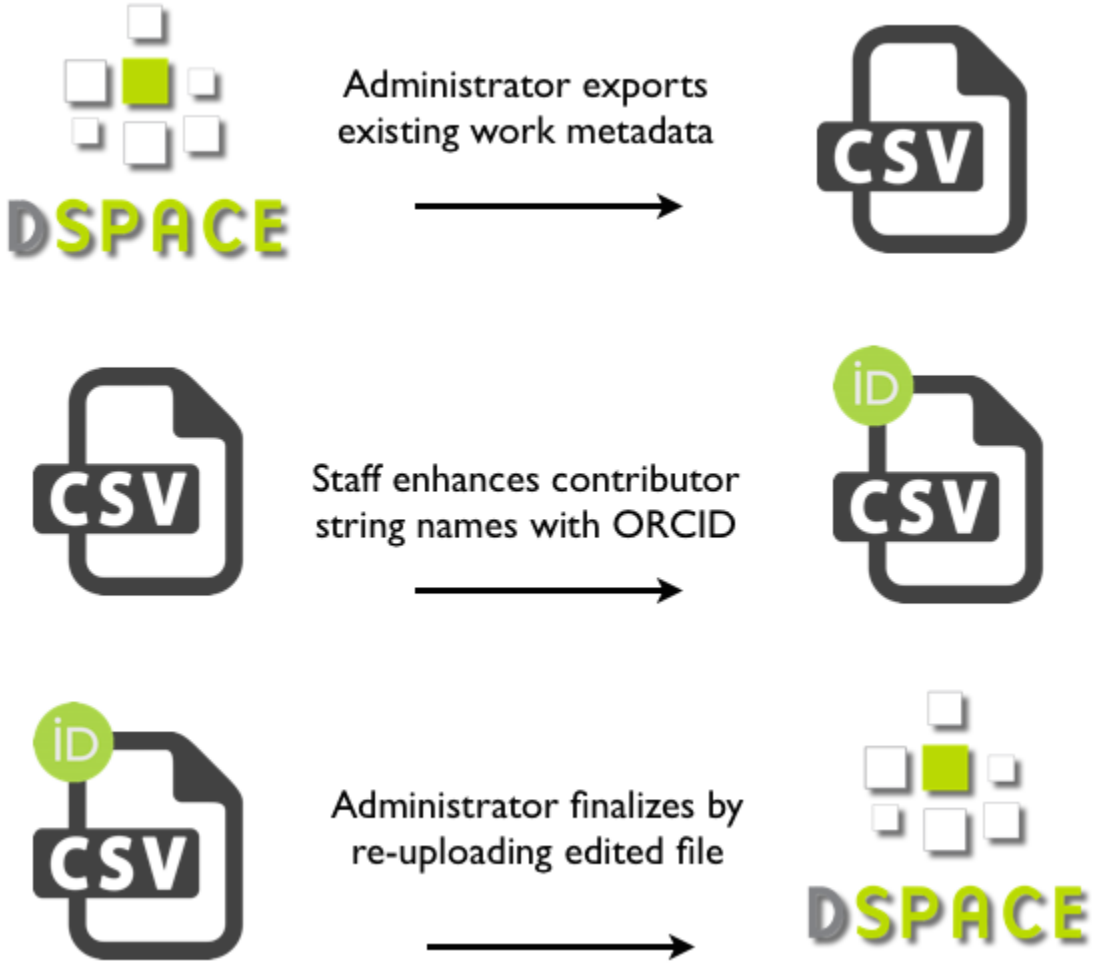
The author ID cannot be changed manually. However the Lookup button will help you change the author name and ID at the same time.

Clicking the Lookup button brings back the Lookup User Interface. This works just the same way as in the submission forms.



Editing existing items using Batch CSV Editing

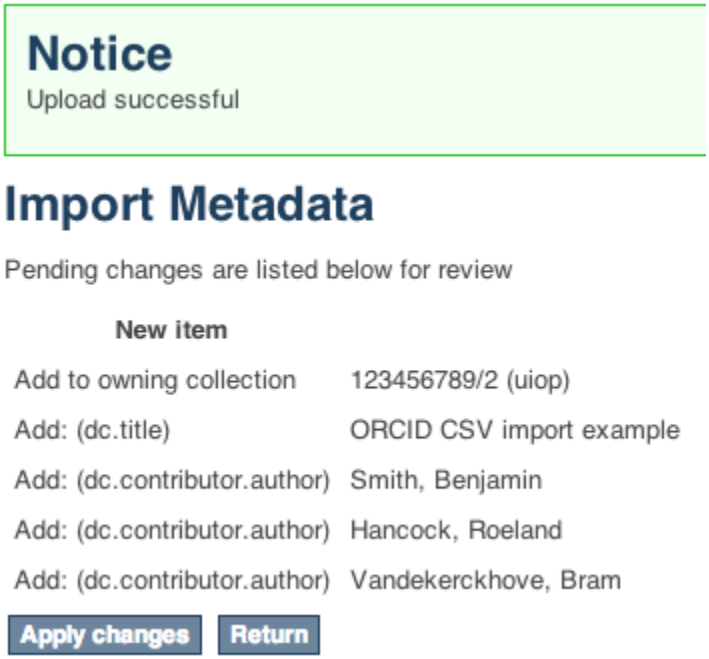
Instructions on how to use the Batch CSV Editing are found on the Batch Metadata Editing documentation page.



ORCID Integration is provided through the Batch CSV Editing feature with an extra available headers "ORCID:dc.contributor.author". The usual CSV headers only contain the metadata fields: e.g. "dc.contributor.author". In addition to the traditional header, another dc.contributor.author header can be added with the "ORCID:" prefix. The values in this column are supposed to be ORCIDs.

id	collection	dc.title	ORCID:dc.contributor.author	dc.contributor.author
+	123456789/2	ORCID CSV import example	0000-0001-8932-6872||0000-0001-9152-849X	Smith, Benjamin

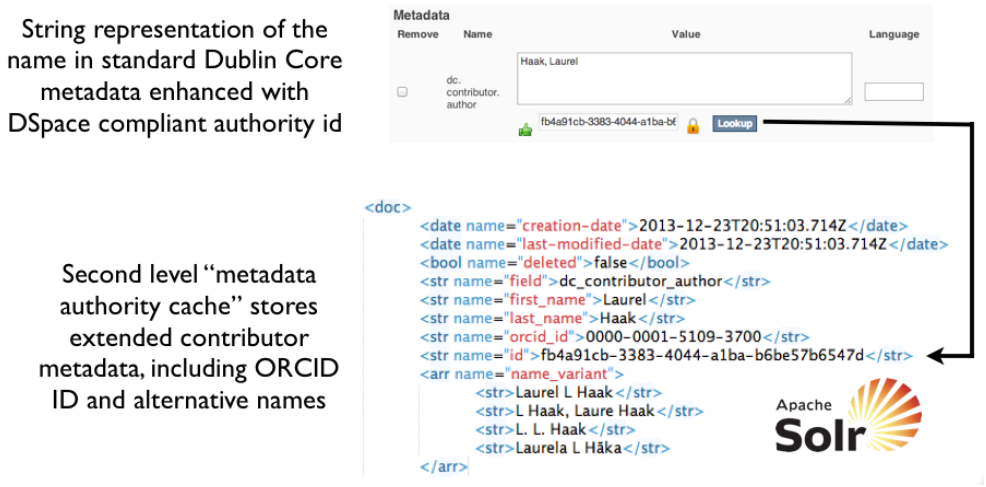
For each of the ORCID authors a lookup will be done and their names will be added to the metadata. All the non-ORCID authors will be added as well. The authority keys and solr records are added when the reported changes are applied.



Storage of related metadata

ORCID authorities not only link a digital identifier to a name. It regroups a load of metadata going from alternative names and email addresses to keywords about their works and much more. The metadata is obtained by querying the ORCID web services. In order to avoid querying the ORCID web services every time, all these related metadata is gathered in a "metadata authority cache" that DSpace can access directly.

In practice the cache is provided by an apache solr server. When a look-up is made and an author is chosen that is not yet in the cache, a record is created from an ORCID profile and added to the cache with the list of related metadata. The value of the Dublin Core metadata is based on the first and last name as they are set in the ORCID profile. The authority key for this value links directly to the solr document's id. DSpace does not provide a way to edit these records manually.



The information in the authority cache can be updated by running the following command line operation:

Command used:	`[dspace]/bin/dspace dsrun org.dspace.authority.UpdateAuthorities`
Arguments	description
-i	update specific solr records with the given internal ids (comma-separated)

-h	prints this help message

This will iterate over every solr record currently in use (unless the -i argument is provided), query the ORCID web service for the latest data and update the information in the cache. If configured, the script will also update the metadata of the items in the repository where applicable.

The configuration property can be set in `config/modules/solrauthority.cfg`, or overridden in your `local.cfg` (see Configuration Reference).

```
solrauthority.auto-update-items = false | true
```

When set to true and this is script is run, if an authority record's information is updated the whole repository will be scanned for this authority. Every metadata field with this authority key will be updated with the value of the updated authority record.

Configuration

In the Enabling the ORCID authority control section, you have been told to add this block of configuration.

NOTE: you can use local.cfg for these

For all of the configuration options described below, you can use either dspace.cfg or local.cfg. Either will work. It is possible that, when you compile your code with Maven, and you have tests enabled, your build will fail. DSpace unit tests utilize parts of dspace.cfg, and the configuration options you will utilize below are known to cause unit test errors. The easiest way to avoid this situation is to use the local.cfg file.

```
solr.authority.server=${solr.server}/authority
choices.plugin.dc.contributor.author = SolrAuthorAuthority
choices.presentation.dc.contributor.author = authorLookup
authority.controlled.dc.contributor.author = true
authority.author.indexer.field.1=dc.contributor.author

# These ORCID settings are now required for ORCID Authority
orcid.domain-url = https://orcid.org
# You can use either the Public API or Member API
orcid.api-url = https://pub.orcid.org/v3.0

# You do NOT need to pay for a Member API ID to use ORCID Authority.
# Instead, you just need a Public API ID from a free ORCID account.
# https://info.orcid.org/documentation/features/public-api/
orcid.application-client-id = MYID
orcid.application-client-secret = MYSECRET
```

The ORCID Integration feature is an extension on the authority control in DSpace. Most of these properties are extensively explained on the Authority Control of Metadata Values documentation page. These will be revisited but first we cover the properties that have been newly added.

- The **`solr.authority.server`** is the url to the solr core. Usually this would be on the **`solr.server`** next to the oai, search and statistics cores.
- **`authority.author.indexer.field.1`** and the subsequent increments configure which fields will be indexed in the authority cache. However before adding extra fields into the solr cache, please read the section about Adding additional fields under ORCID.

That's it for the novelties. Moving on to the generic authority control properties:

- With the **`authority.controlled`** property every metadata field that needs to be authority controlled is configured. This involves every type of authority control, not only the fields for ORCID integration.
- The **`choices.plugin`** should be configured for each metadata field under authority control. Setting the value on SolrAuthorAuthority tells DSpace to use the solr authority cache for this metadatafield, cfr. Storage of related metadata.
- The **`choices.presention`** should be configured for each metadata field as well. The traditional values for this property are `select|suggest|lookup`. A new value, `authorLookup`, has been added to be used in combination with the SolrAuthorAuthority choices plugin. While the other values can still be used, the authorLookup provides a richer user interface in the form of a popup on the submission page.
- The browse indexes need to point to the new authority-controlled index: `webui.browse.index.2 = author:metadata:dc.contributor.*,dc.creator:text` should become **webui.browse.index.2 = author:metadataAuthority:dc.contributor.author:authority**
- More existing configuration properties are available but their values are independent of this feature and their default values are usually fine: **`choices.closed`** , **`authority.required`**, **`authority.minconfidence`** .

For the cache update script, one property can be set in `config/modules/solrauthority.cfg`:

```
auto-update-items = false | true
```

The default value for when the property is missing is false.

The final part of configuration is to add the authority consumer in front of the list of event consumers. Add "authority" in front of the list as displayed below.

```
event.dispatcher.default.consumers = authority, versioning, discovery, eperson, qaeventsdelete, ldnmessage
```

Without the consumer there is no automatic indexing of the authority cache and the metadata will not even have authority keys.

Changes to the configuration always require a server restart before they're in effect.

Adding additional fields under ORCID

Other metadata fields besides "dc.contributor.author" can benefit from the ORCID authority control at the same time. Here is an example of how to get the same ORCID functionality for the "dc.contributor.editor" metadata field assuming that "dc.contributor.author" is already configured correctly. It can be achieved by modifying configuration files only.

First add the same configuration fields that have been added for the "dc.contributor.author"

```
choices.plugin.dc.contributor.editor = SolrAuthorAuthority
choices.presentation.dc.contributor.editor = authorLookup
authority.controlled.dc.contributor.editor = true

authority.author.indexer.field.1=dc.contributor.author
authority.author.indexer.field.2=dc.contributor.editor
```

This is enough to get the look-up interface on the submission page and on the edit metadata page. The authority keys will be added and indexed with the information from orcid just as it happens with the Authors.

But you're not completely done yet, There is one more configuration step. Because now when adding new editors in the metadata that are not retrieved through the external look-up, their first and last name will not be displayed in the look-up interface next time you look for them.

To fix this, open the file at `config/spring/api/orcid-authority-services.xml` and find this spring bean:

```
<bean name="AuthorityTypes" class="org.dspace.authority.AuthorityTypes">
    <property name="types">
        <list>
            <bean class="org.dspace.authority.orcid.Orcidv3AuthorityValue"/>
            <bean class="org.dspace.authority.PersonAuthorityValue"/>
        </list>
    </property>
    <property name="fieldDefaults">
        <map>
            <entry key="dc_contributor_author">
                <bean class="org.dspace.authority.PersonAuthorityValue"/>
            </entry>
        </map>
    </property>
</bean>
```

The map inside the "fieldDefaults" property needs an additional entry for the editor field:

```
<entry key="dc_contributor_editor">
    <bean class="org.dspace.authority.PersonAuthorityValue"/>
</entry>
```

With this last change everything is set up to work correctly. The rest of this configuration file is meant for JAVA developers that wish to provide integration with other systems beside ORCID. Developers that wish to display other fields than first and last name can also have a look in that section.

Note: Each metadata field has a separate set of authority records. Authority keys are not shared between different metadata fields. E. g. multiple dc.contributor.author can have the same authority key and point to the same authority record in the cache. But when an ORCID is chosen for a dc.contributor.editor field, a separate record is made in the cache. Both records are updated from the same source and will contain the same information. The difference is that when performing a look-up of a person that has been introduced as an authority for an author field but not yet as an editor, it will show as record that is not yet present in the repository cache.

Integration with other systems beside ORCID

The authority cache and look-up functionality can be extended to use other sources than ORCID or to show more information in the look-up interface. However some JAVA development is necessary for this. Specific instructions can be found in the readme file of the org.dspace.authority package.

FAQ

Which information from ORCID is currently indexed in the authority cache?

Here is a breakdown of the fields stored in the solr cache.

The system/dspace related fields are: *id, field, value, deleted, creation_date, last_modified_date, authority_type*.

The fields with data coming directly from ORCID are: *first_name, last_name, name_variant, orcid_id, label_researcher_url, label_keyword, label_external_identifier, label_biography, label_country.* The field *all_labels* contains all the values from the other fields starting with "label_".

How can I index additional fields in the authority cache?

There is currently no configuration to control which fields are indexed. The only way to achieve this is to modify the source code.

List of the files at work for this job:
`config/spring/api/orcid-authority-services.xml`: OrcidSource contains the URL for orcid's REST API.
org.dspace.authority.orcid.Orcid makes the REST call
+ org.dspace.authority.orcid.xml.XMLtoBio converts the received XML to a java object (Bio).
+ org.dspace.authority.orcid.model.Bio
+ org.dspace.authority.orcid.OrcidAuthorityValue#create(org.dspace.authority.orcid.model.Bio) inserts all the values from Bio into the AuthorityValue subclass.
+ org.dspace.authority.orcid.OrcidAuthorityValue#getSolrInputDocument defines what's included in solr.

The files preceded with a '+' would be necessary to modify to add more info into the cache.

How can I use the information stored in the authority cache?

The look-up UI is currently the only place this information is sent to. However just a limited number of fields are sent. The place in the source code to modify to get more fields there is org.dspace.authority.orcid.OrcidAuthorityValue#choiceSelectMap. This is also documented in the readme of the org.dspace.authority package.

You can eventually query and export the information directly from the SOLR authority core if needed (see the FAQ belows)

How to add additional metadata fields in the authority cache that are not related to ORCID?

Make the same configuration step as for adding additional fields under ORCID. Currently the ORCID suggestions cannot be turned off for specific fields, that would require custom code.

What happens to data if another authority control was already present?

As long as the metadata does not get indexed, there will be no changes. However every time any metadata of an item is modified, the metadata under authority control for that item will be re-indexed. When that happens a record will be inserted in the solr cache. That record's ID will be the authority key of the metadata. This can be done for all metadata at once with the index-authority script.

In short: authority keys that exist prior to enabling the solrauthority are kept. They just won't show in the look-up until they are indexed.

Where can I find the URL that is used to lookup ORCIDs?

It is found in the `config/spring/api/orcid-authority-services.xml` configuration file. Look for the `<bean name="OrcidSource">`, which is initialized with a URL of http://pub.orcid.org

How can I export all the authority records that have an ORCID?

Please note that all the links below assume that you are running solr on localhost and you use the standard core names without prefix.

Authority records with an ORCID are the solr document with a `orcid_id` field.

The following solr query will identify such records

```
orcid_id:*
```

http://localhost:8983/solr/authority/select?indent=true&q.op=OR&q=orcid_id%3A*

to get only the SOLR Authority records used by some DSpace items, you need to perform a `join search` in SOLR (https://solr.apache.org/guide/solr/latest/query-guide/join-query-parser.html) between the authority and the search core. This is a sample query that you can customize to include additional metadata other than `dc.contributor.author`

```
{!join from=dc.contributor.author_authority to=id fromIndex=search}dc.contributor.author_authority:*
```

```
http://localhost:8983/solr/authority/select?indent=true&q.op=OR&q=%7B!join%20from%3Ddc.contributor.author_authority%20to%3Did%20fromIndex%3Depflsearch%7Ddc.contributor.author_authority%3A*
```

eventually you can filter them using the previous query `orcid_id:*` (as a `fq`) to get only the authority records that are used by some dspace items and have an ORCID.

http://localhost:8983/solr/authority/select?fq=orcid_id%3A*&indent=true&q.op=OR&q=%7B!join%20from%3Ddc.contributor.author_authority%20to%3Did%20fromIndex%3Depflsearch%7Ddc.contributor.author_authority%3A*

Usually, you need to export the solr document `id, last_name, first_name` and `orcid_id` fields, the id field contains the `uuid` used in your metadata as "authority" in the `metadatavalue` table. Using a such an export (you can force the CSV format using the wt parameter in solr)

http://localhost:8983/solr/authority/select?fl=id%2Clast_name%2Cfirst_name%2Corcid_id&fq=orcid_id%3A*&indent=true&q.op=OR&q=%7B!join%20from%3Ddc.contributor.author_authority%20to%3Did%20fromIndex%3Depflsearch%7Ddc.contributor.author_authority%3A*&wt=csv

you can

- create a CSV to import the authority record in DSpace as Person item (the SOLR output must be adapted to the DSpace Bulk Edit format)
- manipulate a CSV export of the dspace item to replace the authority value with a relation to the Person items

Batch Metadata Editing

1 Batch Metadata Editing Tool
- 1.1 Export Function
  - 1.1.1 Web Interface Export
  - 1.1.2 Command Line Export
- 1.2 Import Function
  - 1.2.1 Web Interface Import
  - 1.2.2 Command Line Import
- 1.3 CSV Format
  - 1.3.1 File Structure
- 1.4 Editing the CSV
  - 1.4.1 Editing Collection Membership
  - 1.4.2 Adding Metadata-Only Items
  - 1.4.3 Deleting Metadata
  - 1.4.4 Performing 'actions' on items
  - 1.4.5 Migrating Data or Exchanging data
  - 1.4.6 Common Issues
    - 1.4.6.1 Metadata values in CSV export seem to have duplicate columns
    - 1.4.6.2 DSpace responds with "No changes were detected" when CSV is uploaded
- 1.5 Batch Editing, Entities and Relationships
  - 1.5.1 Background about entities and virtual metadata
  - 1.5.2 Admin CSV export
  - 1.5.3 Admin CSV import

Batch Metadata Editing Tool

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

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

For information about configuration options for the Batch Metadata Editing tool, see Batch Metadata Editing Configuration

Export Function

Web Interface Export

Batch metadata exports (to CSV) can be performed from the Administrative menu:

- Login as an Administrative user
- In Sidebase, select "Export" "Metadata". Type in the Community/Collection name.
  - Alternatively, browse to the Community or Collection you wish to export, and then go to "Export" "Metadata". That Community /Collection will be preselected.
- Click "Export". A new Process will be created (in "Processes" menu). Once completed, download the resulting CSV.

Exporting search results to CSV was not added until DSpace 7.3

As of DSpace 7.3, it is possible to Export search results to a CSV *(similar to 6.x).* When logged in as an Administrator, after performing a search a new "Export search results as CSV" button appears. Clicking it will export the metadata of all items in your search results to a CSV. This CSV can then be used to perform batch metadata updates (based on the items in your search results). **- Release Notes#7.3ReleaseNotes**

Please see below documentation for more information on the CSV format and actions that can be performed by editing the CSV.

Command Line Export

The following table summarizes the basics.

Command used:	`[dspace]/bin/dspace metadata-export`
Java class:	org.dspace.app.bulkedit.MetadataExport
Arguments short and (long) forms):	Description
`-f` or `--file`	Required. The filename of the resulting CSV.
`-i` or `--id`	The Item, Collection, or Community handle or Database ID to export. If not specified, **all** items will be exported.

`-a` or `--all`	Include all the metadata fields that are not normally changed (e.g. provenance) or those fields you configured in the `[dspace]/config/modules/bulkedit.cfg` to be ignored on export.
`-h` or `--help`	Display the help page.

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

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

Example:

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

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

Please see below documentation for more information on the CSV format and actions that can be performed by editing the CSV .

Import Function

Importing large CSV files

It is not recommended to import CSV files of more than 1,000 lines (i.e. 1,000 items). When importing files larger than this, it may be difficult for an Administrator to accurately verify the changes that the import tool states it will make. In addition, depending on the memory available to the DSpace site, large files may cause 'Out Of Memory' errors part way through the import process.

Web Interface Import

Batch metadata imports (from CSV) can be performed from the Administrative menu:

- First, complete all editing of the CSV and save your changes
- Login as an Administrative User
- In sidebar, select "Import" "Metadata" and drag & drop the CSV file

Validate a Batch Metadata CSV was not added until DSpace 7.3

As of DSpace 7.3, it is now possible to *validate* a Batch Metadata CSV before applying changes *(similar to 6.x)*. When uploading a CSV for batch updates (using "Import" menu), a new "Validate Only" option is selected by default. When selected, the uploaded CSV will only be validated & you'll receive a report of the detected changes in the CSV. This allows you to verify the changes are correct before applying them. (NOTE: applying the changes requires re-submitting the CSV with the "Validate Only" option deselected) **- Release Notes#7.3ReleaseNotes**

Command Line Import

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.

To run the batch importer, at the command line:

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

Example

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

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

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

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

CSV Format

The CSV (comma separated values) files that this tool can import and export abide by the RFC4180 CSV format. This means that new lines, and embedded commas can be included by wrapping elements in double quotes. Double quotes can be included by using two double quotes. The code does all this for you, and any good csv editor such as Excel or OpenOffice will comply with this convention.

All CSV files are also in UTF-8 encoding in order to support all languages.

File Structure

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

A typical heading row looks like:

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

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

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

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

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

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

Editing the CSV

If you are editing with Microsoft Excel, be sure to open the CSV in Unicode/UTF-8 encoding

By default, Microsoft Excel may not correctly open the CSV in Unicode/UTF-8 encoding. This means that special characters may be improperly displayed and also can be "corrupted" during re-import of the CSV.

You need to tell Excel this CSV is Unicode, by importing it as follows. (*Please note these instructions are valid for MS Office 2007 and 2013. Other Office versions may vary*)

- First, open Excel (with an empty sheet/workbook open)
- Select "Data" tab
- Click "From Text" button (in the "External Data" section)
- Select your CSV file
- Wizard Step 1
  - Choose "Delimited" option
  - Start import at row: 1
  - In the "File origin" selectbox, select "65001 : Unicode (UTF-8)"
    - NOTE: these encoding options are sorted alphabetically, so "Unicode (UTF-8)" appears near the bottom of the list.
  - Click Next
- Wizard Step 2
  - Select "Comma" as the only delimiter
  - Click Next
- Wizard Step 3
  - Select "Text" as the "Column data format" (*Unfortunately, this must be done for each column individually in Excel*)
    - At a minimum, you MUST ensure all date columns (e.g. dc.date.issued) are treated as "Text" so that Excel doesn't autoconvert DSpace's YYYY-MM-DD format into MM/DD/YYYY
    - To avoid such autoconversion, it is safest to ensure each column is treated as "Text". Unfortunately, this means selecting each column one-by-one and choosing "Text" as the "Column data format".
  - Click Finish
- Choose whether to open CSV in the existing sheet or a new one

Tips to Simplify the Editing Process

When editing a CSV, here's a couple of basic tips to keep in mind:

1. The "id" column MUST remain intact. This column also must always have a value in it.
2. To simplify the CSV, you can simply remove any columns you do NOT wish to edit (except for "id" column, see #1). Don't worry, removing the entire column won't delete metadata (see #3)
3. When importing a CSV file, the importer will *overlay* the metadata onto what is already in the repository to determine the differences. It *only* acts on the contents of the CSV file, rather than on the complete item metadata. This means that the CSV file that is exported can be manipulated quite substantially before being re-imported. Rows (items) or Columns (metadata elements) can be removed and will be ignored.
   a. For example, if you only want to edit "dc.subject", you can remove ALL columns EXCEPT for "id" and "dc.subject" so that you can just manipulate the "dc.subject" field. On import, DSpace will see that you've only included the "dc.subject" field in your CSV and therefore will only update the "dc.subject" metadata field for any items listed in that CSV.
4. Because removing an entire column does NOT delete metadata value(s), if you actually wish to delete a metadata value you should leave the column intact, and simply clear out the appropriate row's value (in that column).

Editing Collection Membership

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

Adding Metadata-Only Items

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

Deleting Metadata

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

Performing 'actions' on items

It is possible to perform certain 'actions' on items. This is achieved by adding an 'action' column to the CSV file (after the id, and collection columns). There are three possible actions:

1. ***'expunge'*** This permanently deletes an item. Use with care! This action must be enabled by setting 'allowexpunge = true' in `modules/bulkedit.cfg`
2. ***'withdraw'*** This withdraws an item from the archive, but does not delete it.
3. ***'reinstate'*** This reinstates an item that has previously been withdrawn.

If an action makes no change (for example, asking to withdraw an item that is already withdrawn) then, just like metadata that has not changed, this will be ignored.

Migrating Data or Exchanging data

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

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

Common Issues

Metadata values in CSV export seem to have duplicate columns

DSpace responds with "No changes were detected" when CSV is uploaded

Unfortunately, this response may be caused in many ways

- It's possible the CSV was not saved properly after editing. Check that the edits are in the CSV, and that there were no backend errors in the DSpace logs (which would be an indication of an invalid or corrupted CSV file)
- Depending on the version of DSpace, you may be encountering this known bug with processing linebreaks in CSV files: https://github.com/DSpace/DSpace/issues/6600
- If you are setting a new embargo date in the CSV, ensure that the embargo lift date is a future date. It's been reported that past dates may cause DSpace to ignore item changes.

Batch Editing, Entities and Relationships

Consider the following page for this topic: Configurable Entities

Background about entities and virtual metadata

- In DSpace 7, we have entities. Entities are items with an entity type (there can still be items without an entity type).
- Two entities can be linked to each other. For this purpose relations are defined, which indicate the relationship between the entities. Relationships between two entities are defined by the metadata schema *relation*. The relation reflects how two entities are related to each other, for example *isPersonOfProject* or *isPublicationOfAuthor*.
- Until the introduction of entities, we could only link items to each other by inserting DOIs or URLs of other items into metadata fields *dc.relation.**. What is new about the linking between entities in DSpace 7 is that UUIDs are entered into the fields, i.e. internal identifiers of other entities that DSpace can easily resolve. DSpace "knows" which entities are linked to each other and how.
- On the item view of an entity (remember: an entity is an item with an entity type) metadata of other entities can be displayed. DSpace refers to this as virtual metadata. Virtual metadata does not belong to the item in whose item view it is displayed, but to a linked entity. They can be changed only in the linked entity. As an example: we have the entities journal and journal issue. All journal issues display the title of the journal in their item view. This title is stored only in the journal and is only (dynamically) displayed in the issues.

Admin CSV export

- Virtual metadata is exported with the entities in which it is included. For example, when you export projects, you see a column for the *project.investigator* field. Here, the names of two people have been included as virtual metadata. However, the names are not stored in the project, but exported from the respective person entities at the time of export. The specification *::virtual::* marks this. The specifications *::8585::* and *::27946::* are examples for this documentation and represent IDs of the relations. The specification *::600* comes from the DSpace Authority, which is also specified due to technical circumstances.
- The relation itself is also included in the CSV export, in the *relation.isPersonOfProject* field. Additionally, a *relation.isPersonOfProject.latestForDiscovery* field is created. This field has internal reasons in DSpace and should help to make things faster discoverable. In the fields you again see the *::virtual::8585::600* specification, which are already explained above. Instead of the values of individual metadata fields, you now have the UUIDs of the items that are linked. You can always get these UUIDs from the URL of the item view of an item.

An example heading row of the CSV export file (project entity):

```
id,collection,dc.title,project.investigator,relation.isPersonOfProject,etc,etc,etc.
```

Subsequent example row in the CSV export file (project entity):

```
350,2292,Project title,"Smith, John::virtual::8585::600||Doe, Jane::virtual::27946::600","d89c1eb1-2e7c-4912-
a1eb-f27b17fd6848::virtual::8585::600||e3595b14-6937-47b9-b718-1972cb683943::virtual::27946::600"
```

Admin CSV import

- As always, only the columns and rows that will be changed should be specified. You do not want to import the columns that contain virtual metadata, because they are not stored in the imported items, but in the linked items. So in the above example you don't want to import the *project.investigator* column, but delete it from the CSV.
- To link one item to another you need to create a corresponding column of the *relation* metadata schema, so in our example above *relation.isPersonOfProject*. All columns of the form *relation.*.latestForDiscovery* are created and updated automatically, so you don't want to import them. If you want to create a new relation, of course you don't know the ID of the relation, you can replace it with a +, then DSpace will assign it on its own. Of course, people can also be removed from the column or completely new relations can be created for new items, even if there are no old ones to be taken over.

An example heading row for the CSV import file (project entity):

```
id,collection,dc.title,relation.isPersonOfProject,etc,etc,etc.
```

Subsequent example row for the CSV import file (project entity):

```
350,2292,Project title,"d89c1eb1-2e7c-4912-a1eb-f27b17fd6848::virtual::8585::600||e3595b14-6937-47b9-b718-
1972cb683943::virtual::+::600"
```

Batch Metadata Editing Configuration

The Batch Metadata Editing Tool allows the administrator to extract from the DSpace database a set of records for editing via a CSV file. It provides an easier way of editing large collections.

A full list of all available Batch Metadata Editing Configurations:

Configuration File:	**`[dspace]/config/modules/bulkedit.cfg`**
Property:	`bulkedit.valueseparator`
Example Value:	`bulkedit.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:	`bulkedit.fieldseparator`
Example Value:	`bulkedit.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`. `bulkedit.fieldseparator = tab`
Property:	`bulkedit.authorityseparator`
Example Value:	`bulkedit.authorityseparator = ::`
Informational note	The delimiter used to separate authority data (defaults to a double colon ::)
Property:	`bulkedit.export.max.items`
Example Value:	`bulkedit.export.max.items = 500`
Informational note	Maximum number of items which can be exported at one time. This corresponds to the maximum number of data rows which will appear in the exported CSV. Default is 500 items.
Property:	`bulkedit.gui-item-limit`
Example Value:	`bulkedit.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:	`bulkedit.ignore-on-export`

Example Value:	`bulkedit.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.
Property:	`bulkedit.allowexpunge`
Example Value:	`bulkedit.allowexpunge = false`
Informational note	Should the 'action' column allow the 'expunge' method. By default this is set to false
Property	`bulkedit.allow-bulk-deletion`
Example Value:	`bulkedit.allow-bulk-deletion = dspace.agreements.end-user`

Informational note	Comma separated list of metadata fields that can be deleted in bulk using the 'metadata-deletion' script. By default only the 'dspace.agreements.end-user' field can be deleted in bulk, as doing so allows an Administrator to force all users to re-review the End User Agreement on their next login. However, you may choose to enable additional fields. Keep in mind, any fields listed here may be batch deleted from the Processes UI & such metadata deletions cannot be undone.

DOI Digital Object Identifier

- Persistent Identifier
- DOI Registration Agencies
  - Configure DSpace to use the DataCite API
    - dspace.cfg
    - Metadata conversion
    - Identifier Service
    - Sending metadata updates to DataCite
    - DOIs using DataCite and Item Level Versioning
    - Command Line Interface
    - 'cron' job for asynchronous reservation/registration
    - Limitations of DataCite DOI support
  - Configure DSpace to use EZID service for registration of DOIs
    - Limitations of EZID DOI support
- Adding support for other Registration Agencies
- Configuring pre-registration of Identifiers
  - Why mint in submission?
  - Enable the Identifiers step
  - Configure filters and behaviour
  - Administrator registration
- Standardizing the format and location of DOI metadata
  - Usage
  - How-to guide for a typical institution

Persistent Identifier

It is good practice to use Persistent Identifiers to address items in a digital repository. There are many different systems for Persistent Identifiers: Handle , DOI , urn:nbn, purl and many more. It is far out of the scope of this document to discuss the differences of all these systems. For several reasons the Handle System is deeply integrated in DSpace, and DSpace makes intensive use of it. With DSpace 3.0 the Identifier Service was introduced that makes it possible to also use external identifier services within DSpace.

DOIs are Persistent Identifiers like Handles are, but as many big publishing companies use DOIs they are quite well-known to scientists. Some journals ask for DOIs to link supplemental material whenever an article is submitted. Beginning with DSpace 4.0 it is possible to use DOIs in parallel to the Handle System within DSpace. By "using DOIs" we mean automatic generation, reservation and registration of DOIs for every item that enters the repository. These newly registered DOIs will not be used as a means to build URLs to DSpace items. Items will still rely on handle assignment for the item urls.

DOI Registration Agencies

To register a DOI one has to enter into a contract with a DOI registration agency which is a member of the International DOI Foundation. Several such agencies exist. Different DOI registration agencies have different policies. Some of them offer DOI registration especially or only for academic institutions, others only for publishing companies. Most of the registration agencies charge fees for registering DOIs, and all of them have different rules describing for what kind of item a DOI can be registered. To make it quite clear: to register DOIs with DSpace you have to enter into a contract with a DOI registration agency.

DataCite is an international initiative to promote science and research, and a member of the International DOI Foundation. The members of DataCite act as registration agencies for DOIs. Some DataCite members provide their own APIs to reserve and register DOIs; others let their clients use the DataCite API directly. Starting with version 4.0 DSpace supports the administration of DOIs by using the DataCite API directly or by using the API from EZID (which is a service of the University of California Digital Library). This means you can administer DOIs with DSpace if your registration agency allows you to use the DataCite API directly or if your registration agency is EZID.

Configure DSpace to use the DataCite API

If you use a DOI registration agency that lets you use the DataCite API directly, you can follow the instructions below to configure DSpace. In case EZID is your registration agency the configuration of DSpace is documented here: Configure DSpace to use EZID service for registration of DOIs.

To use DOIs within DSpace you have to configure several parts of DSpace:

- enter your DOI prefix and the credentials to use the API from DataCite in dspace.cfg,
- configure the script which generates some metadata,
- activate the DOI mechanism within DSpace,
- configure a cron job which transmits the information about new and changed DOIs to the registration agency.

dspace.cfg

After you enter into a contract with a DOI registration agency, they'll provide you with user credentials and a DOI prefix. You have to enter these in the dspace cfg. Here is a list of DOI configuration options in dspace.cfg:

Configuration File:	**[dspace]/config/dspace.cfg**

Property:	`identifier.doi.user`
Example Value:	`identifier.doi.user = user123`
Informational Note:	Username to login into the API of the DOI registration agency. You'll get it from your DOI registration agency.
Property:	`identifier.doi.password`
Example Value:	`identifier.doi.password = top-secret`
Informational Note:	Password to login into the API of the DOI registration agency. You'll get it from your DOI registration agency.
Property:	`identifier.doi.prefix`
Example Value:	`identifier.doi.prefix = 10.5072`
Informational Note:	The prefix you got from the DOI registration agency. All your DOIs start with this prefix, followed by a slash and a suffix generated from DSpace. The prefix can be compared with a namespace within the DOI system.
Property:	`identifier.doi.namespaceseparator`
Example Value:	`identifier.doi.namespaceseparator = dspace-`
Informational Note:	This property is optional. If you want to use the same DOI prefix in several DSpace installations or with other tools that generate and register DOIs it is necessary to use a namespace separator. All the DOIs that DSpace generates will start with the DOI prefix, followed by a slash, the namespace separator and some number generated by DSpace. For example, if your prefix is 10.5072 and you want all DOIs generated by DSpace to look like 10.5072/dspace-1023 you have to set this as in the example value above.
Property:	`identifier.doi.resolver`
Example Value:	`identifier.doi.resolver = https://doi.org`
Informational Note:	URL for the DOI resolver. This will be the stem for generated DOIs. This property is optional, and defaults to the example value above.
Property:	crosswalk.dissemination.DataCite.publisher
Example Value:	crosswalk.dissemination.DataCite.publisher = My University Press
Informational Note:	The name of the publishing institution or publisher.
Property:	`crosswalk.dissemination.DataCite.publisherRor`
Example Value:	`crosswalk.dissemination.DataCite.publisherRor = https://ror.org/0123abc40`
Informational Note:	The ROR (as a full URL) of the publishing institution or publisher which is named by `crosswalk.dissemination.DataCite.publisher`
Property:	crosswalk.dissemination.DataCite.hostingInstitution
Example Value:	crosswalk.dissemination.DataCite.hostingInstitution = My University
Informational Note:	The name of the organization/institution which hosts this instance of the object. If not configured, it will default to the value of crosswalk.dissemination.DataCite.publisher.
Property:	crosswalk.dissemination.DataCite.dataManager
Example Value:	crosswalk.dissemination.DataCite.dataManager = My University Department of Geology
Informational Note:	If not configured, it will default to the value of crosswalk.dissemination.DataCite.publisher.

Please don't use the test prefix 10.5072 with DSpace. The test prefix 10.5072 differs from other prefixes: It answers GET requests for all DOIs even for DOIs that are unregistered. DSpace checks that it mint only unused DOIs and will create an Error: "Register DOI ... failed: DOI_ALREADY_EXISTS". Your registration agency can provide you an individual test prefix, that you can use for tests.

Metadata conversion

To reserve or register a DOI, DataCite requires that metadata be supplied which describe the object that the DOI addresses. The file [dspace]/config /crosswalks/DIM2DataCite.xsl controls the conversion of metadata from the DSpace internal format into the DataCite format:
[dspace]/config/crosswalks/DIM2DataCite.xsl

```
<!--
    Document   : DIM2DataCite.xsl
    Created on : January 23, 2013, 1:26 PM
    Author     : pbecker, ffuerste
    Description: Converts metadata from DSpace Intermediat Format (DIM) into
                 metadata following the DataCite Schema for the Publication and
                 Citation of Research Data, Version 2.2
-->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:dspace="http://www.dspace.org/xmlns/dspace/dim"
                xmlns="http://datacite.org/schema/kernel-2.2"
                version="1.0">

    <!-- CONFIGURATION -->
    <!-- Please add your DOI-Prefix and your namespace separator here (e.g. 10.5072-dspace-). -->
    <xsl:variable name="prefix">10.5072-dspace-</xsl:variable>
    <!-- The content of the following variable will be used as element publisher. -->
    <xsl:variable name="publisher">My University</xsl:variable>
    <!-- The content of the following variable will be used as element contributor with contributorType
datamanager. -->
    <xsl:variable name="datamanager"><xsl:value-of select="$publisher" /></xsl:variable>
    <!-- The content of the following variable will be used as element contributor with contributorType
hostingInstitution. -->
    <xsl:variable name="hostinginstitution"><xsl:value-of select="$publisher" /></xsl:variable>
    <!-- Please take a look into the DataCite schema documentation if you want to know how to use these elements.

         http://schema.datacite.org -->


    <!-- DO NOT CHANGE ANYTHING BELOW THIS LINE EXCEPT YOU REALLY KNOW WHAT YOU ARE DOING! -->
...
```

To change the Publisher, data manager and/or hosting Institution that will be send to DataCite as part of the metadata, use the `crosswalk.dissemination.DataCite.*` properties in `local.cfg`.

If you want to know more about the DataCite Schema, have a look at the documentation. If you change this file in a way that is not compatible with the DataCite schema, you won't be able to reserve and register DOIs anymore. Do not change anything if you're not sure what you're doing. To get the XML on which the XSLT processor will start, use the following command:

```
[dspace]/bin/dspace dsrun org.dspace.content.crosswalk.XSLTDisseminationCrosswalk dim 123456789/3
```

To get the XML that will be send to DataCite replace 'dim' with 'DataCite'. If the DOI is not stored in the metadata, DSpace will add it automatically as identifier. So don't worry if the XML produced by this command does not contain the DOI. Once the DOI is stored in the metadata, it should also be contained in the XML.

Identifier Service

The Identifier Service manages the generation, reservation and registration of identifiers within DSpace. You can configure it using the config file located in [dspace]/config/spring/api/identifier-service.xml. In the file you should already find the code to configure DSpace to register DOIs. Just read the comments and remove the comment signs around the two appropriate beans.

After removing the comment signs the file should look something like this (I removed the comments to make the listing shorter):

[dspace]/config/spring/api/identifier-service.xml

```
<!--
    Copyright (c) 2002-2010, DuraSpace.  All rights reserved
    Licensed under the DuraSpace License.

    A copy of the DuraSpace License has been included in this
    distribution and is available at: http://www.dspace.org/license
-->

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

    <bean id="org.dspace.identifier.service.IdentifierService"
          class="org.dspace.identifier.IdentifierServiceImpl"
          autowire="byType"
          scope="singleton"/>

    <bean id="org.dspace.identifier.DOIIdentifierProvider"
        class="org.dspace.identifier.DOIIdentifierProvider"
        scope="singleton">
        <property name="configurationService"
            ref="org.dspace.services.ConfigurationService" />
        <property name="DOIConnector"
            ref="org.dspace.identifier.doi.DOIConnector" />
    </bean>

    <bean id="org.dspace.identifier.doi.DOIConnector"
        class="org.dspace.identifier.doi.DataCiteConnector"
        scope="singleton">
        <property name='DATACITE_SCHEME' value='https'/>
        <property name='DATACITE_HOST' value='mds.test.datacite.org'/>
        <property name='DATACITE_DOI_PATH' value='/doi/' />
        <property name='DATACITE_METADATA_PATH' value='/metadata/' />
        <property name='disseminationCrosswalkName' value="DataCite" />
    </bean>
</beans>
```

If you use other IdentifierProviders beside the DOIIdentifierProvider there will be more beans in this file.

Please pay attention to configure the property DATACITE_HOST. Per default it is set to the DataCite test server. To reserve real DOIs you will have to change it to mds.datacite.org. Ask your registration agency if you're not sure about the correct address.

Sending metadata updates to DataCite

DSpace should send updates to DataCite whenever the metadata of an item changes. To do so, you must enable the DOIConsumer in your dspace.cfg (or local.cfg. You should remove the comments in front of the two following properties or add them to the local.cfg:

[dspace]/config/dspace.cfg

```
event.consumer.doi.class = org.dspace.identifier.doi.DOIConsumer
event.consumer.doi.filters = Item+Modify_Metadata
```

Then you should add 'doi' to the property `event.dispatcher.default.consumers`. After adding it, this property may look like this:

[dspace]/config/dspace.cfg

```
event.dispatcher.default.consumers = versioning, discovery, eperson, harvester, doi
```

DOIs using DataCite and Item Level Versioning

If you enabled Item Level Versioning you should enable the `VersionedDOIIdentifierProvider` instead of the `DOIIdentifierProvider`. The `VersionedDOIIdentifierProvider` ensures that newer versions of the same Item gets a DOI looking as the DOI of the first version of and item, extended by a dot and the version number. With DSpace 6 this also became the default for handles if Item Level Versioning is enabled. In the configuration file `[dspace]/config/spring/api/identifier-service.xml` you'll find the possiblity to enable the `VersionedDOIIdentifierProvider`. If you want to use versioned DOIS, please comment out the `DOIIdentifierProvider` as only one of both DOIProviders should be enabled at the same time.

Command Line Interface

To make DSpace resistant to outages of DataCite we decided to separate the DOI support into two parts. When a DOI should be generated, reserved or minted, DSpace does this in its own database. To perform registration and/or reservation against the DOI registration agency a job has to be started using the command line. Obviously this should be done by a cron job periodically. In this section we describe the command line interface, in case you ever want to use it manually. In the next section you'll see the cron job that transfers all DOIs designated for reservation and/or registration.

The command line interface in general is documented here: Command Line Operations.

The command used for DOIs is '`doi-organiser`'. You can use the following options:

Option (short)	Option (long)	Parameter	Description
-d	--delete-all		Transmit information to the DOI registration agency about all DOIs that were deleted.
	--delete-doi	DOI	Transmit information to the DOI registration agency that the specified DOI was deleted. The DOI must already be marked for deletion; you cannot use this command to delete a DOI for an exisiting item.
-h	--help		Print online help.
-l	--list		List all DOIs whose changes were not committed to the registration agency yet.
-q	--quiet		The doi-organiser sends error reports to the mail address configured in the property alert.recipient in dspace.cfg. If you use this option no output should be given to stdout. If you do not use this option the doi-organiser writes information about successful and unsuccessful operations to stdout and stderr. You can find information in dspace.log of course.
-r	--register-all		Transmit information about all DOIs that should be registered.
	--register-doi	DOI | ItemID | handle	If a DOI is marked for registration, you can trigger the registration at the DOI registration agency by this command. Specify either the DOI, the ID of the item, or its handle.
-s	--reserve-all		Transmit to the DOI registration agency information about all DOIs that should be reserved.
	--reserve-doi	DOI | ItemID | handle	If a DOI is marked for registration, you can trigger the registration at the DOI registration agency by this command. Specify either the DOI, the ID of the item, or its handle.
-u	--update-all		If a DOI is reserved for an item, the metadata of the item will be sent to DataCite. This command transmits new metadata for items whose metadata were changed since the DOI was reserved.
	--update-doi	DOI | ItemID | handle	If a DOI needs an update of the metadata of the item it belongs to, you can trigger this update with this command. Specify either the DOI, the ID of the item, or its handle.

Currently you cannot generate new DOIs with this tool. You can only send information about changes in your local DSpace database to the registration agency.

'cron' job for asynchronous reservation/registration

When a DOI should be reserved, registered, deleted or its metadata updated, DSpace just writes this information into its local database. A command line interface is supplied to send the necessary information to the registration agency. This behavior makes it easier to react to outages or errors while using the API. This information should be sent regularly, so it is a good idea to set up a cron job instead of doing it manually.

There are four commands that should be run regularly:

- Update the metadata of all items that have changed since their DOI was reserved.
- Reserve all DOIs marked for reservation
- Register all DOIs marked for registration
- Delete all DOIs marked for deletion

In DSpace, a DOI can have different states (see the following table). After updating an item's metadata the state of its assigned DOI is set back to the last state it had before. So, e.g., if a DOI has the state "to be registered" and the metadata of its item changes, it will be set to the state "needs update". After the update is performed its state is set to "to be registered" again. Because of this behavior **the order of the commands above matters**: the update command must be executed before all of the other commands above.

State #	State Name	DSpace State Description	DataCite State Name
0	UNKNOWN	???	N/A*

1	TO_BE_REGISTERED	The submission has been approved and the DOI has been registered in DSpace. The registration has not yet been published to DataCite.	Draft
2	TO_BE_RESERVED	The minted DOI has been associated with an item. However, the reservation has not yet been published to DataCite.	N/A*
3	IS_REGISTERED	The registration has been published to DataCite and a url back to DSpace has been associated with the DOI.	Findable
4	IS_RESERVED	The reservation has been published to DataCite.	Draft
5	UPDATE_RESERVED	A "Reserved" DOI's metadata has been modified in DSpace, but its modification has not yet been communicated to DataCite.	Draft
6	UPDATE_REGISTERED	A "Registered" DOI's metadata has been modified in DSpace, but its modification has not yet been communicated to DataCite.	Findable
7	UPDATE_BEFORE_REGISTRATION	A "To be registered" DOI's metadata has been modified in DSpace, but its modification has not yet been communicated to DataCite.	N/A*
8	TO_BE_DELETED	An item with a registered DOI has been deleted in DSpace. However, the DOI still exists in DataCite.	
9	DELETED	The DOI has been deleted in both DSpace and DataCite.	Tombstone? or nothing visible anymore?
10	PENDING	The DOI was minted during submission. It was shown to a user that the submitted item would get this DOI if it ever gets any. The DOI is pending until it is archived and passes any filter by the DOIIdentifierProvider or is assigned to the item manually by an admin.	
11	MINTED	A DOI was created during the submission and a user was informed that the submission will get this DOI if any. Then the user changed the submission in a way that ceased to satisfy the DOI filter. In case the item passes the filter again, we want to re-apply the same DOI again (without creating new database rows), so we store it as being minted. DataCite was not informed about this DOI.	

* An item in this state is not visible in DataCite

The cron job should perform the following commands with the rights of the user your DSpace installation runs as:

```
[dspace]/bin/dspace doi-organiser -u -q
[dspace]/bin/dspace doi-organiser -s -q
[dspace]/bin/dspace doi-organiser -r -q
[dspace]/bin/dspace doi-organiser -d -q
```

The doi-organiser sends error messages as email and logs some additional information. The option -q tells DSpace to be quiet. If you don't use this option the doi-organiser will print messages to stdout about every DOI it successfully reserved, registered, updated or deleted. Using a cron job these messages would be sent as email.

In case of an error, consult the log messages. If there is an outage of the API of your registration agency, DSpace will not change the state of the DOIs so that it will do everything necessary when the cron job starts the next time and the API is reachable again.

The frequency the cron job runs depends on your needs and your hardware. The more often you run the cron job the faster your new DOIs will be available online. If you have a lot of submissions and want the DOIs to be available really quickly, you probably should run the cron job every fifteen minutes. If there are just one or two submissions per day, it should be enough to run the cron job twice a day.

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

```
crontab -e
```

The following line tells cron to run the necessary commands twice a day, at 1am and 1pm. Please notice that the line starting with the numbers is one line, even it it should be shown as multiple lines in your browser.

```
# Send information about new and changed DOIs to the DOI registration agency:
0 1,13 * * *  [dspace]/bin/dspace doi-organiser -u -q ; [dspace]/bin/dspace doi-organiser -s -q ; [dspace]/bin
/dspace doi-organiser -r -q ; [dspace]/bin/dspace doi-organiser -d -q
```

Limitations of DataCite DOI support

Every DSpace installation expects to be the only application that generates DOIs which start with the prefix and the namespace separator you configured. DSpace does not check whether a DOI it generates is reserved or registered already.

That means if you want to use other applications or even more than one DSpace installation to register DOIs with the same prefix, you'll have to use a unique namespace separator for each of them. Also you should not generate DOIs manually with the same prefix and namespace separator you configured within DSpace. For example, if your prefix is 10.5072 you can configure one DSpace installation to generate DOIs starting with 10.5072/papers-, a second installation to generate DOIs starting with 10.5072/data- and another application to generate DOIs starting with 10.5072/results-.

DOIs will be used in addition to Handles. This implementation does not replace Handles with DOIs in DSpace. That means that DSpace will still generate Handles for every item, every collection and every community, and will use those Handles as part of the URL of items, collections and communities.

DSpace currently generates DOIs for items only. There is no support to generate DOIs for Communities and collections yet.

When using DSpace's support for the DataCite API probably not all information would be restored when using the AIP Backup and Restore (see https://github.com/DSpace/DSpace/issues/5203). The DOIs included in metadata of Items will be restored, but DSpace won't update the metadata of those items at DataCite anymore. You can even get problems when minting new DOIs after you restored older once using AIP.

Configure DSpace to use EZID service for registration of DOIs

The EZID IdentifierProvider operates synchronously, so there is much less to configure. You will need to un-comment the `org.dspace.identifier.EZIDIdentifierProvider` bean in `config/spring/api/identifier-service.xml` to enable DOI registration through EZID.

In `config/dspace.cfg` you will find a small block of settings whose names begin with `identifier.doi.ezid`. You should uncomment these properties and give them appropriate values. Sample values for a test account are supplied.

name	meaning
identifier.doi.ezid.shoulder	The "shoulder" is the DOI prefix issued to you by the EZID service. DOIs minted by this instance of DSpace will be the concatenation of the "shoulder" and a locally unique token.
identifier.doi.ezid.user identifier.doi.ezid.password	The username and password by which you authenticate to EZID.
identifier.doi.ezid.publisher	You may specify a default value for the required `datacite.publisher` metadatum, for use when the Item has no publisher.
crosswalk.dissemination.DataCite.publisher	Should match identifier.doi.ezid.publisher.
crosswalk.dissemination.DataCite.hostingInstitution	Name of the hosting institution. If not configured, it will be set to the value of crosswalk.dissemination.DataCite.publisher.
crosswalk.dissemination.DataCite.dataManager	Name of the data manager. If not configured, it will be set to the value of crosswalk.dissemination.DataCite.publisher.

Back in `config/spring/api/identifier-service.xml` you will see some other configuration of the `EZIDIdentiferProvider` bean. In most situations, the default settings should work well. But, here's an explanation of options available:

- *EZID Provider / Registrar settings:* By default, the EZIDIdentifierProvider is configured to use the CDLib provider (ezid.cdlib.org) in the `EZID_SCHEME`, `EZID_HOST` and `EZID_PATH` settings. In most situations, the default values should work for you. However, you may need to modify these values (especially the `EZID_HOST`) if you are registered with a different EZID provider. In that situation, please check with your provider for valid "host" and "path" settings. If your provider provides EZID service at a particular path on its host, you may set that in `EZID_PATH`.
  - NOTE: As of the writing of this documentation, the default CDLib provider settings should also work for institutions that use Purdue (ezid.lib.purdue.edu) as a provider. Currently, Purdue and CDLib currently share the same infrastructure, and both `ezid.cdlib.org` and `ezid.lib.purdue.edu` point to the same location.
- *Metadata mappings:* You can alter the mapping between DSpace and EZID metadata, should you choose. The `crosswalk` property is a map from DSpace metadata fields to EZID fields, and can be extended or changed. The `key` of each `entry` is the name of an EZID metadata field; the `value` is the name of the corresponding DSpace field, from which the EZID metadata will be populated.
- *Crosswalking / Transforms:* You can also supply transformations to be applied to field values using the `crosswalkTransform` property. Each `key` is the name of an EZID metadata field, and its `value` is the name of a Java class which will convert the value of the corresponding DSpace field to its EZID form. The only transformation currently provided is one which converts a date to the year of that date, named `org.dspace.identifier.ezid.DateToYear`. In the configuration as delivered, it is used to convert the date of issue to the year of publication. You may create new Java classes with which to supply other transformations, and map them to metadata fields here. If an EZID metadatum is not named in this map, the default mapping is applied: the string value of the DSpace field is copied verbatim.

Limitations of EZID DOI support

DOIs will be used in addition to Handles. This implementation does not replace Handles with DOIs in DSpace. That means that DSpace will continue to generate Handles for every item, every collection and every community, and will use those Handles as part of the URL of items, collections and communities.

Currently, the EZIDIdentifierProvider has a known issue where it stores its DOIs in the `dc.identifier` field, instead of using the `dc.identifier.uri` field (which is the one used by DataCite DOIs and Handles). See https://github.com/DSpace/DSpace/pull/1006 for more details. This will be corrected in a future version of DSpace.

DSpace currently generates DOIs for items only. There is no support to generate DOIs for Communities and Collections yet.

Adding support for other Registration Agencies

If you want DSpace to support other registration agencies, you just have to write a Java class that implements the interface DOIConnector ([dspace-source]/dspace-api/src/main/java/org/dspace/identifier/doi/DOIConnector.java). You might use the DataCiteConnector ([dspace-source]/dspace-api/src/main/java/org/dspace/identifier/doi/DataCiteConnector.java) as an example. After developing your own DOIConnector, you configure DSpace as if you were using the DataCite API directly. Just use your DOIConnector when configuring the IdentifierService instead of the DataCiteConnector.

Configuring pre-registration of Identifiers

Why mint in submission?

Users often want to see what DOI they **will** get so they can alter their PDF, coverpage, other metadata, and so on.

This feature should ensure that users can see their future DOI, and if necessary, a warning that if certain conditions are not met, the DOI will not be registered after approval.

Keeping a DOI in pending status does use up an integer from the total DOI namespace, but it also ensures that the submitter, reviewers, administrators etc know what the DOI will be if it is ever registered in the future.

If this is really not desired, eg. there are many item types which should never get a DOI, then there is a way to configure a filter that avoids minting a new PENDING DOI at all unless conditions are met in submission.

Enable the Identifiers step

See Submission User Interface#Configuringthe%22Identifiers%22step

Configure filters and behaviour

To enable this feature and configure the exact way it works, edit the `${dspace.dir}/dspace/modules/identifiers.cfg` configuration file

Property:	`identifiers.submission.register`
Example Value:	`true`
Informational Note:	Enable this feature. Default: false. Handles will be registered at time of submission. DOIs (if item filters evaluate to true) will be minted in a "pending" state for items, to be registered or queued for registration at archival.
Property:	`identifiers.submission.filter.install`
Example Value:	`doi-filter`
Informational Note:	Bean ID of a logical item filter (see `config/modules/spring/api/item-filters.xml`) that will be used to evaluate whether a DOI should be queued for registration when this item is installed (archived) in DSpace. This filter will be applied whether or not a "pending" DOI is already minted for the item. (If a filter is absent or null, an item will always be evaluated as 'true')
Property:	`identifiers.submission.filter.workspace`
Example Value:	`always_true_filter`
Informational note	Bean ID of a logical item filter (see `config/modules/spring/api/item-filters.xml`) that will be used to evaluate whether a DOI should be minted as "pending" for registration when this item is first submitted as a workspace item in DSpace. Depending on the value of `identifiers.submission.strip_pending_during_submission` this filter will be checked whenever the workspace item changes, to see if it now qualifies for a DOI. Default: always_true_filter (If a filter is absent or null, an item will always be evaluated as 'true')

Property:	`identifiers.submission.strip_pending_during_submission`
Example Value:	`true`
Informational Note:	If, during workspace item changes, the workspace filter **no longer** evaluates to true, should any DOIs be stripped? (moved to MINTED or DELETED status) This is useful in situations where the submitter needs real-time feedback as to whether their item qualifies for a DOI.
Property:	`identifiers.item-status.register-doi`
Example Value:	`false`
Informational Note:	Allow administrators to queue DOIs for registration in the Item Status page. Default: false. **Important:** This configuration property **must** be set, even if it matches the default, as it is exposed as a REST configuration property to the frontend.

Administrator registration

If an item does not have a DOI at all, or if an item has a MINTED or PENDING DOI, a user with ADMIN rights over the item may queue the DOI registration from the Item Status page. No filters will be applied to this action. This requires `identifiers.item-status.register-doi` to be `true` in identifiers configuration (see above)

Standardizing the format and location of DOI metadata

As of DSpace 10 the default practice for storing DOI metadata has changed:

- DOIs minted from within DSpace will be stored in `dc.identifier.doi` metadata
- DOIs of documents imported from crosswalks will be stored in `dc.relation.hasversion`

In both cases, DOIs will now be stored in the HTTPS URL form `https://doi.org/10.<whatever>`, rather than just plain `10.<whatever>` or `doi:10.<whatever>` or the old HTTP URL form `http://dx.doi.org/10.<whatever>`.

In previous versions, DOIs could end up inconsistently in `dc.identifier` or `dc.identifier.doi` or `dc.identifier.uri`, and may have had different formats, in both cases depending where they came from.

The `doi-migrate` command line tool helps to migrate DOI metadata between fields and normalize their format. It can move existing DOI metadata from the older fields and formats to this newer standard, but can also move to other fields if desired.

```
[dspace]/bin/dspace doi-migrate [options]
```

Usage

```
Selecting which DOIs to migrate:

--only-prefix <doi-prefix>     Only migrate DOIs with the given prefix
--except-prefix <doi-prefix>   Only migrate DOIs without the given prefix

Choosing which fields to affect:

-f, --from <field-name>    Move DOIs from this field
-t, --to <field-name>      Move DOIs into this field
(--from and --to can be the same if only the formatting of DOIs should be adjusted)

Operating modes:

-n, --dry-run  Don't actually perform a migration, just generate a CSV (on standard output)
--rehydrate    Perform the migrations from a previous --dry-run invocation (CSV read from standard input)
```

You must either give both `--to` and `--from` plus one of `--only-prefix` and `--except-prefix`, OR `--rehydrate`.

The tool recognizes DOIs in any of the following formats:

- a plain DOI such as `10.1234/56789`
- an old-style DOI URL like `http://dx.doi.org/10.1234/56789` (including if HTTPS was used)
- a new-style DOI URL like `https://doi.org/10.1234/56789` (including if plain HTTP was used)
- a DOI URI like `doi:10.1234/56789`

It normalizes all of these to a new-style DOI URL with HTTPS (`https://doi.org/10.1234/56789`)

How-to guide for a typical institution

The tool is set up to assume that all of your DOI minted in-house have the same prefix, and all other prefixes are external. (The prefix is the part of the DOI before the `/` character.) This example will assume an institution whose own publications have the prefix `10.1729`. The institution's minted DOIs have so far been placed in `dc.identifier`.

First we will create a CSV file of all the in-house DOIs that need to be migrated. We will move them into `dc.identifier.doi` field in the new format.

```
dspace doi-migrate --dry-run --from dc.identifier --to dc.identifier.doi --only-prefix 10.1729 > in-house-dois.csv
```

This will create a file `in-house-dois.csv` that can be checked in any spreadsheet software, including Microsoft Excel, Apple Numbers, Google Sheets, or LibreOffice Calc. Check that all of the proposed changes to metadata look correct. The columns are in the order: Item UUID; field name that the metadata will be moved from; current value of that metadata field on the item; field name that the metadata will be moved to; changed/normalized value of the metadata field on that item.

Having checked that the proposed changes look okay, run the migration with:

```
dspace doi-migrate --rehydrate < in-house-dois.csv
```

If there are multiple prefixes for publications whose DOIs were minted locally, this process can be repeated several times.

Then we will migrate all the DOIs for publications which were imported from external sources like ArXiv. The importers used to store these in `dc.identifier.doi` without the HTTPS URL; following the new DSpace standard, we will move them to `dc.relation.hasversion`. The usage is very similar:

```
dspace doi-migrate --dry-run --from dc.identifier.doi --to dc.relation.hasversion --except-prefix 10.1729 >
imported-dois.csv
```

Again, we check the `imported-dois.csv` file to make sure that everything looks to be in order. Then we actually run the migrations:

```
dspace doi-migrate --rehydrate < imported-dois.csv
```

Item Level Versioning

1 What is Item Level Versioning?
2 Important warnings
3 Disabling Item Level Versioning
4 Initial Requirements
5 User Interface
    5.1 General behaviour: Linear Versioning
    5.2 Creating a new version of an item
    5.3 View the history and older versions of an item
6 Architecture
    6.1 Versioning model
7 Configuration
    7.1 Versioning Service Override
    7.2 Identifier Service Override
    7.3 Version History Visibility
        7.3.1 Hide Editor/Submitter details in version table
    7.4 Allowing submitters to version their items
8 Identified Challenges & Known Issues
    8.1 Conceptual compatibility with Embargo
    8.2 Conceptual compatibility with Item Level Statistics

What is Item Level Versioning?

Versioning is a new functionality to build the history of an item. Users will have the opportunity to create new version of an existing item any time the will make a change.

Supported in 7.1 or above

Item level versioning was not fully supported in DSpace 7.0 (you were only able to view existing versions). It was restored in DSpace 7.1. See DSpace Release 7.0 Status

Important warnings

Item Level Versioning on Entities configuration

Configurable Entities are supported in Item Level Versioning support starting from version 7.3. More details about the configuration specific to Configurable Entities can be found on that page.

AIP Backup & Restore functionality only works with the Latest Version of Items

If you are using the AIP Backup and Restore functionality to backup / restore / migrate DSpace Content, you must be aware that the "Item Level Versioning" feature is **not yet compatible** with AIP Backup & Restore. **Using them together may result in accidental data loss.** Currently the AIPs that DSpace generates only store the *latest version* of an Item. Therefore, past versions of Items will always be lost when you perform a restore / replace using AIP tools. See https://github.com/DSpace/DSpace/issues/4751.

DSpace 6+ changed the way Handles are created for versioned Items

Starting with 6.0, the way DSpace crates Handles for versioned Items was changed. If you want to keep the old behavior of DSpace 4 and 5 you have to enable the `VersionedHandleIdentifierProviderWithCanonicalHandles` in the XML configuration files `[dspace]/config/spring/api/identifier-service.xml`. See IdentifierServiceOverride below for details and the comments in the configuration file.

Disabling Item Level Versioning

By default, Item Level Versioning is enabled in DSpace 7. You may choose to disable it by updating this configuration in your local.cfg:

```
versioning.enabled = false
```

Additionally, you will need to make the following changes to disable all versioning-related features:

- Switch to using the basic "HandleIdentifierProvider" in your `[dspace]/config/spring/api/identifier-services.xml`. Make sure to comment out the "VersionedHandleIdentifierProvider" and replace it with this:

```
<!-- This HandleIdentiferProvider should be used when versioning is disabled -->
<bean id="org.dspace.identifier.HandleIdentifierProvider" class="org.dspace.identifier.
HandleIdentifierProvider" scope="singleton">
    <property name="configurationService" ref="org.dspace.services.ConfigurationService"/>
</bean>
```

- Remove the "versioning" consumer from the list of default Event Consumers in either your dspace.cfg or local.cfg. Look for this configuration:

```
# Remove the "versioning" entry in this list
# (NOTE: Your list of consumers may be different based on the features you've enabled)
#event.dispatcher.default.consumers = versioning, discovery, eperson

# For example:
event.dispatcher.default.consumers = discovery, eperson
```

Once these changes are made, you will need to restart your servlet container (e.g. Tomcat) for the new settings to take effect.

Initial Requirements

The Item Level Versioning implementation builds on following requirements identified by the stakeholders who supported this contribution: Initial Requirements Analysis

1. What should be *Versionable*
    a. Versioning happens at the level of an Individual Item
    b. Versioning should preserve the current state of *metadata*, *bitstreams* and *resource policies* attached to the item.
2. Access, Search and Discovery
    a. Only the most recent version of an item is available via the search interface
    b. Previous versions of Items should continue to be visible, citable and accessible
    c. The Bitstreams for previous versions are retained. If something was once retrievable, it should always be retrievable.
3. Identifiers
    a. Each version of an Item is represented by a separate "*versioned*" identifier
    b. A base "*versionhistory*" Identifier points to the most recent version of the Item.
    c. A revision identifier also exists that is unique to the specific version.
    d. When a new version of an Item is deposited, a new revision identifier will be created.
4. Presentation
    a. On the item page, there is a link to view previous/subsequent versions.
    b. By examining the metadata or identifiers, it is possible to determine whether an item is the most recent version, the original version, or an intermediate version.
5. Access Control and Rights
    a. Certain roles should be able to generate a new version of the item via submission.
    b. To submitters, collection manager, administrators will be given to option to create new version of an item.
    c. Rights to access a specific Item should transmute as well to previous versions
    d. Rights to access a specific Bitstream should also transmute to previous versions.
6. Data Integrity
    a. The relationships between versions should not be brittle and breakable by manipulating Item metadata records.
    b. The relationships between versions should be preserved and predictable in various Metadata Exports (OAI, Packagers, ItemExport)
    c. The relationships between versions should be maintained in SWORD and AIP packaging and be maintained in updates and restorations.

User Interface

General behaviour: Linear Versioning

From the user interface, DSpace offers **linear** versioning. As opposed to hierarchical versioning, linear version has following properties:

- A new version can be created started from any available version but will be always be put at the end of the version history (it will be the latest)
- Only one in-progress version can exist at any time. When new version has been created and still needs to pass certain steps of the workflow, it is temporarily impossible to create another new version until the workflow steps are finished and the new version has replaced the previous one.

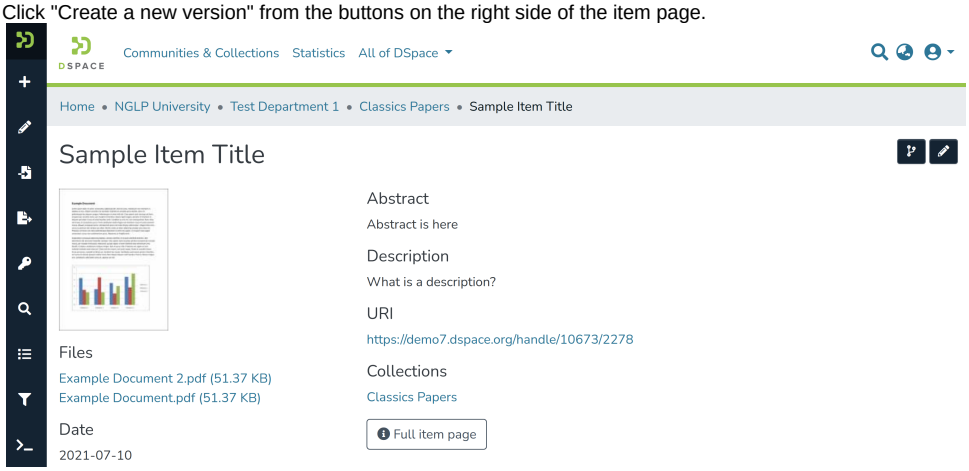
Creating a new version of an item

Administrators, collection/community administrators and eventually the original submitter can create new versions of an item from the Item View page. By default the original submitter is not allowed to create new version but the permission can be granted with the following property

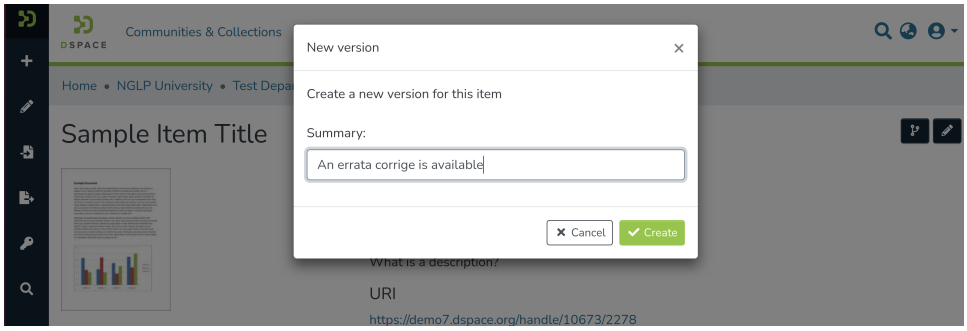
modules/versioning.cfg

```
versioning.submitterCanCreateNewVersion=false
```

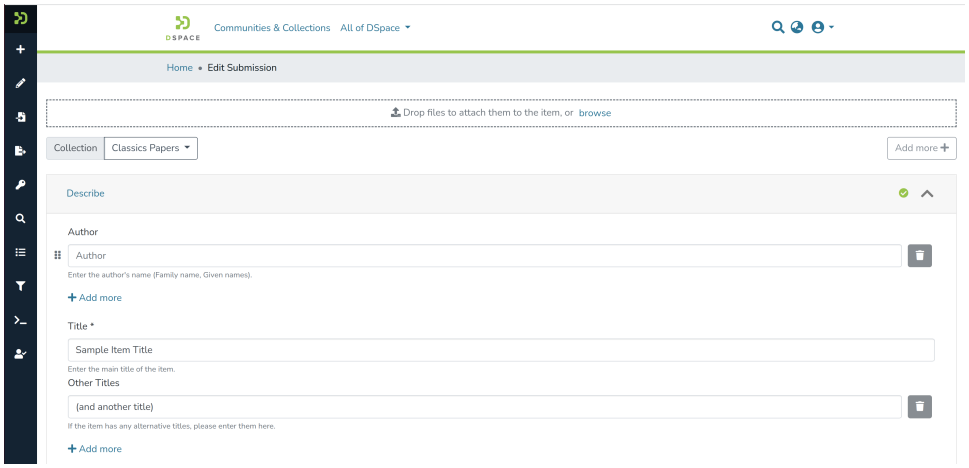
1. Click "Create a new version" from the buttons on the right side of the item page.



2. Provide the reason for creating a new version that will later on be stored and displayed in the version summary.

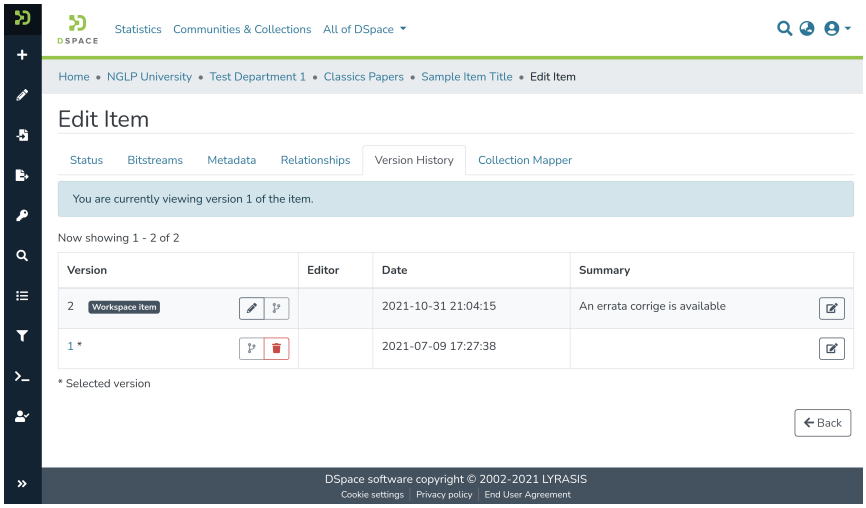


3. Your new version is now creates as a new Item in your Workspace. It requires you to go through the submission and workflow steps like you would do for a normal, new submission to the collection. The rationale behind this is that if you are adding new files or metadata, you will also need to accept the license for them. In addition to this, the versioning functionality does not bypass any quality control embedded in the workflow steps.



After the submission steps and the execution of subsequent workflow steps, the new version becomes available in the repository.

Versions can be also managed via the edit item page, in the dedicated versions tab



View the history and older versions of an item

An overview of the version history, including links to older versions of an item, is available at the bottom of an Item View page. The repository administrator can decide whether the version history should be available to all users or restricted to administrators. By default, this information is available to all users. Information displayed includes the version number, Submitter/Editor name (only if enabled), date, and summary/description. As necessary, you may change the visibility of this table or the "Editor" column using the "Version History Visibility" configurations below.

Version History

You are currently viewing version 2 of the item.

Now showing 1 - 2 of 2

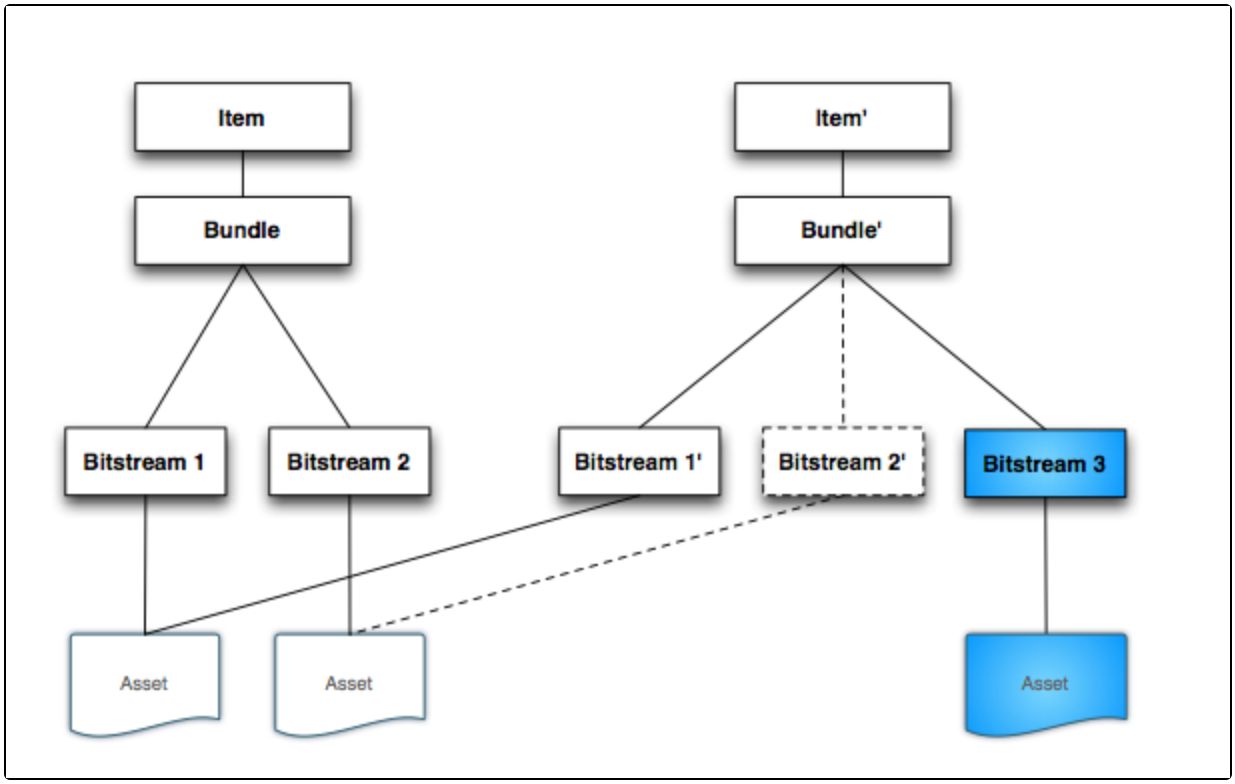
Version	Editor	Date	Summary
2 *		2021-10-31 21:04:15	An errata corrige is available
1		2021-07-09 17:27:38	

* Selected version

Architecture

Versioning model

For every new Version a separate DSpace Item will be created that replicates the metadata, bundle and bitstream records. The bitstream records will point to the same file on the disk.



The *Cleanup* method has been modified to retain the file if another Bitstream record point to it (the dotted lines in the diagram represent a bitstream deleted in the new version), in other words the file will be deleted only if the Bitstream record processed is the only one to point to the file *(count(INTERNAL_ID)=1).*

Configuration

Versioning Service Override

You can override the default behaviour of the Versioning Service using following XML configuration file, deployed under your dspace installation directory:

[dspace_installation_dir]/config/spring/api/versioning-service.xml

In this file, you can specify which metadata fields are automatically "reset" (i.e. cleared out) during the creation of a new item version. By default, all metadata values (and bitstreams) are copied over to the newly created version, with the exception of **dc.date.accessioned** and **dc.description.provenance.** You may specify additional metadata fields to reset by adding them to the "ignoredMetadataFields" property in the "versioning-service.xml" file:

```
<!-- Default Item Versioning Provider, defines behavior for replicating
     Item, Metadata, Budles and Bitstreams. Autowired at this time. -->
<bean class="org.dspace.versioning.DefaultItemVersionProvider">
    <property name="ignoredMetadataFields">
         <set>
            <value>dc.date.accessioned</value>
            <value>dc.description.provenance</value>
         </set>
    </property>
</bean>
```

Identifier Service Override

Persistent Identifiers are used to address Items within DSpace. The handle system is deeply integrated within DSpace, but since version 4 DSpace can also mint DOIs. DSpace 4 and 5 supported one type of versioned handle: The initial version of an Item got a handle, e.g. 10673/100. This handle was called the canonical one. When a newer version was created, the canonical handle was moved to identify the newest version. The previously newest version got a new handle build out of the canonical handle extended by a dot and the version number. In the image below you see a version history of an item using this handle strategy.

The canonical handle will always point to the newest version of an Item. This makes sense if you hide the version history. Normal users won't be able to find older versions and will always see just the newest one. Please keep in mind, that older versions can be accessed by "guessing" the versioned Handle if you do not remove the read policies manually. The downside of this identifier strategy is that there is no permanent handle to cite the currently newest version, as it will get a new Handle when a newer version is created.

With DSpace 6 versioned DOIs (using DataCite as DOI registration agency) were added and the default versioned Handle strategy was changed. Starting with DSpace 6 the `VersionedHandleIdentifierProvider` creates a handle for the first version of an item. Every newer version gets the same handle extended by a dot and the version number. To stay by the example from above, the first version of an Item gets the Handle 10673/100, the second version 10673/100.2, the third version 10673.3 and so on. This strategy has the downside that there is no handle pointing always to the newest version. But each version gets an identifier that can be use to cite exactly this version. If page numbers changes in newer editions the old citations stay valid.

In DSpace 4 and 5 only the strategy using canonical handles (one handle that always points to the newest version) were implemented. In DSpace 6 the strategy of creating a new handle for each version was implemented. With DSpace 6 this new strategy become the default. The strategy using canonical handle still exists in DSpace but you have to enable the `VersionedHandleIdentifierWithCanonicalHandles` in the file `[dspace]/config/spring/api/identifier-service.xml`. With DSpace 6 versioned DOIs were introduced using the strategy that every new version gets a new DOI (extended by a dot and the version numbers for versions >= 2). To use versioned Handle you have to enable DOIs, you have to use DataCite as registration agency and you have to enable the `VersionedDOIIdentifierProvider` in the named configuration file.

You can configure which persistent identifiers should be used by editing following XML configuration file, deployed under your dspace installation directory:

[dspace_installation_dir]/config/spring/api/identifier-service.xml

No changes to this file are required to enable Versioning. This file is currently only relevant if you want to keep the identifier strategy from DSpace 4 and 5 or if you want to enable DOIs or even versioned DOIs.

Version History Visibility

By default, **all** users will be able to see the version history. To ensure that only administrators can see the Version History, enable `versioning.item.history.view.admin` in the `[dspace]/config/modules/versioning.cfg` OR in your `local.cfg` file.

```
# Setting this to "true" will hide the entire "Version History" table from
# all users *except* Administrators
versioning.item.history.view.admin=false
```

Hide Editor/Submitter details in version table

In either the `[dspace]/config/modules/versioning.cfg` configuration file or your `local.cfg`, you can customize the configuration option `versioning.item.history.include.submitter`. By default this is set to false, which means that information about the submitter is only viewable by administrators. If you want to expose the submitters information to everyone (which be useful if all submitters uses generic institutional email addresses, but may conflict with local privacy laws if personal details are included) you can set this configuration property to true.

```
# This property controls whether the name of the submitter/editor
# of a version should be included in the version history of an item.
# Set to "true" to show the submitter to all users who have access to the table.
# Set to "false" to hide the submitter information from everyone except Administrators
versioning.item.history.include.submitter=false
```

Allowing submitters to version their items

With DSpace 6.0 it became possible to allow submitters to create new versions of their own items. The new versions are going through the normal workflow process if the collection is configured this way. To allow submitters to create new versions of Item they originally submitted, you have to change the configuration property `versioning.submitterCanCreateNewVersion` and set it to `true`. It is part of the configuration file `[dspace]/config/modules/versioning.cfg` but can be overridden in your `local.cfg` too.

Identified Challenges & Known Issues

Item Level Versioning has a substantial conceptual impact on many DSpace features. Therefore it has been accepted into DSpace as an optional feature. Following challenges have been identified in the current implementation. As an early adopter of the Item Level Versioning feature, your input is greatly appreciated on any of these.

Conceptual compatibility with Embargo

Lifting an embargo currently does not interact with Item Level Versioning. Additional implementation would be required to ensure that lifting an embargo actually creates a new version of the item.

Conceptual compatibility with Item Level Statistics

Both on the level of pageviews and downloads, different versions of an item are treated as different items. As a result, an end user will have the impression that the stats are being "reset" once a new version is installed, because the previous downloads and pageviews are allocated to the previous version.

One possible solution would be to present an end user with aggregated statistics across all viewers, and give administrators the possibility to view statistics per version.

Bitstream versioning and replacement

Draft

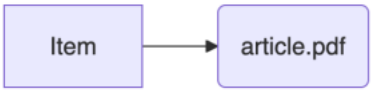
This page contains additional information related to https://github.com/DSpace/DSpace/pull/10554 and https://github.com/DSpace/dspace-angular/pull/4368

Example

In the flowcharts below, direct database relations are shown as solid arrows and metadata links as dashed arrows (with the metadata field specified). A metadata link pointing from Bitstream A to Bitstream B implies that Bitstream A has a metadata value with Bitstream B's UUID in the corresponding metadata field.

Consider a user building up a version history

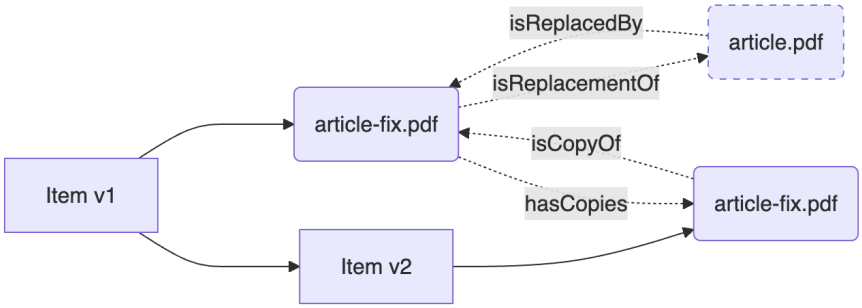
1. Starting with an Item with one Bitstream



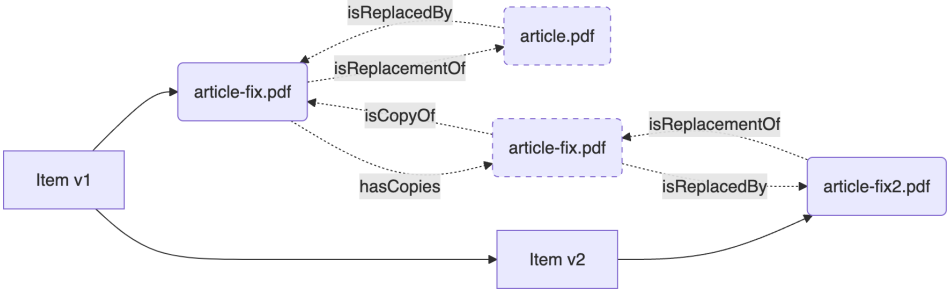
2. The user replaces the Bitstream (either from the submission form or the Bitstream edit page if the Item has been deposited already)
    - The original Bitstream is removed from its Bundle, marked `deleted=true`, but remains in the database; a new Bitstream is created and added to the original Bitstream's Bundle
    - The two Bitstreams are linked together with metadata: `dspace.bitstream.isReplacedBy` and `dspace.bitstream.isReplacementOf`.



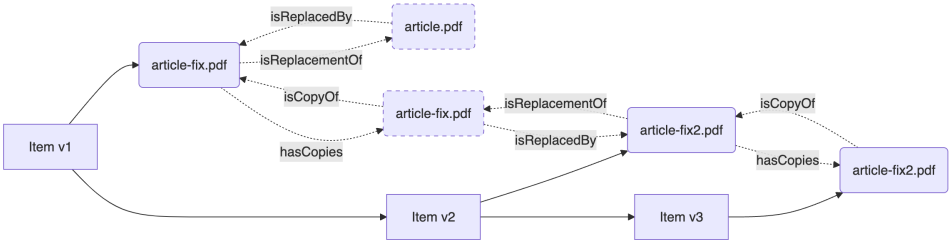
3. The user creates a new version of the Item
    - The Bitstream is copied and linked to the previous version with metadata: `dspace.bitstream.isCopyOf`, `dspace.bitstream.hasCopies`.



4. Next, suppose the user replaces the newly copied bitstream again within that new version of the Item



5. Finally, suppose the user creates another version, thereby copying the newly replaced Bitstream

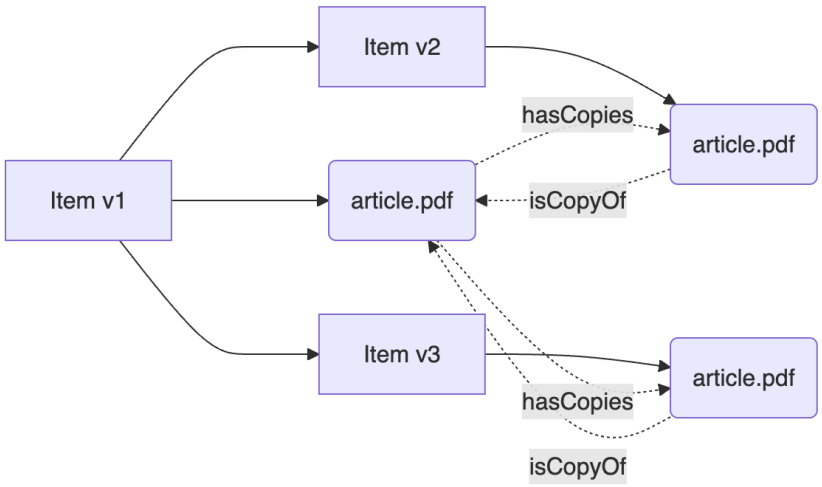


As long as the deleted Bitstreams remain in the database, the version history of this particular file can be traced by the metadata links.

In order to maintain this, DSpace's Bitstream cleanup script should be run with the `--leave` argument in order to retain the Bitstreams and their metadata (their content will still be removed from the bitstore).

DSpace allows a new version of an Item to be created from any point in its version history. For example, you can start both version 2 and 3 from version 1, leading to a "forked" history as shown below.

This means that a Bitstream may have more than one copy (hence the plural used for the `dspace.bitstream.hasCopies` metadata field).



Mapping/Linking Items to multiple Collections

- Introduction
- Using the Item Mapper
- Implications
    - Mapping collection vs Owning collection
    - Mapping an item does not modify access rights

Introduction

The Item Mapper is a tool in the DSpace web user interface allowing repository managers to display the same item in multiple collections at once. Thanks to this feature, a repository manager is not forced to duplicate items to display them in different collections

Using the Item Mapper

In the User Interface, the item mapper can be accessed when editing an Item.

- Login as someone with Edit permissions
- Search/browse to the Item
- Click the "Edit this Item" button
- Click "Mapped" collections" button on the "Status" tab
    - You'll be shown a list of currently mapped collections (if any)
    - You can also map the item to a new collection by clicking on "Map new collections" tab, and searching for the Collection(s)

Implications

Mapping collection vs Owning collection

The relation between an item and the collection in which it is mapped is different from the relation that this item has with the collection to which it was originally submitted. This second collection is referred to as the "owning" collection. When an item is deleted from the owning collection, it automatically disappears from the mapping collection. From within the mapping collection, the only thing that can be deleted is the mapping relation. Removing this mapping relation does not affect the presence of the item in the owning collection.

Mapping an item does not modify access rights

When an item gets mapped into a collection, it does not receive new access rights. It retains the authorizations that it inherited from the collection that "owns" it. Collection admins who do not have read access to an item will not be able to map them to other collections.

Metadata Recommendations

1 Recommended Metadata Fields
2 Local Fields

Recommended Metadata Fields

DSpace provides a broad list of metadata fields out of the box (see: Metadata and Bitstream Format Registries), and a variety of options for adding content to DSpace (both from the UI and from other services). No matter which Ingest option you use, DSpace recommends ensuring that the following metadata fields are specified:

- **Title** (`dc.title`)
  - When submitting an Item via the DSpace web user interface, this field is **required**.
  - If you add an Item to DSpace through another means (SWORD, etc), it is recommend to specify a title for an Item. Without a title, the Item will show up in DSpace a "Untitled".
- **Publication Date** (`dc.date.issued`)
  - When submitting an Item via the DSpace web user interface, this field is **required** (by default).
    - However, your System Administrator can choose to enable the "Initial Questions" step within the Submission User Interface. Enabling this step will cause the following to occur: If the item is said to be "published", then the Publication Date will be required. If the item is said to be "unpublished" then the Publication Date will be auto-set to today's date (date of submission). *WARNING:* Google Scholar has recommended against automatically assigning this "dc.date.issued" field to the date of submission as it often results in incorrect dates in Google Scholar results. See https://github.com/DSpace/DSpace/issues/4850 and https://github.com/DSpace/DSpace/issues/5112 for more details.
  - If you add and Item to DSpace through another means (SWORD, etc), it is recommended to specify the date in which an Item was published, in ISO-8601 (e.g. 2007, 2008-01, or 2011-03-04). This ensures DSpace can accurately report the publication date to services like Google Scholar. If an item is unpublished, you can either chose to leave this blank, or pass in the literal string "today" (which will tell DSpace to automatically set it to the date of ingest)

DSpace will not auto-assign a "dc.date.issued"

As of DSpace 4.0, the system will not assign a "dc.date.issued" when unspecified. Previous versions of DSpace (3.0 or below) would set "dc.date.issued" to the date of accession (dc.date.accessioned), if it was unspecified during ingest.

If you are adding content to DSpace without using the DSpace web user interface, there are two recommended options for assigning "dc.date.issued"

- If the item is previously published before, please set "dc.date.issued" to the date of publication in ISO-8601(e.g. 2007, 2008-01, or 2011-03-04)
- If the item has never been previously published, you may set "dc.date.issued='today'" (the literal string "today"). This will cause DSpace to automatically assign "dc.date.issued" to the date of accession (dc.date.accessioned), as it did previously
  - You can also chose to leave "dc.date.issued" as unspecified, but then the new Item will have an empty date within DSpace.

Obviously, we recommend specifying as much metadata as you can about a new Item. For a full list of supported metadata fields, please see: Metadata and Bitstream Format Registries

Local Fields

You may encounter situations in which you will require an appropriate place to store information that does not immediately fit with the description of a field in the default registry. The recommended practice in this situation is to create new fields in a separate schema. You can choose your own name and prefix for this schema such as *local.* or *myuni.*

It is generally discouraged to use any of the fields from the default schema as a place to store information that doesn't correspond with the fields description. This is especially true if you are ever considering the option to open up your repository metadata for external harvesting.

Moving Items

1 Moving Items via Web UI
2 Moving Items via the Batch Metadata Editor

Moving Items via Web UI

It is possible for Administrators to move items one at a time from the User Interface.

- Login as an Administrator
- Browse/search for the item.
- Click "Edit this Item" on the item page
- When editing an item, on the 'Edit item' screen, click the "Move..." button.
- Search for the new Collection for the item to appear in. By default, 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 policies' 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 policies' option, ensure that this will not override system-managed authorizations such as those imposed by the embargo system.

Moving Items via the Batch Metadata Editor

Items may also be moved in bulk by using the CSV batch metadata editor (see Editing Collection Membership section under Batch Metadata Editing).

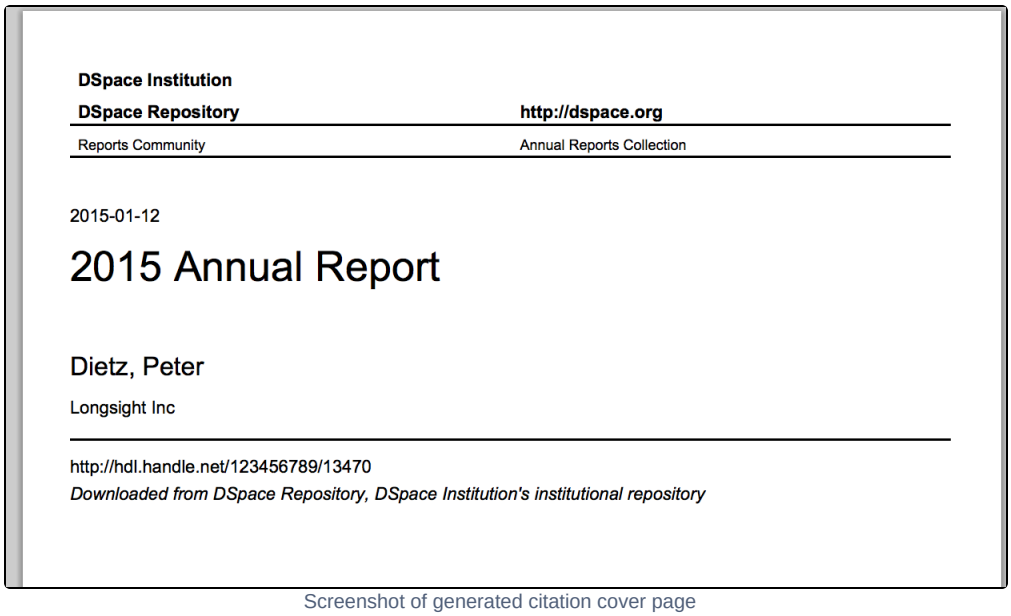
PDF Citation Cover Page

Enabling PDF Cover Pages may affect your site's visibility in Google Scholar (and similar search engines)

Google Scholar specifically warns against automatically generating PDF Cover Pages, as they can break the metadata extraction techniques used by their search engine. Be aware that enabling PDF Cover Pages may also cause those items to no longer be indexed by Google Scholar. For more information, please see the "Indexing Repositories: Pitfalls and Best Practices" talk from Anurag Acharya (co-creator of Google Scholar) presented at the Open Repositories 2015 conference.

Adding a cover page to retrieved documents from DSpace that include additional citation information has been sought, as documents uploaded to the repository might have had their context stripped from them, when they are just a PDF. Context that might have surrounded the document would be the journal, publisher, edition, and more. Without that information, the document might just be a few pages of text, with no way to piece it together. Since repository policy might be to include this information as metadata to the Item, this metadata can be added to the citation cover page, so that the derivative PDF includes all of this information.

The citation cover page works by only storing the original PDF in DSpace, and then generating the citation-cover-page PDF on-the-fly. An alternative set up would be to run the PDF Citation Coverpage Curation Task on the DSpace repository contents, and then disseminate the pre-generated citation-version instead of generating it on the fly.



Screenshot of generated citation cover page

Configuration settings for Citation Cover Page

Configuration file renamed to citation-page.cfg and configurations names have changed

As of DSpace 6.0, the configuration file for this feature was **renamed** from `disseminate-citation.cfg` to `citation-page.cfg`. The renaming was to clarify the purpose of this configuration file, as its previous name was misleading / confusing to some users.

In addition, all configurations below have now been prefixed with "citation-page" (e.g. the `enable_globally` configuration has been renamed to `citation-page.enable_globally`)

As of DSpace 9.0, a Thymeleaf templating engine was implemented to give greater control over layout and fields/metadata used in cover pages. The header, fields, and footing configuration properties were removed from `citation-page.cfg` as this work is now done in `dspace-api/src/main/resources/dspace_coverpage.html`

In the `{dspace.dir}/config/modules/citation-page.cfg` file review the following fields to make sure they are uncommented:

Property:	citation-page.enable_globally
Example Values:	citation-page.enable_globally = true
Informational Note:	Boolean to determine is citation-functionality is enabled globally for entire site. This will enable the citation cover page generator for all PDFs. Default: disabled

Property:	citation-page.enabled_collections
Example Values:	citation-page.enabled_collections = 1811/123, 1811/234
Informational Note:	List of collection handles to enable the cover page generator for bitstreams within. Default: blank
Property:	**citation-page.enabled_communities**
Example Values:	citation-page.enabled_communities = 1811/222, 1811/333
Informational Note:	List of community handles to enable the cover page generator for bitstreams within. Default: blank
Property:	**citation-page.citation_as_first_page**
Example Values:	citation-page.citation_as_first_page = true
Informational Note:	Should the citation page be the first page cover (true), or the last page (false). Default: true, (first page)
Property:	**citation-page.page_format**
Example Values:	citation-page.page_format = LETTER
Informational Note:	A4 or LETTER or another valid CSS3 page size. See https://flyingsaucerproject.github.io/flyingsaucer/r8/guide/users-guide-R8.html#xil_34 Default: LETTER
Property:	**citation-page.cover-template**
Example Values:	citation-page.cover-template = dspace_coverpage
Informational Note:	The name of the HTML Thymeleaf template to use when generating cover pages. The template name is converted to a Thymeleaf resource e.g. `dspace_coverpage` will reference `dspace-api/main/resources/dspace_coverpage.html` Default: dspace_coverpage

Layout and metadata settings for Citation Cover Page

As of DSpace 9.0, the cover page is generated with an HTML templating engine. This allows for more flexible header, content, and footer layout.

The template uses the Thymeleaf templating language.

The default cover page template is in `dspace-api/main/src/resources/dspace_coverpage.html` and can be overridden using configuration properties as above.

The header, body, and footer content from the default template can be seen below. The metadata availability comes from the `CoverPageContributor` implementations (e.g. `DefaultCoverPageContributor`)

Variables can be referenced with `${variable_name}`

```
<div class="page">
    <div class="header">
        <img th:src="@{classpath:dspace_logo.png}" src="dspace_logo.png"/>

        <div>
            <a th:href="@{https://dspace.org/}" href="https://dspace.org">
                https://dspace.org/
            </a>
        </div>
    </div>

    <div class="body">
        <div class="metadata">
            <!-- The metadata fields from the current item can be referenced here in the format
schema_field_qualifier -->
            <div class="title" th:text="${metadata_title}">Test title</div>

            <div class="author" th:text="${metadata_author}">Author 1; Author 2; Author 3</div>

            <div class="editor" th:text="${metadata_editor}">Editor 1; Editor 2; Editor 3</div>

            <div class="issued" th:text="${dc_date_issued}">2017</div>

            <div class="publisher" th:text="${dc_publisher}">Publisher Name</div>

            <div class="identifier">
                <a th:href="${dc_identifier_uri}" href="${dc_identifier_uri}" th:text="${dc_identifier_uri}"
>https://hdl.handle.net/123456789/12345</a>
            </div>

            <div class="citation" th:text="${dc_identifier_citation}">Author 1 (2017), Test title. Publisher
Name. Retrieved from https://hdl.handle.net/123456789/12345</div>
        </div>
    </div>

    <div class="footer">
        Downloaded from a DSpace repository.

        <a th:href="@{https://dspace.org}" href="https://dspace.org">
            https://dspace.org/
        </a>
    </div>
```

Request Withdrawn and Reinstate of an item

- Overview
  - Enabling this Feature
- Withdrawal
  - Submitter
  - Admin
- Reinstate
  - Submitter
  - Admin

Overview

The implementation enables all the authenticated users to create WITHDRAWN and REINSTATE requests for a deposited item using the quality assurance event mechanism, which is handled by administrators and specific groups.

Once a WITHDRAWN or REINSTATE request is submitted, it can also be cancelled.

To enable Request Withdrawn and Reinstate of an item, you MUST first enable the following:

- Quality Assurance

Enabling this Feature

This feature is only enabled to Administrators by default. However, if you wish to allow additional users to make these requests for withdrawal or reinstatement of an Item, you can specify the Group of users who is allowed to make these requests.

If you want to allow **all authenticated users** to have this feature, you can configure this setting to use the "Anonymous" group:

```
# Setting this to "Anonymous" allows any authenticated user to submit a request.
# NOTE: It will NOT allow Anonymous users to submit the request as all requests MUST be connected to an EPerson
account.
qaevents.withdraw-reinstate.group = Anonymous
```

in the `local.cfg` file.

Once enabled, when you are logged in you will see a "Request Withdrawal" or "Request Reinstatement" button on Item pages (depending on their current status). See example screenshots below.

Withdrawal

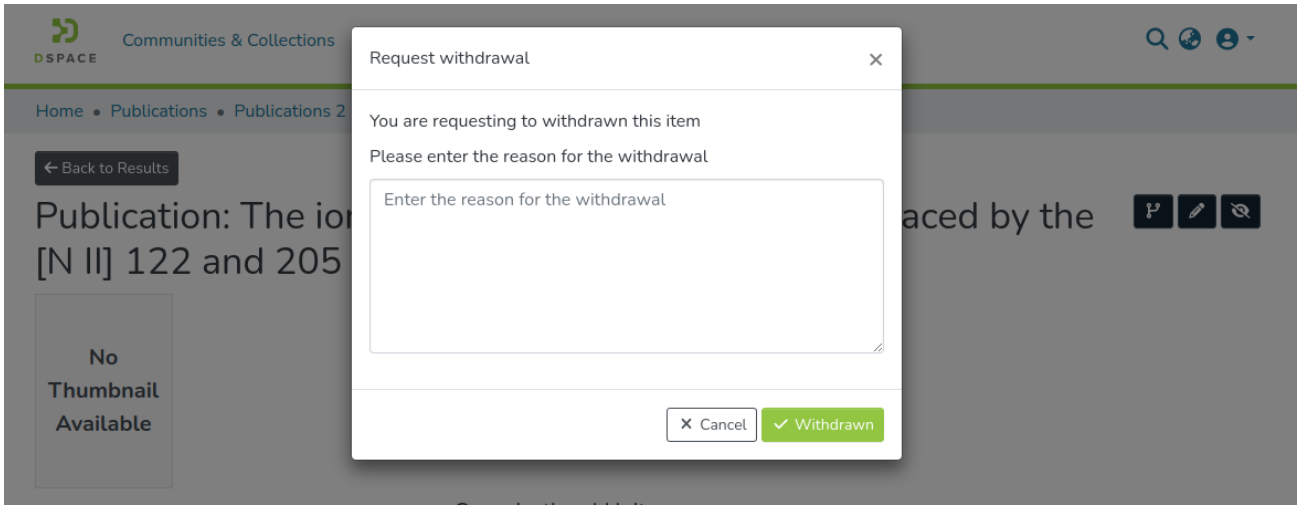
Submitter

The user can withdraw an item from the details page of the archived item. Any authenticated user has the permission to withdraw an item.

Click on the eye icon to start the withdrawal of the item.



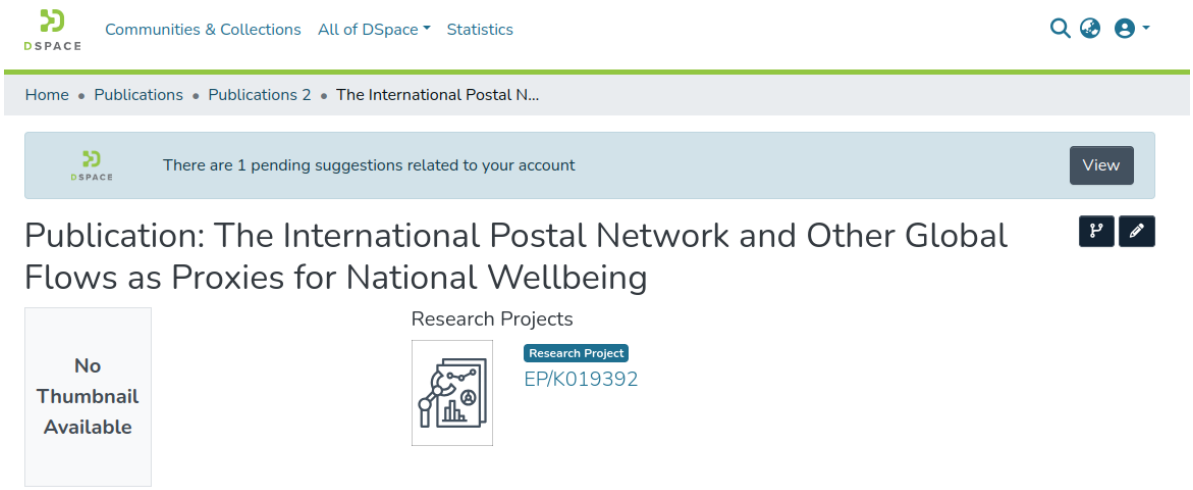
Following that, describe the reason for the withdrawal and subsequently confirm it.



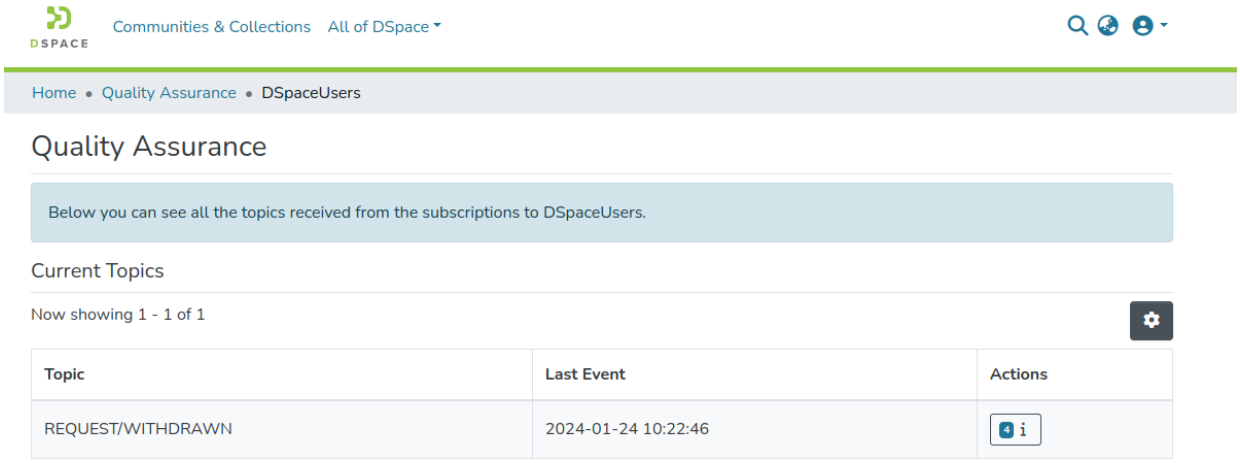
An automatic notification will appear in a notification box, indicating that a withdrawal has been requested for this particular item.

If the withdrawal request is approved, the user will be notified and will be given the opportunity to reinstate the request.

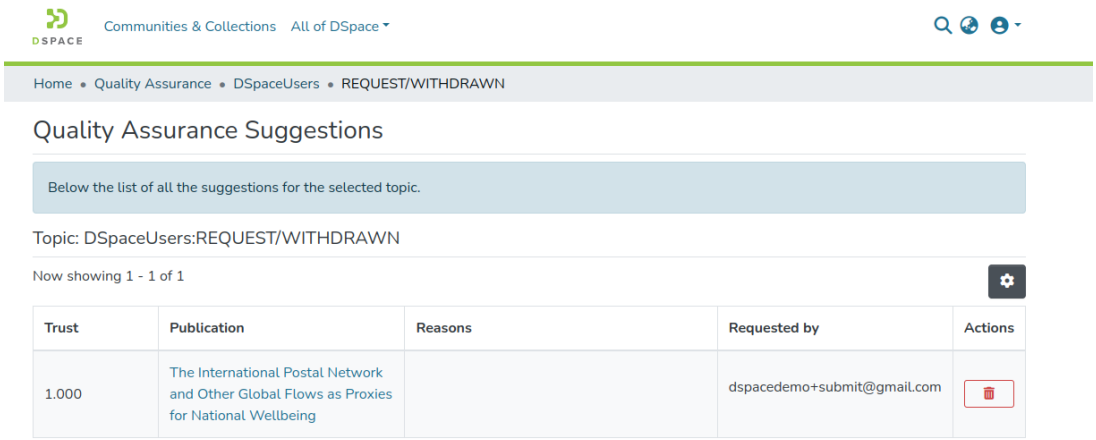
Clicking on *View* on the item page will lead to the "Quality Assurance" page, which lists the topics/requests. In case there is only one topic available, it will automatically redirect to the item list.



The action button displays the count of elements associated with a specific topic. Clicking on it will lead to a different view of the list of items, tailored based on the user's role.

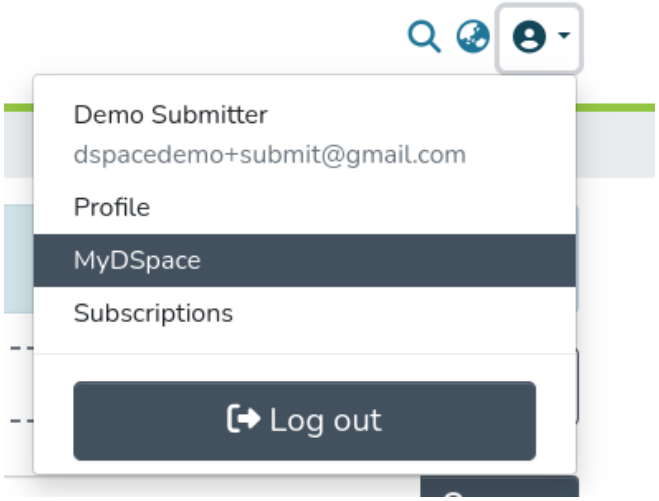


Non-admin users, for example, a submitter, only have the authority to reject the withdrawal request.

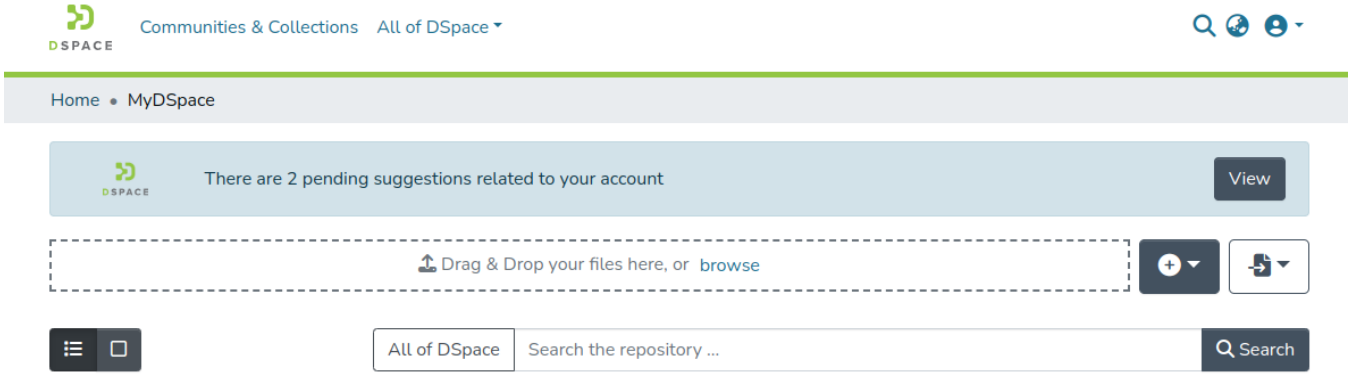


Admin

Admin users have the possibility to access all the requests for all the items from MyDspace.

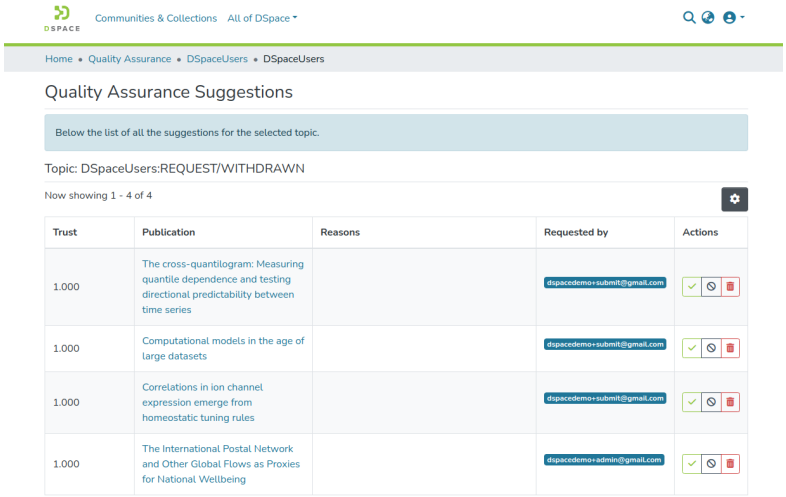


A message will warn the administrator about the requests to check. The *View* button redirects to the "Quality Assurance" page.



The "Quality Assurance" page lists the topics/requests, with the action button displaying the number of elements associated with a specific topic.

The action button will lead to a view of the list of items, with actions based on the user's role.



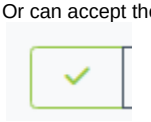
An administrator can reject the withdrawal.



Can ignore it, the request can be reconsidered later



Or can accept the request, and this item will not be discoverable and will not be archived. It can only be accessed through the direct link.

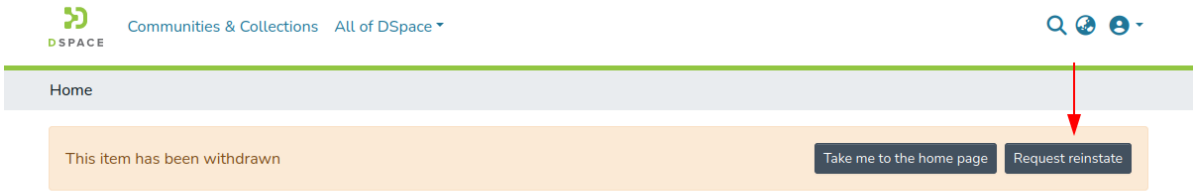


Reinstate

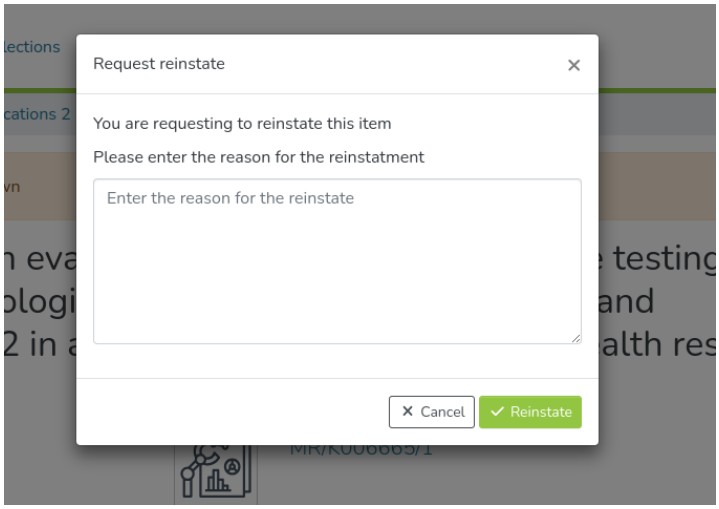
Submitter

After the withdrawal request is approved, the user can request to reinstate the item.

In particular, the submitter will see a “Reinstate” button available next to the “Take me to the home page” button on the withdrawn item page and will no longer see the item's details.

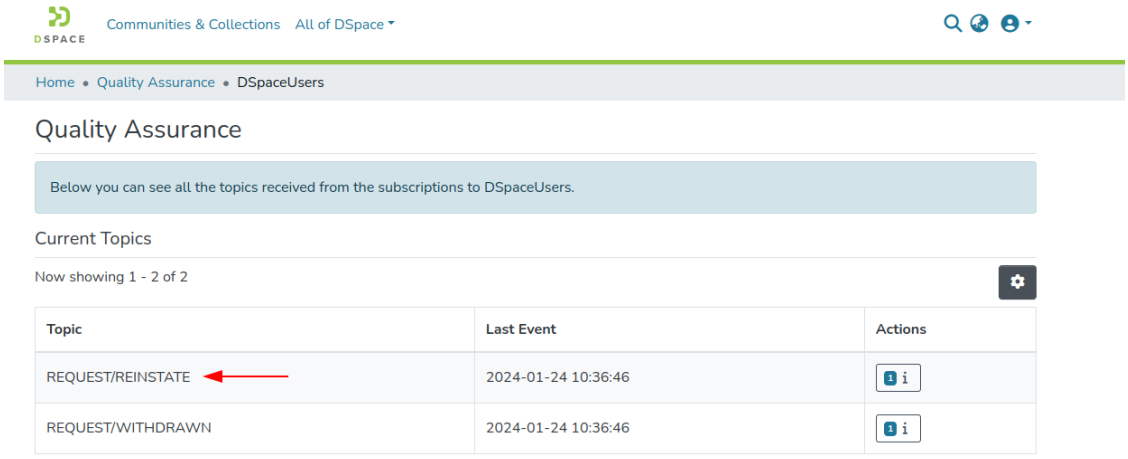


To request the reinstatement, optionally provide a reason and confirm it.



Clicking on *View* on the item page will lead to the "Quality Assurance" page, which lists the topics/requests. In case there is only one topic available, it will automatically redirect to the item list.

In this case, the Reinstate topic will list all the reinstate requests.

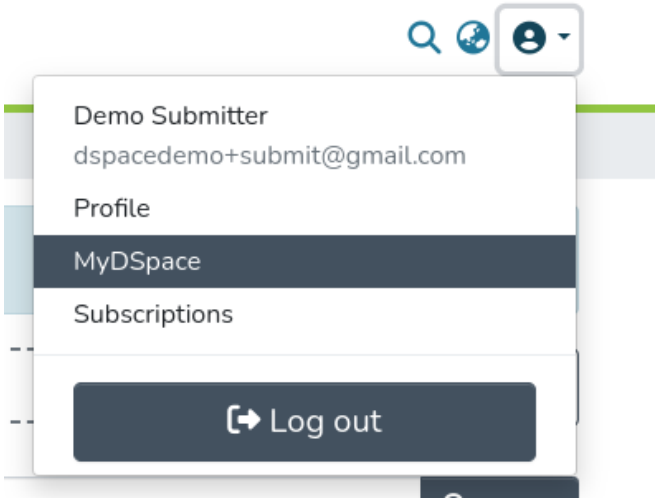


The "Quality Assurance" page lists the topics/requests, with the action button displaying the number of elements associated with a specific topic. In case there is only one topic available, it will automatically redirect to the item list ("accept/ignore/reject" page).

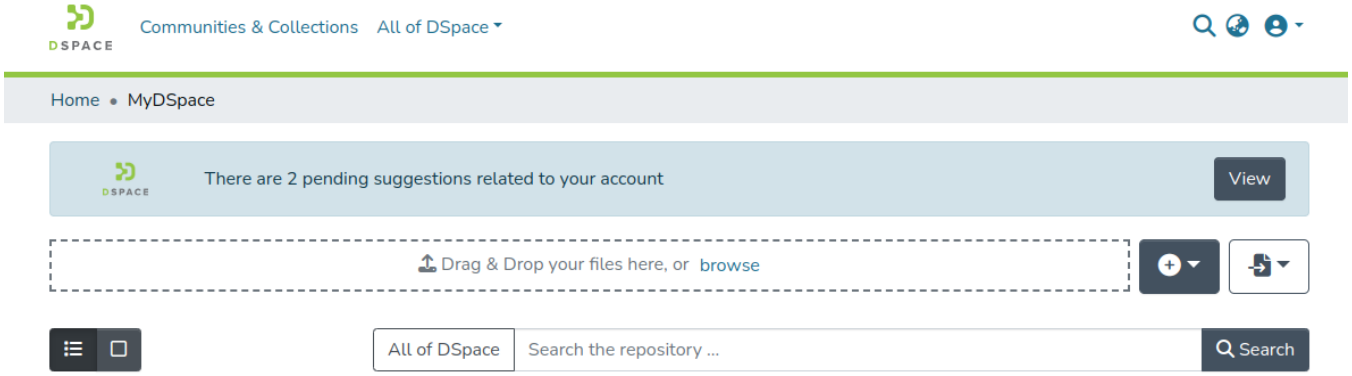
The action button will lead to a view of the list of items, with actions based on the user's role.

Admin

Like in the case of withdrawal, administrators can access all the requests for all the items from MyDspace.

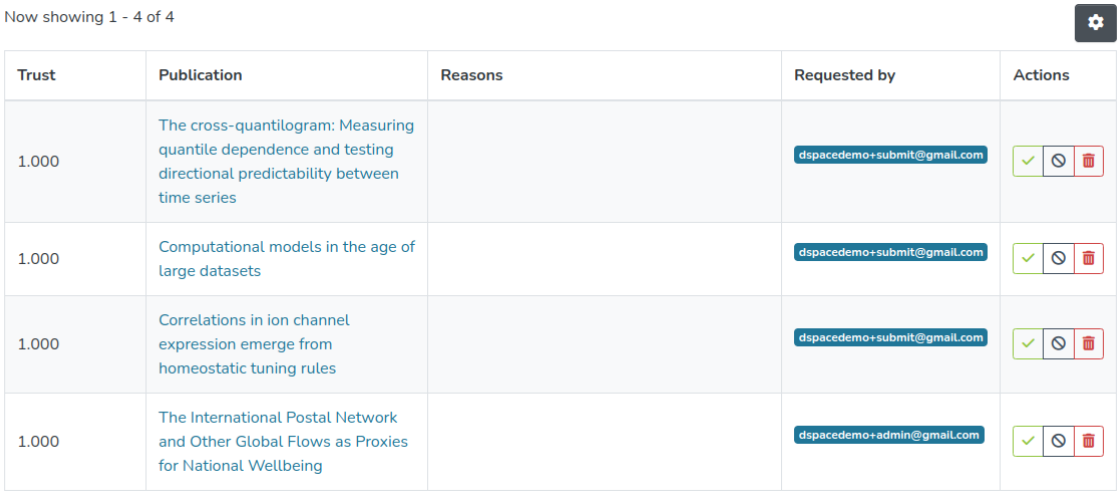


A message will warn the administrator about the requests to check. The *View* button redirects to the "Quality Assurance" page.



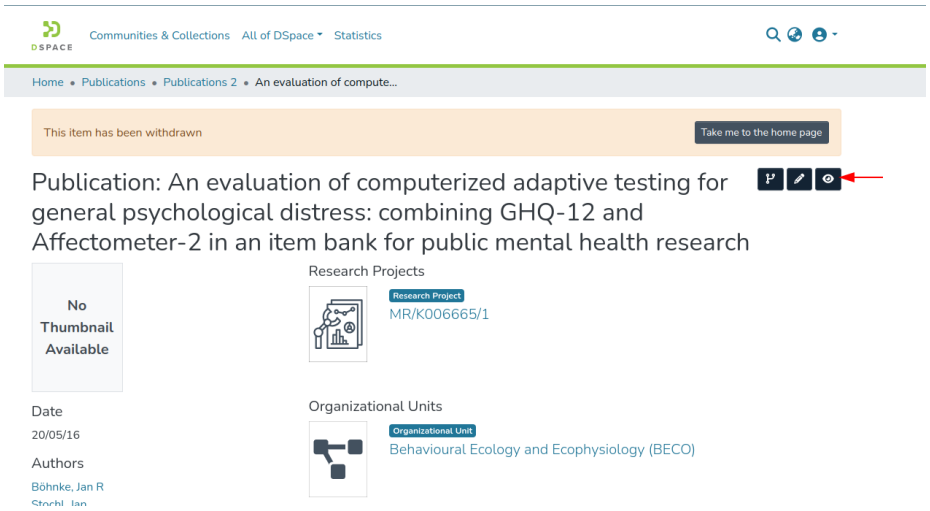
The "Quality Assurance" page also lists the Reinstate topic. The possible actions are:

- reject the reinstatement.
- ignore it, and the request can be reconsidered later.
- accept the reinstatement, and the item will be discoverable again.



Administrators can also start reinstatement requests for all the withdrawn items.

On the page of a withdrawn item, the administrator can use the eye icon on the top right to proceed with a request for reinstatement.



Updating Items via Simple Archive Format

1 Item Update Tool
1.1 DSpace Simple Archive Format
1.2 ItemUpdate Commands
1.2.1 CLI Examples

Item Update Tool

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

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

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

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

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

DSpace Simple Archive Format

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

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

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

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

ItemUpdate Commands

Command used:	`[dspace]/bin/dspace itemupdate`
Java class:	*org.dspace.app.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).

Managing Community Hierarchy

1 Sub-Community Management

Sub-Community Management

Reindex content for new permissions to take effect

After moving or changing an existing Community hierarchy, it is important to reindex your content. Moving a Community under a new parent may result in the inheritance of new/different permissions from that new parent Community. These new permissions will not take effect until you reindex your content. Keep in mind, you may not need to reindex all content, but may be able to simply reindex the content under the new parent Community.

```
./dspace index-discovery -i [new-parent-uuid]
```

DSpace provides an administrative tool, 'CommunityFiliator', for managing community sub-structure. 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.

Nested metadata in DSpace items

In some cases, you need to store groups of related information, such as the author and their affiliation as they appear in the publication “header”, the project title, grant number, funding program related to the publication, or the journal title, ISSN, EISSN, etc.

In the first two examples, the group of metadata is repeatable. This means that the first value in the “author names” must be associated with the first value in the “affiliations”, and so on.

Generally, in DSpace, such scenarios are handled by using **additional entities linked to the item**. This is, of course, the proper way to manage this use case, but supporting additional entities in the system is not always an option or feasible in all contexts. In addition, you may want to allow the submitter to enter rich, structured information during item submission, to be stored or later consolidated into a separate entity.

In other cases, even when the information refers to external entities—such as in the example of author affiliations—this information also needs to be stored or “frozen” at the item level. For instance, you may be interested in knowing which institutions are listed as affiliations in a publication, regardless of the institutions currently linked to each author. Even assuming that you can precisely track an author’s affiliations over time (a complete career history), it would still not be possible to determine with certainty which institution the author chose to list in the publication signature. This is because the relevant date may be partially uncertain, and because the author may have selected one institution over another based on which actually supported the work described in the publication.

To meet these requirements, DSpace allows you to create groups of **nested metadata** in both the submission form and the item page.

The default group in the submission form is rendered as a nested form and is opened in a modal dialog, in order to isolate the form and define relationships between groups of metadata.

- Required Configuration
  - 1. Define a Step in item-submission.xml
  - 2. Define Grouped Forms in submission-forms.xml
    - Child form (publicationStepGroup-dc-contributor-author):
  - Inline-group group type

Required Configuration

To use nested metadata groups, update your `submission-forms.xml` and `item-submission.xml`:

1. Define a Step in item-submission.xml

```
<step-definition id="publicationStepGroup" mandatory="true">
    <heading>submit.progressbar.describe.stepone</heading>
    <processing-class>org.dspace.app.rest.submit.step.DescribeStep</processing-class>
    <type>submission-form</type>
</step-definition>
```

Then reference it in a submission process:

```
<submission-process name="publication-submission">
    <step id="collection"/>
    <step id="publicationStepGroup"/>
</submission-process>
```

2. Define Grouped Forms in submission-forms.xml

For `group` input type (used with relation-field), create a main form and child forms following the naming convention `formName-dc-element-qualifier`:

Main form (`publicationStepGroup`):

```
<form name="publicationStepGroup">
    <row>
        <relation-field>
            <relationship-type>isAuthorOfPublication</relationship-type>
            <search-configuration>person</search-configuration>
            <repeatable>true</repeatable>
            <label>Author</label>
            <hint>Enter the author's name.</hint>
            <linked-metadata-field>
                <dc-schema>dc</dc-schema>
                <dc-element>contributor</dc-element>
                <dc-qualifier>author</dc-qualifier>
                <input-type>group</input-type>
            </linked-metadata-field>
            <externalsources>orcid</externalsources>
        </relation-field>
    </row>
</form>
```

Child form (`publicationStepGroup-dc-contributor-author`):

```
<form name="publicationStepGroup-dc-contributor-author">
    <row>
        <field>
            <dc-schema>dc</dc-schema>
            <dc-element>contributor</dc-element>
            <dc-qualifier>author</dc-qualifier>
            <label>Author</label>
            <input-type>onebox</input-type>
            <repeatable>false</repeatable>
            <required>You must enter at least the author.</required>
            <hint>Enter the names of the authors.</hint>
        </field>
    </row>
    <row>
        <field>
            <dc-schema>oairecerif</dc-schema>
            <dc-element>author</dc-element>
            <dc-qualifier>affiliation</dc-qualifier>
            <label>Affiliation</label>
            <input-type>onebox</input-type>
            <repeatable>false</repeatable>
            <required />
            <hint>Enter the affiliation of the author.</hint>
        </field>
    </row>
</form>
```

For `inline-group` input type (simple field grouping):

```
<form name="publicationStepGroup">
    <row>
        <field>
            <dc-schema>dc</dc-schema>
            <dc-element>contributor</dc-element>
            <dc-qualifier>editor</dc-qualifier>
            <repeatable>false</repeatable>
            <label>Editor</label>
            <input-type>inline-group</input-type>
            <hint>The editors of this publication.</hint>
            <required>You must enter at least the author.</required>
        </field>
    </row>
</form>
```

Note: Child forms must be named using the pattern: `{parentFormName}-{dc-element}-{dc-qualifier}` for the group input type to resolve correctly.

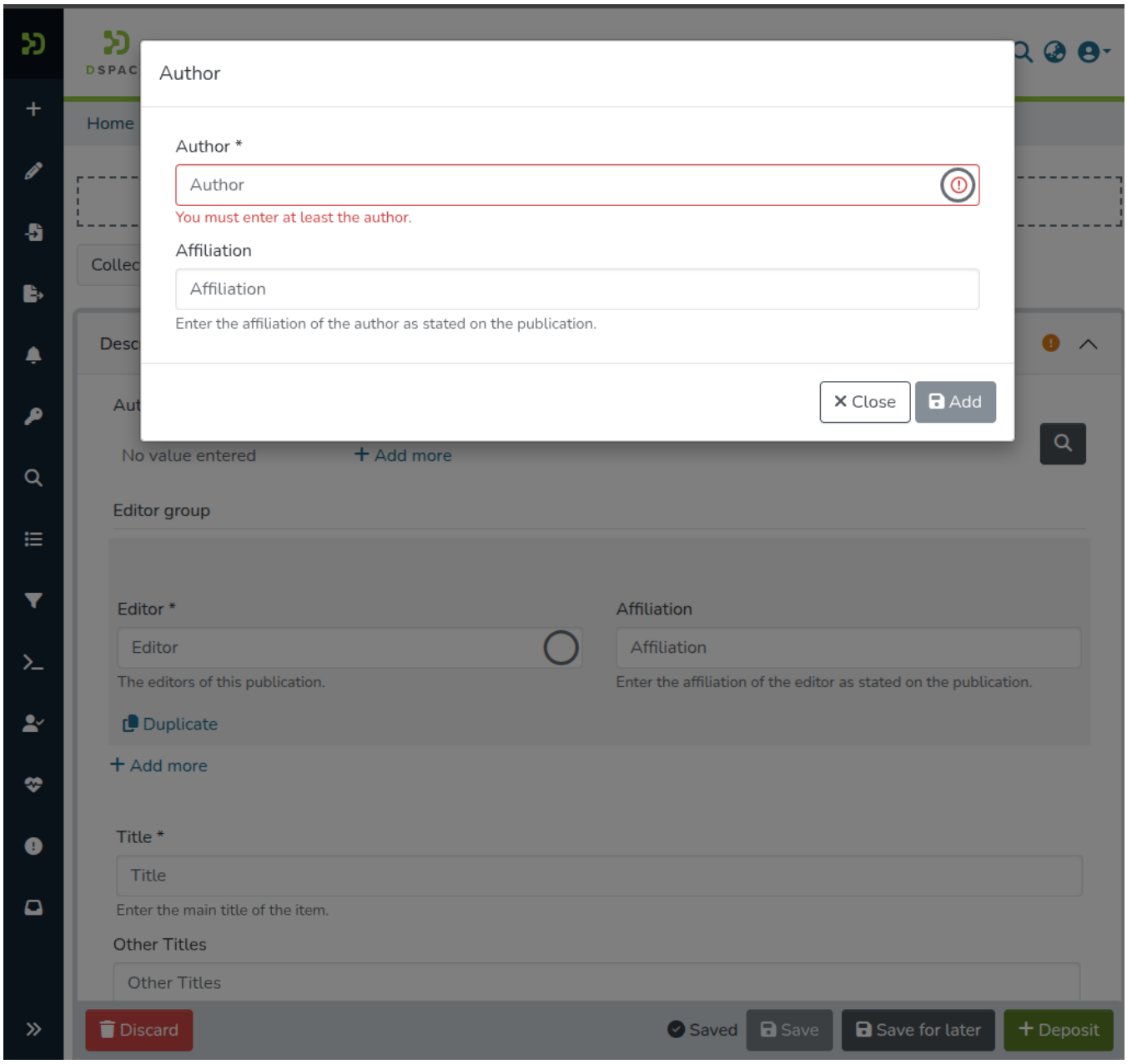
Inline-group group type

A new group type called inline-group has been added to mantain the groups inside the main form and not in a modal, follows and example of config:

Please note that the inline-group as a limitation: all the fields needs to be configured inside one tag.
If you want to achieve a display on multiple rows, it can be don through styling, adding for example to each field the style tag as follows: <style>col-12</style>
This is a known limitation and should be added to the documentation as such.

```
 <row>
              <field>
                <dc-schema>dc</dc-schema>
                <dc-element>contributor</dc-element>
                <dc-qualifier>editor</dc-qualifier>
                <label>Editor group</label>
                <input-type>inline-group</input-type>
                <repeatable>true</repeatable>
                <required />
                <hint>Add the editor of this publication</hint>
              </field>
</row>

<form name="publicationStep-dc-contributor-editor">
          <row>
            <field>
              <dc-schema>dc</dc-schema>
              <dc-element>contributor</dc-element>
              <dc-qualifier>editor</dc-qualifier>
              <label>Editor</label>
              <input-type>onebox</input-type>
              <repeatable>false</repeatable>
              <required>You must enter at least the author.</required>
              <hint>The editors of this publication.</hint>
            </field>
            <field>
              <dc-schema>oairecerif</dc-schema>
              <dc-element>editor</dc-element>
              <dc-qualifier>affiliation</dc-qualifier>
              <label>Affiliation</label>
              <input-type>onebox</input-type>
              <repeatable>false</repeatable>
              <required />
              <hint>Enter the affiliation of the editor as stated on the publication.</hint>
            </field>
          </row>
</form>
```



Please note that is not possible to add repeatable fields inside the nested forms, this would result in errors in the UI breaking the form. This is a known limitation and should be added to the documentation as such.

Input type specific configuration, as the regex, should be added inside the nested form configuration, as they don't work on input types which represent nested forms. (not an actual control in the form).

Note: For inline groups is possible to configure an additional button with the property showInlineGroupDuplicateButton , which if true, will display under the form a button to duplicate the whole section, metadata values included.

ORCID Integration

Since DSpace 7.3 a bidirectional ORCID integration is available for DSpace. This feature allows for authentication via ORCID, as well as synchronizing data between ORCID and DSpace, via the usage of Researcher Profiles.

Acknowledgments

The ORCID integration was originally developed by 4Science in DSpace-CRIS. It is the result of years of collaboration with several institutions and the ORCID team that has helped to correct, improve and broaden the scope of the integration. We want to thank the University of Hong Kong that was the first institution to fund development activities in this regard back in 2015 and the TUHH Hamburg University of Technology that have funded the initial porting of the ORCID integration to the new Angular/REST architecture introduced in DSpace 7. Last but not least, funds have been received by the DSpace community to port this feature from DSpace-CRIS to DSpace.

- Overview
- User features
  - Login via ORCID
  - Connect/Disconnect the local profile to ORCID
  - Import publications from ORCID
  - Import a person from ORCID
  - Displaying ORCID iDs on work landing pages
- Configuration
  - Enable the integration
  - Configure the push of information from DSpace to ORCID
    - Mapping of the DSpace Person Items to ORCID Works
    - Mapping of DSpace Publication items to ORCID Works
    - Mapping of DSpace Project items to ORCID Funding
  - Configure the import features
  - Configure the author lookup in submission
- Troubleshooting & common issues
  - I'm having trouble testing the ORCID integration. What should I check?
  - I cannot find the ORCID features described by this page in my installation
  - I'm unable to authenticate via ORCID
  - After logging in via ORCID, a new DSpace account was created instead of using my existing DSpace account
  - I'm having trouble creating test accounts on ORCID to experiment with the features
  - I have configured my Public ORCID API credentials in DSpace but I get an error attempting to login via ORCID
  - I don't find my publications looking up for my ORCID iD
  - I cannot push all my publications, only few or none of them are listed in the queue
  - Push of publications to ORCID fails
  - Push of projects to ORCID fails

Overview

DSpace provides a bidirectional integration with ORCID based on the ORCID API v3.0. Both the Public ORCID API and the Membership API are supported.

The table below summarizes the supported features according to the type of ORCID API configured.

Feature	No credentials*	Public API	Member API
Authentication		✓	✓
Connect local profile to ORCID (authenticated ID)		✓	✓
ORCID Registry Lookup - import Person records	✓	✓	✓
ORCID Registry Lookup - as authority	✓	✓	✓
Import publication from ORCID	✓	✓	✓
Push biographic data to ORCID			✓
Push publications to ORCID (works)			✓
Push projects to ORCID (fundings)			✓

*** No credentials:** please note that ORCID strongly recommends to apply at least for a free public API Key as this will help to trouble-shoot integration problems and get support from ORCID. There is also a chance to get better performance/priority over "unknown" client.

User features

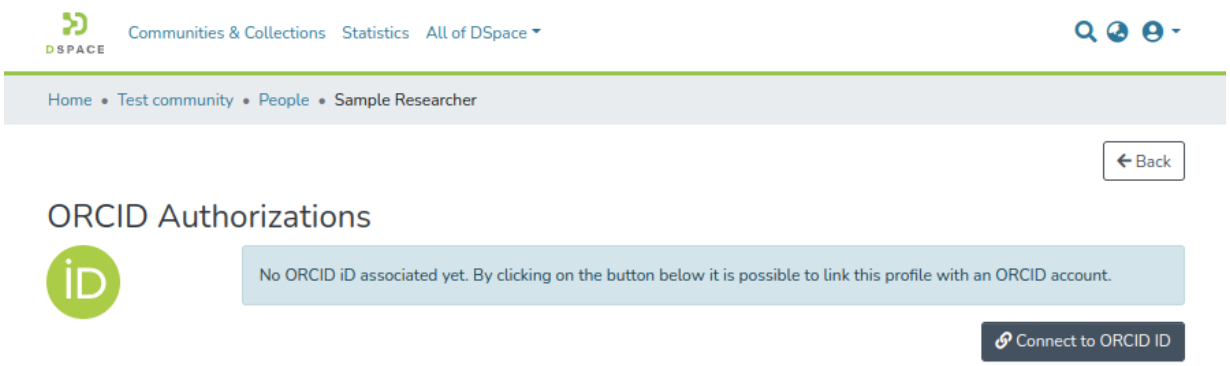
Login via ORCID

Once enabled, an option to login via ORCID is provided to the user among the other authentication methods configured in the system. There are several different scenarios that need to be considered for the ORCID Login process; please see ORCID Login: Guide to Repository access and Account linking for more information. The ORCID authentication doesn't allow the user to reset their password from DSpace.

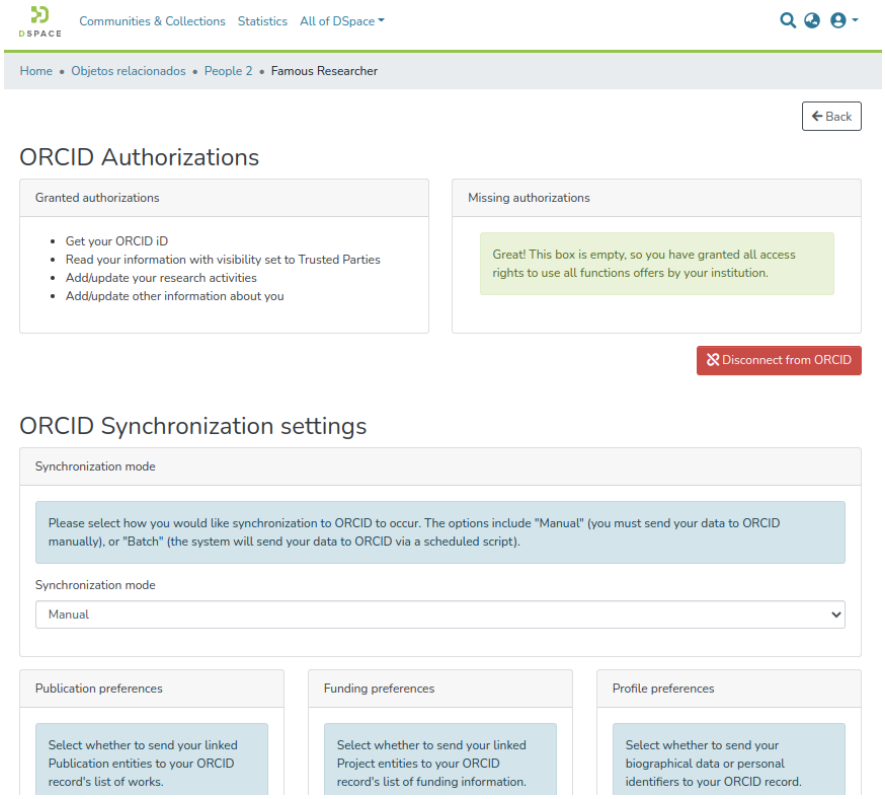


Connect/Disconnect the local profile to ORCID

The researcher can connect (or disconnect) their DSpace local Researcher Profile with ORCID from the Person item detail page.



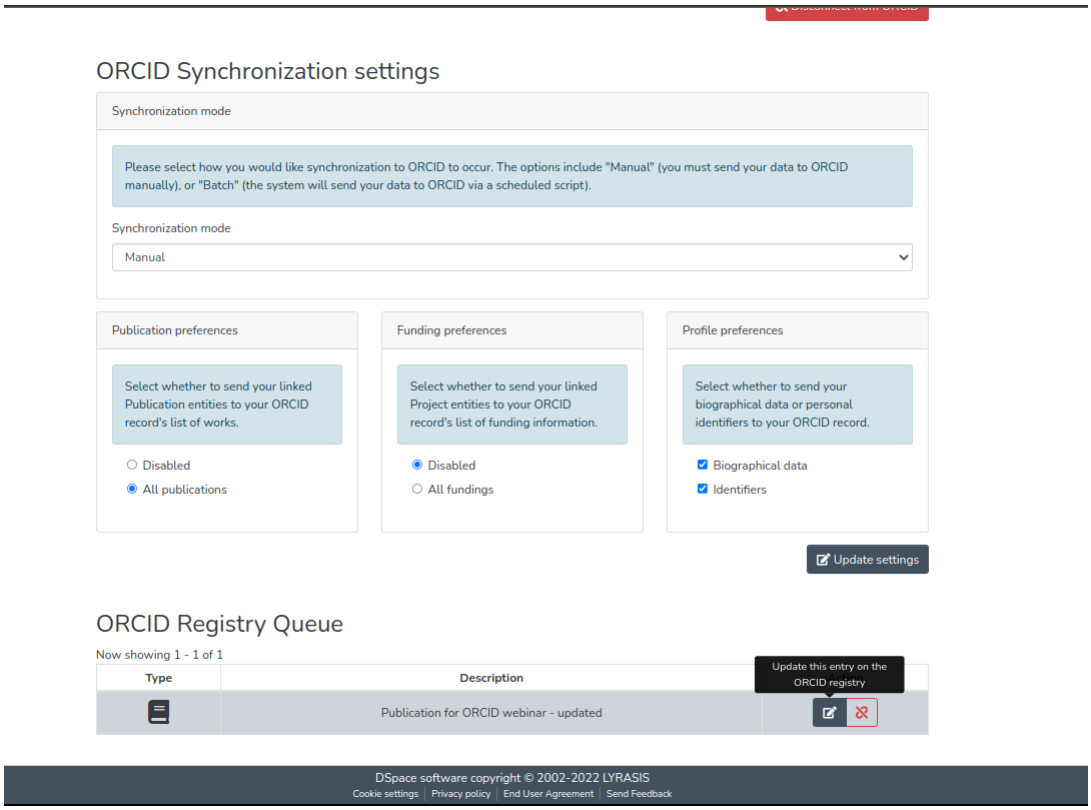
If a user clicks the "Disconnect from ORCID" button, their ORCID access token will be revoked and the DSpace repository will no longer be a "trusted party" on the user's ORCID account, according to ORCID's best practices.



Once a profile has been connected they can manage their synchronization preferences deciding what should be pushed to ORCID, including:

- biographic data
- Publication (entities) linked with their Researcher Profile. (Publication entities are synced to Works in ORCID.)
- Project (entities) linked with their Researcher Profile. (Project entities are synced to Fundings in ORCID.)
- Code for pushing Product and Patent information to ORCID is available for organizations who wish to use it; see: https://github.com/DSpace/DSpace/pull/9853

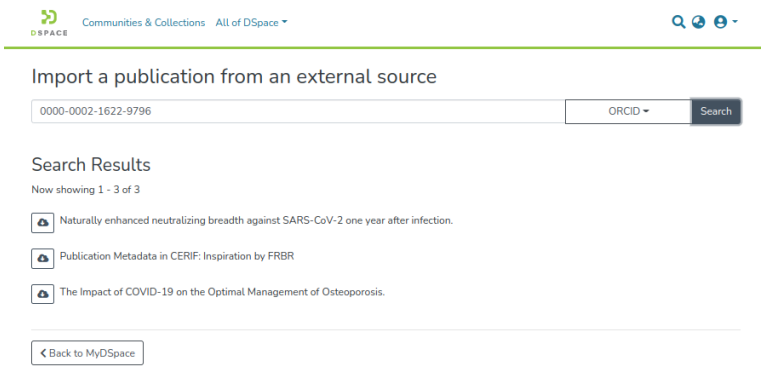
NOTE: The ORCID synchronization feature is disabled by default, even when ORCID Authentication is enabled. See Configuration section below for how to enable it.



The synchronization can happen automatically over the night or manually. The list of information that should be pushed or updated from DSpace to ORCID is presented in a queue and can be manually discarded or immediately pushed by the researcher.

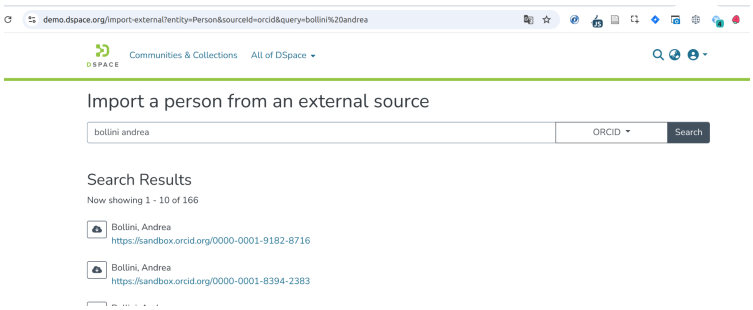
Import publications from ORCID

It is possible to import a publication from ORCID using the "Import from external sources" button in the MyDSpace page. Once you select the Publication entity type you will be able to find ORCID as a Source and you can get the list of publications (ORCID works) that appear in an ORCID profile by searching for its ORCID iD.



Import a person from ORCID

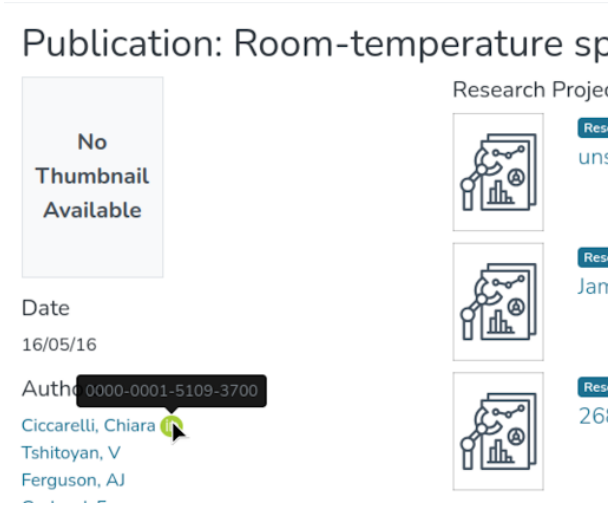
It is possible to import a person from ORCID using the "Import from external sources" button in the MyDSpace page. Once you select the Person entity type you will be able to find ORCID as a Source and you can get the list of person matching the provided query terms



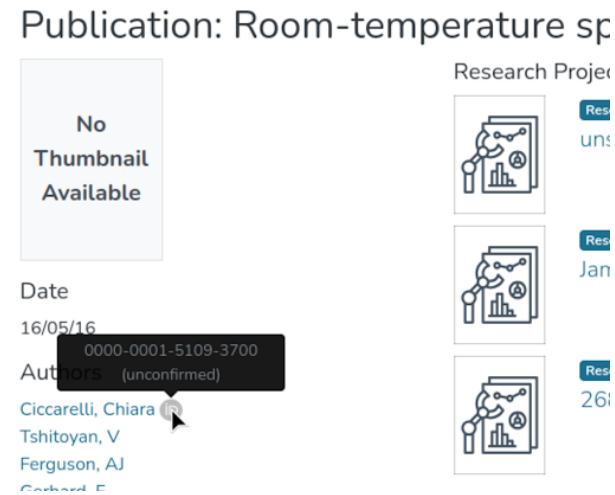
the person import from ORCID is more useful and common to be used in the context of the management of relationship among research outputs and activities (publications, projects, etc) and person, see the Configure the author lookup in submission

Displaying ORCID iDs on work landing pages

For ORCID iDs that have been associated with a user profile (Person Entity), either by a user connecting their authenticated ORCID iD with their profile, or an administrator adding an unauthenticated ORCID iD to a Person Entity profile: the ORCID iD icon will appear next to the user's name on any works landing pages where the user is an author, with the user's ORCID iD displayed upon hovering over the ORCID iD icon. Authenticated ORCID iDs will appear as a green ORCID iD icon:



Unauthenticated ORCID iDs will appear as a grey icon with the word "unconfirmed" in parentheses:



Configuration

Enable the integration

All the ORCID features requires a minimal common set of properties to configure in the `local.cfg`

```
# These URLs are for testing against ORCID's Sandbox API
# These are only useful for testing, and you must first request a Sandbox API Key from ORCID
orcid.domain-url= https://sandbox.orcid.org
orcid.api-url = https://api.sandbox.orcid.org/v3.0
orcid.public-url = https://pub.sandbox.orcid.org/v3.0
# Keep in mind, these API keys MUST be for the Sandbox API if you use "sandbox.orcid.org" URLs above!
orcid.application-client-id = <YOUR-SANDBOX-ORCID-CLIENT-ID>
orcid.application-client-secret = <YOUR-SANDBOX-ORCID-CLIENT-SECRET>

# Once you are ready to switch to Production, you need to update these settings to use ORCID's production API
# See https://github.com/ORCID/ORCID-Source/tree/master/orcid-api-web#endpoints
# orcid.domain-url= https://orcid.org
# orcid.api-url = https://api.orcid.org/v3.0
# orcid.public-url = https://pub.orcid.org/v3.0
# DON'T FORGET TO UPDATE YOUR API KEY! It must be a valid Public or Member API Key
# orcid.application-client-id = <YOUR-PRODUCTION-ORCID-CLIENT-ID>
# orcid.application-client-secret = <YOUR-PRODUCTION-ORCID-CLIENT-SECRET>
```

- **Enable in Production:** To enable the main integration (i.e. connect a local profile with ORCID and push data to the ORCID registry) you MUST to be an ORCID Member, get a Member API Key and properly enable and configure the feature in DSpace. See also "How do I register for Member API credentials?" from ORCID.
- **Enable in Testing:** To test ORCID integration, it's possible to use the ORCID Sandbox (without being an ORCID member). However, to do so, you must request a Sandbox Member API Key. See also "How do I register a public api client?".
- **Setting the "redirect URL" in ORCID:** In the ORCID API Credentials request form you will be asked to enter one or more redirect URLs for your application (DSpace). You will need to enter here the root URLs of your REST and user interfaces, which could be different. If the root URLs of both are the same, then just enter the URL of your user interface.
    - For example, for the DSpace 7 official demo, we use these redirect URLs:
        - User Interface: https://demo7.dspace.org (please note the absence of a /home or any subpaths)
        - REST API: https://api7.dspace.org (please note the absence of a /server or any subpaths)
    - For more information on valid ORCID redirect URLs, see "How do redirect URIs work?" from ORCID.
- **Configure the Client ID and Client Secret in DSpace:** Once ORCID has reviewed and approved your request, you will get from them the Client ID and Client Secret that need to be set in the `local.cfg` among other properties See the configuration examples above.

Please note that by default DSpace will request permissions to READ and WRITE all the information from the ORCID profile, as this will enable support for all of the features. You can fine-tune that by overriding the following properties. Please note that if you are going to configure Public API Credentials you MUST update this configuration keeping only the `/authenticate` scope as all the other scopes require Member API.

```
# The scopes to be granted by the user during the login on ORCID (see https://info.orcid.org/faq/what-is-an-
oauth-scope-and-which-scopes-does-orcid-support/)
orcid.scope = /authenticate
# The below scopes are ONLY valid if you have a Member API Key. They should be commented out if you only have a
Public API Key
orcid.scope = /read-limited
orcid.scope = /activities/update
orcid.scope = /person/update
```

To enable **ORCID Authentication** you need to uncomment the following line in the `modules/authentication.cfg` file or add it to your `local.cfg`

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

Please note that you are NOT required to enable the ORCID Authentication to use the other ORCID features, including the synchronisation ones. It is also possible to use just the ORCID Authentication without enabling all the other features.

When a user loggs in via ORCID the system will attempt to reuse an existing account looking up by email. If none is found then a new account is created in DSpace. It is possible to disable the creation of new accounts by setting the following property:

```
authentication-orcid.can-self-register = false
```

To enable **ORCID Synchronization**, you need to uncomment the following or add it to your `local.cfg`

```
# the properties below are required only for the sync / linking part (not for authentication or import)
orcid.synchronization-enabled = true
# you need to enable the orcidqueue consumer to keep track of what need to be sync between DSpace and ORCID
event.dispatcher.default.consumers = versioning, discovery, eperson, orcidqueue
```

The push of DSpace data (Person, Publication, Project) to ORCID is based on mappings defined in the `config/modules/orcid.cfg` file. You will find details below in the dedicated paragraphs.

The ORCID Synchronization features depend on other features that must be enabled: DSpace User Profile, Configurable Entities at least Person, Publication, Project & OrgUnit.

The synchronization features are classified as *experimental* at the time of 7.3 and it **MUST** be enabled manually. Due to the strict validation rules applied on the ORCID side and the absence of friendly edit UI for the archived items in DSpace (see issues#2876), it is hard at this time to achieve an optimal UX.

A few special configurations exist for **ORCID Disconnection** settings:

```
# Configuration with which it is established which user can disconnect a profile from orcid (none, only the
admin, only the owner or both).
# Allowed values are disabled, only_admin, only_owner or admin_and_owner
orcid.disconnection.allowed-users = admin_and_owner

# Configuration if the orcid sync settings should be remain on the profile when it is disconnected from orcid
or not
# (true = retain sync settings, false = delete old sync settings)
orcid.disconnection.remain-sync = true
```

Configure the push of information from DSpace to ORCID

Please note that many fields on the ORCID side are subject to validation, i.e. only values from controlled-list can be used and some fields are mandatory. The table below summarizes the validations that are defined at the time of DSpace 7.3. ORCID updates such rules periodically, usually modifying (enlarging) the controlled-list, and changes to the mandatory fields can also happen.

ORCID Entity	Corresponding DSpace Entity	Mandatory fields	Controlled fields
Person	Person		Country (iso-3166 2 code letter)
Work	Publication	Title Type Publication Date (>= 1900) External Identifier (at least one)	Type (https://info.orcid.org/documentation/integration-and-api-faq/#easy-faq-2682) Identifier.Type (https://pub.orcid.org/v3.0/identifiers)
Funding	Project	Title External Identifier (at least one) Funding Agency (Organisation) Currency if an Amount is provided	Amount.Currency (https://www.iso.org/iso-4217-currency-codes.html)
Organisation	OrgUnit	External Identifier (at least one) Address City Country	See https://support.orcid.org/hc/en-us/articles/360006894674-Metadata-in-the-Funding-section Country (iso-3166 2 code letter) Identifier.Type (ROR, LEI, CrossRef Funder ID, RINGOLD, GRID)

To provide more meaningful messages to the user DSpace implements a local validation before trying to push the record to ORCID. This validation verifies the data using the rules above so that a specific message is displayed to the user. If for any reason another error is returned by ORCID a generic message is shown to the user and the exact technical message received by ORCID is logged in the `dspace.log` file and stored in the `orcidhistory` table.

The local validation can be turned off. This validation is mainly intended as a development/debug option and should be not enabled in production.

```
orcid.validation.work.enabled = true
orcid.validation.funding.enabled = true
```

Mapping of the DSpace Person Items to ORCID Works

Metadata in a DSpace "Person" item, once that the item has been linked to ORCID by the researcher owning the ORCID profile, can be pushed to ORCID to fill the Profile Section of the ORCID Person using the `org.dspace.orcid.service.OrcidProfileSectionFactoryService` configured via this mapping bean:

```
    <!-- Configuration of ORCID profile sections factory.
    Each entry of sectionFactories must be an implementation of OrcidProfileSectionFactory.-->
    <bean class="org.dspace.orcid.service.impl.OrcidProfileSectionFactoryServiceImpl">
            <constructor-arg name="sectionFactories">
                    <list>

                            <bean class="org.dspace.orcid.model.factory.impl.OrcidSimpleValueObjectFactory">
                                    <constructor-arg name="sectionType" value="OTHER_NAMES" />
                                    <constructor-arg name="preference" value="BIOGRAPHICAL" />
                                    <property name="metadataFields" value="${orcid.mapping.other-names}" />
                                </bean>

                            <bean class="org.dspace.orcid.model.factory.impl.OrcidSimpleValueObjectFactory">
                                    <constructor-arg name="sectionType" value="KEYWORDS" />
                                    <constructor-arg name="preference" value="BIOGRAPHICAL" />
                                    <property name="metadataFields" value="${orcid.mapping.keywords}" />
                                </bean>

                            <bean class="org.dspace.orcid.model.factory.impl.OrcidSimpleValueObjectFactory">
                                    <constructor-arg name="sectionType" value="COUNTRY" />
                                    <constructor-arg name="preference" value="BIOGRAPHICAL" />
                                    <property name="metadataFields" value="${orcid.mapping.country}" />
                                </bean>

                            <bean class="org.dspace.orcid.model.factory.impl.
OrcidPersonExternalIdentifierFactory">
                                    <constructor-arg name="sectionType" value="EXTERNAL_IDS" />
                                    <constructor-arg name="preference" value="IDENTIFIERS" />
                                    <property name="externalIds" value="${orcid.mapping.person-external-
ids}" />
                                </bean>

                            <bean class="org.dspace.orcid.model.factory.impl.OrcidSimpleValueObjectFactory">
                                    <constructor-arg name="sectionType" value="RESEARCHER_URLS" />
                                    <constructor-arg name="preference" value="IDENTIFIERS" />
                                    <property name="metadataFields" value="${orcid.mapping.researcher-
urls}" />
                                </bean>

                    </list>
            </constructor-arg>
    </bean>
```

The above configuration links each piece of information that can be synchronized from DSpace to ORCID with a preference that the user can manage on the DSpace side (i.e. sync of the keywords is linked to the BIOGRAPHICAL preference) and defines which DSpace metadata will be used to fill the ORCID field. The bean reads the metadata mapping from the `config/modules/orcid.cfg`

```
### Other names mapping ###
orcid.mapping.other-names = person.name.variant
orcid.mapping.other-names = person.name.translated

### Keywords mapping ###
orcid.mapping.keywords = dc.subject

### Country mapping ###
orcid.mapping.country = person.country
orcid.mapping.country.converter =

### Person External ids mapping ###
##orcid.mapping.person-external-ids syntax is <metadatafield>::<type>
orcid.mapping.person-external-ids = person.identifier.scopus-author-id::SCOPUS
orcid.mapping.person-external-ids = person.identifier.rid::RID

### Researcher urls mapping ###
orcid.mapping.researcher-urls = dc.identifier.uri
```

Mapping of DSpace Publication items to ORCID Works

The mapping of DSpace default type has been established based on the ORCID works type specifications (as consulted on 2025-02-05): https://github.com/ORCID/orcid-model/blob/master/src/main/java/org/orcid/jaxb/model/common/WorkType.java

In DSpace, ORCID document type mapping is defined in the following properties file: mapConverter-dspace-to-orcid-publication-type.properties

Other document type uses

If you are using document types other than those provided by DSpace (i.e : document type not defined or document type in other language), it will be mapped to *other* ORCID document type. Changing it will require to edit DSpace mapConverter-dspace-to-orcid-publication-type.properties and ensure to conform with ORCID works type definitions.

When editing mapConverter-dspace-to-orcid-publication-type.properties spaces in the doctype label must be escaped. For example Book chapter must be noted Book\ chapter.

A DSpace "Publication" item is pushed to ORCID as a "Work" using the `org.dspace.orcid.model.factory.impl.OrcidWorkFactory` configured via this mapping bean:

```
        <bean id="orcidWorkFactoryFieldMapping" class="org.dspace.app.orcid.model.OrcidWorkFieldMapping" >
                <property name="contributorFields" value="${orcid.mapping.work.contributors}" />
                <property name="externalIdentifierFields" value="${orcid.mapping.work.external-ids}" />
                <property name="publicationDateField" value="${orcid.mapping.work.publication-date}" />
                <property name="titleField" value="${orcid.mapping.work.title}" />
                <property name="journalTitleField" value="${orcid.mapping.work.journal-title}" />
                <property name="shortDescriptionField" value="${orcid.mapping.work.short-description}" />
                <property name="subTitleField" value="${orcid.mapping.work.sub-title}" />
                <property name="languageField" value="${orcid.mapping.work.language}" />
                <property name="languageConverter" ref="${orcid.mapping.work.language.converter}" />
                <property name="typeField" value="${orcid.mapping.work.type}" />
                <property name="typeConverter" ref="${orcid.mapping.work.type.converter}" />
                <property name="citationType" value="${orcid.mapping.work.citation.type}" />
        </bean>
```

that reads the mapping from the `config/modules/orcid.cfg` file:

```
### Work (Publication) mapping ###
orcid.mapping.work.title = dc.title
orcid.mapping.work.sub-title =
orcid.mapping.work.short-description = dc.description.abstract
orcid.mapping.work.publication-date = dc.date.issued
orcid.mapping.work.language = dc.language.iso
orcid.mapping.work.language.converter = mapConverterDSpaceToOrcidLanguageCode
orcid.mapping.work.journal-title = dc.relation.ispartof
orcid.mapping.work.type = dc.type
orcid.mapping.work.type.converter = mapConverterDSpaceToOrcidPublicationType

##orcid.mapping.work.contributors syntax is <metadatafield>::<role>
orcid.mapping.work.contributors = dc.contributor.author::author
orcid.mapping.work.contributors = dc.contributor.editor::editor

##orcid.mapping.work.external-ids syntax is <metadatafield>::<type> or $simple-handle::<type>
##The full list of available external identifiers is available here https://pub.orcid.org/v3.0/identifiers
orcid.mapping.work.external-ids = dc.identifier.doi::doi
orcid.mapping.work.external-ids = dc.identifier.scopus::eid
orcid.mapping.work.external-ids = dc.identifier.pmid::pmid
orcid.mapping.work.external-ids = $simple-handle::handle
orcid.mapping.work.external-ids = dc.identifier.isi::wosuid
orcid.mapping.work.external-ids = dc.identifier.issn::issn
```

In the above configuration the "simple" properties are mapped matching the ORCID field name on the left (i.e. short-description) with the DSpace metadata that holds such information (i.e dc.description.abstract). For the ORCID Type field a special "converter" is configured so that the value of the DSpace metadata (i.e. dc.type) is mapped to the controlled-list of types accepted by ORCID (https://info.orcid.org/faq/what-work-types-does-orcid-support/). The value of the `orcid.mapping.work.type.converter` matches the name of a bean defined in the `config/spring/api/orcid-services.xml`

```
    <bean name="mapConverterDSpaceToOrcidPublicationType" class="org.dspace.util.SimpleMapConverter" init-
method="init">
                <property name="converterNameFile" value="orcid/mapConverter-dspace-to-orcid-publication-type.
properties" />
                <property name="configurationService" ref="org.dspace.services.ConfigurationService" />
                <property name="defaultValue" value="other"/>
        </bean>
```

Finally a special treatment is needed for the external ids as this is a complex field on the ORCID side composed of two values: the identifier type (from a controlled-list) and the identifier value. In this case the configuration maps a DSpace metadata field (i.e. dc.identifier.doi) to a specific identifier type (i.e. the part after :: , doi).

Mapping of DSpace Project items to ORCID Funding

A DSpace "Project" item is pushed to ORCID as a "Funding" using the `org.dspace.orcid.model.factory.impl.OrcidFundingFactory` configured via this mapping bean:

```
        <bean id="orcidFundingFactoryFieldMapping" class="org.dspace.orcid.model.OrcidFundingFieldMapping" >
                <property name="contributorFields" value="${orcid.mapping.funding.contributors}" />
                <property name="externalIdentifierFields" value="${orcid.mapping.funding.external-ids}" />
                <property name="titleField" value="${orcid.mapping.funding.title}" />
                <property name="typeField" value="${orcid.mapping.funding.type}" />
                <property name="typeConverter" ref="${orcid.mapping.funding.type.converter}" />
                <property name="amountField" value="${orcid.mapping.funding.amount}" />
                <property name="amountCurrencyField" value="${orcid.mapping.funding.amount.currency}" />
                <property name="amountCurrencyConverter" ref="${orcid.mapping.funding.amount.currency.
converter}" />
                <property name="descriptionField" value="${orcid.mapping.funding.description}" />
                <property name="startDateField" value="${orcid.mapping.funding.start-date}" />
                <property name="endDateField" value="${orcid.mapping.funding.end-date}" />
                <property name="organizationRelationshipType" value="${orcid.mapping.funding.organization-
relationship-type}" />
        </bean>
```

that reads the mapping from the `config/modules/orcid.cfg` file

```
### Funding mapping ###
orcid.mapping.funding.title = dc.title
orcid.mapping.funding.type =
orcid.mapping.funding.type.converter = mapConverterDSpaceToOrcidFundingType
##orcid.mapping.funding.external-ids syntax is <metadatafield>::<type>
##The full list of available external identifiers is available here https://pub.orcid.org/v3.0/identifiers
orcid.mapping.funding.external-ids = dc.identifier::grant_number
orcid.mapping.funding.external-ids = dc.identifier.other::other-id
orcid.mapping.funding.description = dc.description
orcid.mapping.funding.start-date = project.startDate
orcid.mapping.funding.end-date = project.endDate
##orcid.mapping.funding.contributors syntax is <metadatafield>::<type>
orcid.mapping.funding.contributors = project.investigator::lead
orcid.mapping.funding.organization-relationship-type = isOrgUnitOfProject
orcid.mapping.funding.amount = project.amount
orcid.mapping.funding.amount.currency = project.amount.currency
orcid.mapping.funding.amount.currency.converter = mapConverterDSpaceToOrcidAmountCurrency
```

in the above configuration the "simple" properties are mapped matching the ORCID field name on the left (i.e. description) with the DSpace metadatafield that holds such information (i.e dc.description). For the ORCID Type field a special "converter" is configured so that the value of the DSpace metadata (i.e. dc.type) is mapped to the controlled-list of types accepted by ORCID (https://support.orcid.org/hc/en-us/articles/360006894674-Metadata-in-the-Funding-section). The value of the `orcid.mapping.funding.type.converter` matches the name of a bean defined in the `config/spring/api/orcid-services.xml` The same apply for the currency `orcid.mapping.funding.amount.currency.converter = mapConverterDSpaceToOrcidAmountCurrency`

```
<bean name="mapConverterDSpaceToOrcidFundingType" class="org.dspace.util.SimpleMapConverter" init-method="init">
        <property name="converterNameFile" value="orcid/mapConverter-dspace-to-orcid-funding-type.properties" />
        <property name="configurationService" ref="org.dspace.services.ConfigurationService" />
        <property name="defaultValue" value=""/>
</bean>

<bean name="mapConverterDSpaceToOrcidAmountCurrency" class="org.dspace.util.SimpleMapConverter" init-method="
init">
        <property name="converterNameFile" value="orcid/mapConverter-dspace-to-orcid-amount-currency.
properties" />
        <property name="configurationService" ref="org.dspace.services.ConfigurationService" />
        <property name="defaultValue" value=""/>
</bean>
```

Finally a special treatment is needed for Funder that is a mandatory field on the ORCID side. In this case the mapping defines which relation is used to link the Project with the Funder (OrgUnit)

```
orcid.mapping.funding.organization-relationship-type = isOrgUnitOfProject
```

Configure the import features

The Import features from ORCID have been implemented using the Live Import Framework

The following bean is used to configure the **import of person records from ORCID**. It is activated as an external source in `config/spring/api/external-services.xml`

```
    <bean class="org.dspace.external.provider.impl.OrcidV3AuthorDataProvider" init-method="init">
        <property name="sourceIdentifier" value="orcid"/>
        <property name="orcidUrl" value="${orcid.domain-url}" />
        <property name="clientId" value="${orcid.application-client-id}" />
        <property name="clientSecret" value="${orcid.application-client-secret}" />
        <property name="OAUTHUrl" value="${orcid.token-url}" />
        <property name="orcidRestConnector" ref="orcidRestConnector"/>
        <property name="supportedEntityTypes">
            <list>
                <value>Person</value>
            </list>
        </property>
    </bean>
```

the mapping between ORCID Person and the DSpace Person Item is the following, currently hard-coded:

ORCID	DSpace
Name/FamilyName	person.firstName
Name/GivenName	person.givenName
Name/Path	person.identifier.orcid
ORCID Profile URL	dc.identifier.uri

The following bean is instead used to configure the **import of publication records** from ORCID (Work)

```
    <bean id="orcidPublicationDataProvider" class="org.dspace.external.provider.impl.
OrcidPublicationDataProvider">
        <property name="sourceIdentifier" value="orcidWorks"/>
        <property name="fieldMapping" ref="orcidPublicationDataProviderFieldMapping"/>
        <property name="supportedEntityTypes">
            <list>
                <value>Publication</value>
            </list>
        </property>
    </bean>
```

The mapping of the ORCID Work metadata to the DSpace metadata is performed by the following bean in `config/spring/api/orcid-services.xml`

```
        <bean id="orcidPublicationDataProviderFieldMapping" class="org.dspace.orcid.model.
OrcidWorkFieldMapping" >
                <property name="contributorFields" value="${orcid.external-data.mapping.publication.
contributors}" />
                <property name="externalIdentifierFields" value="${orcid.external-data.mapping.publication.
external-ids}" />
                <property name="publicationDateField" value="${orcid.external-data.mapping.publication.issued-
date}" />
                <property name="titleField" value="${orcid.external-data.mapping.publication.title}" />
                <property name="journalTitleField" value="${orcid.external-data.mapping.publication.is-part-
of}" />
                <property name="shortDescriptionField" value="${orcid.external-data.mapping.publication.
description}" />
                <property name="languageField" value="${orcid.external-data.mapping.publication.language}" />
                <property name="languageConverter" ref="${orcid.external-data.mapping.publication.language.
converter}" />
                <property name="typeField" value="${orcid.external-data.mapping.publication.type}" />
                <property name="typeConverter" ref="${orcid.external-data.mapping.publication.type.converter}"
/>
        </bean>
```

that reads the mapping from the `config/modules/orcid.cfg` file.

Please note that such mapping is separated from the mapping used to push information from DSpace to ORCID but usually, as provided in the default configuration, the mapping should be the same.

```
### Work (Publication) external-data.mapping ###
orcid.external-data.mapping.publication.title = dc.title

orcid.external-data.mapping.publication.description = dc.description.abstract
orcid.external-data.mapping.publication.issued-date = dc.date.issued
orcid.external-data.mapping.publication.language = dc.language.iso
orcid.external-data.mapping.publication.language.converter = mapConverterOrcidToDSpaceLanguageCode
orcid.external-data.mapping.publication.is-part-of = dc.relation.ispartof
orcid.external-data.mapping.publication.type = dc.type
orcid.external-data.mapping.publication.type.converter = mapConverterOrcidToDSpacePublicationType

##orcid.external-data.mapping.publication.contributors syntax is <metadatafield>::<role>
orcid.external-data.mapping.publication.contributors = dc.contributor.author::author
orcid.external-data.mapping.publication.contributors = dc.contributor.editor::editor

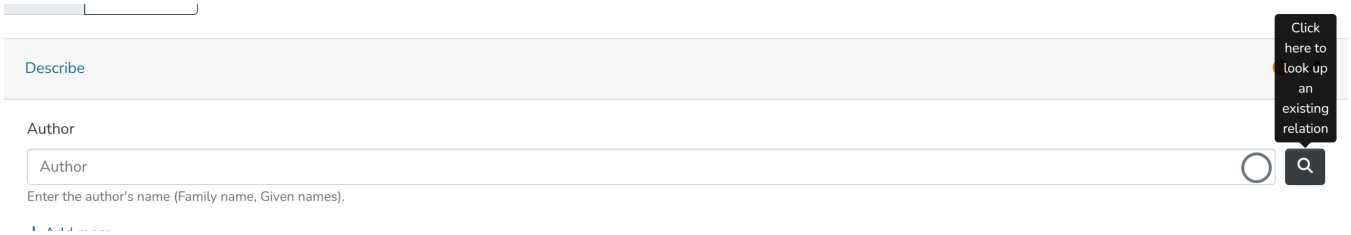
##orcid.external-data.mapping.publication.external-ids syntax is <metadatafield>::<type> or $simple-handle::
<type>
##The full list of available external identifiers is available here https://pub.orcid.org/v3.0/identifiers
orcid.external-data.mapping.publication.external-ids = dc.identifier.doi::doi
orcid.external-data.mapping.publication.external-ids = dc.identifier.scopus::eid
orcid.external-data.mapping.publication.external-ids = dc.identifier.pmid::pmid
orcid.external-data.mapping.publication.external-ids = dc.identifier.isi::wosuid
orcid.external-data.mapping.publication.external-ids = dc.identifier.issn::issn
```

Configure the author lookup in submission

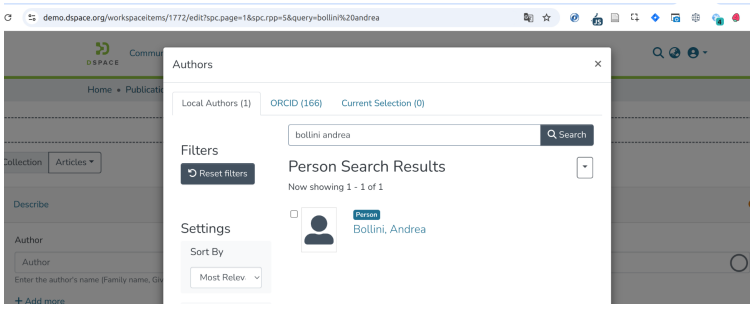
Please note that there are two different possibilities:

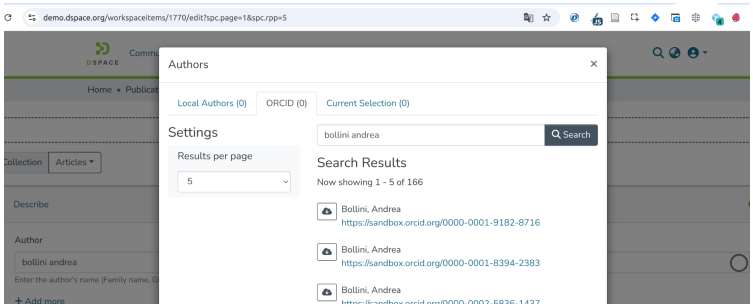
- via an ORCID lookup authority available for a DSpace repository that is not using Configurable Entities
- via a relation among the research output item (Publication, etc.) and a Person Item bound to the ORCID Person External Source defined in the previous paragraph. The screen below belongs to this second scenario (linking via a relations)

During the submission of a publications (or other entities that have been configured to be in relation with the Person entity), the metadata that is used as fallback (see Configure a Relation-Field) has a search icon



clicking on it, a dialog will be open allowing to search among the existing person item (Local Authors tab) or in external sources





Troubleshooting & common issues

The troubleshooting guide from ORCID can help as well https://info.orcid.org/documentation/integration-guide/troubleshooting/

I'm having trouble testing the ORCID integration. What should I check?

Please double check the documentation and the other FAQs to be sure that you have followed all of the instructions to enable the integration correctly. If you still have trouble, contact the DSpace tech community via email or slack providing as much detail as possible. If the issue is related to the synchronization of DSpace local data with ORCID it would be useful to share information about the content of your `orcidhistory` table and any relevant message that you could have in the `dspace.log` file.

I cannot find the ORCID features described by this page in my installation

The ORCID features must be enabled by changing some configuration files. Please refer to the ***Enable the integration section*** above.

I'm unable to authenticate via ORCID

If you have correctly enabled the ORCID authentication feature and you are able to start the OAuth flow with ORCID but get a failure when you are redirected back to DSpace, it could be due to privacy settings on your ORCID record. The DSpace ORCID authentication **requires that you release an email address to match your ORCID account with a DSpace account** or to create a new one at your first login. Make your ORCID account email address public or visible to trusted parties. This is often not the case for a freshly created account on ORCID.

If you are encountering this issue, you'll see a message like this in your "dspace.log" file on the backend:

```
2022-08-04 11:43:42,124 ERROR unknown unknown org.dspace.authenticate.OrcidAuthenticationBean @ An error occurs
registering a new EPerson from ORCID
java.lang.IllegalStateException: The email is configured private on orcid
```

After logging in via ORCID, a new DSpace account was created instead of using my existing DSpace account

Currently, the ORCID integration with DSpace relies on a matching email address to find your existing account. If your ORCID account and DSpace account have **different** email addresses associated with them, then it is possible that a new (duplicative) user account will be created.

I'm having trouble creating test accounts on ORCID to experiment with the features

Please refer to the ORCID trouble-shooting guide https://info.orcid.org/documentation/integration-guide/troubleshooting/ A frequent mistake working with the ORCID sandbox environment is to forget that only email addresses @mailinator.com are allowed for account created on the sandbox. Remember to validate your email address once the account as been created visiting the online inbox at mailinator.com

I have configured my Public ORCID API credentials in DSpace but I get an error attempting to login via ORCID

When you use public ORCID API credentials you can only use a subset of the integration features (check). Moreover you need to limit the scopes (permissions) requested to the user via the ORCID authentication to the `/authenticate` scope. Please check the ***Enable the integration section*** above.

I don't find my publications looking up for my ORCID iD

Please check "`config/modules/orcid.cfg`" (or your `local.cfg`) to see if the system has been properly configured to use the production ORCID API. There is a chance that your installation is still configured to use the ORCID Sandbox that is appropriate for the testing and development phase of the integration. The ORCID sandbox doesn't contain the same data as the Public environment.

I cannot push all my publications, only few or none of them are listed in the queue

Please double check that the `orcidqueue` consumer has been enabled (in dspace.cfg or local.cfg) and that the orcid settings of your profile have the "All publications" checkbox flagged. ORCID features require the use of the new Configurable Entities. Only Publication item are synchronized with ORCID; simple "untyped" Items will not be synchronized. Please consider to convert your legacy collection to "Publication" collection and set a `dspace.entity.type = Publication` metadata on your legacy items.

Push of publications to ORCID fails

This is usually due to validation errors. ORCID could complain about missing mandatory fields or invalid values for fields that are linked to a controlled-list. Please check the table in the ***Configure the push of information from DSpace to ORCID*** and the ***Mapping of DSpace Publication items to ORCID Works paragraph*** to solve this. Your dspace.log file may also provide useful error messages.

Push of projects to ORCID fails

This is usually due to validation errors. Make sure that all required metadata fields exist on the Project Entity and any linked OrgUnit Entities. ORCID could complain about missing mandatory fields or invalid values for fields that are linked to a controlled-list. Please check the table in the ***Configure the push of information from DSpace to ORCID*** and the ***Mapping of DSpace Project items to ORCID Funding paragraph*** to solve this. Your dspace.log file may also provide useful error messages.

ORCID Login: Guide to Repository access and Account linking

DRAFT

The ORCID Login functionality described on this page is available out of the box since DSpace 9.0. For organizations using earlier versions of DSpace, code is available for implementing full ORCID Login features for DSpace 7.x (https://github.com/DSpace/DSpace/pull/9846) and DSpace 8.x (https://github.com/DSpace/DSpace/pull/9847).

The ORCID login allows the user to access the repository using their ORCID login credentials through the ORCID system.

By clicking on **Log in via ORCID**, the user will be redirected externally to log in to ORCID website.

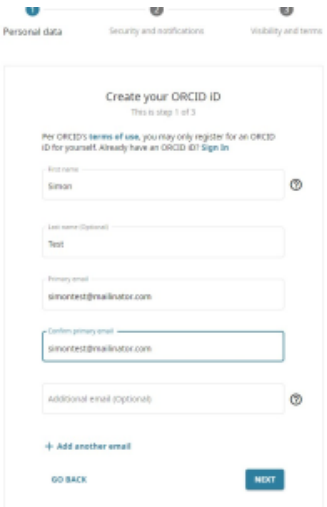
When logging in for the first time, different scenarios can occur.

- Never registered users
  - Unknown or same email as ORCID
  - Confirming a different email from ORCID
- User registered in DSpace with a different email from ORCID
- User logging in with same email for ORCID and DSpace
- ORCID Login flow

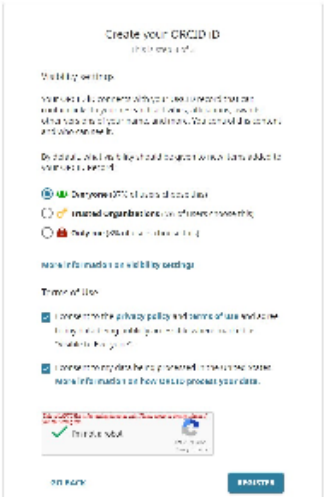
Never registered users

Unknown or same email as ORCID

When clicking on **Log in with ORCID**, the ORCID log in page will show, and the user can sign in with their ORCID credentials or register for an ORCID iD if they don't already have one.



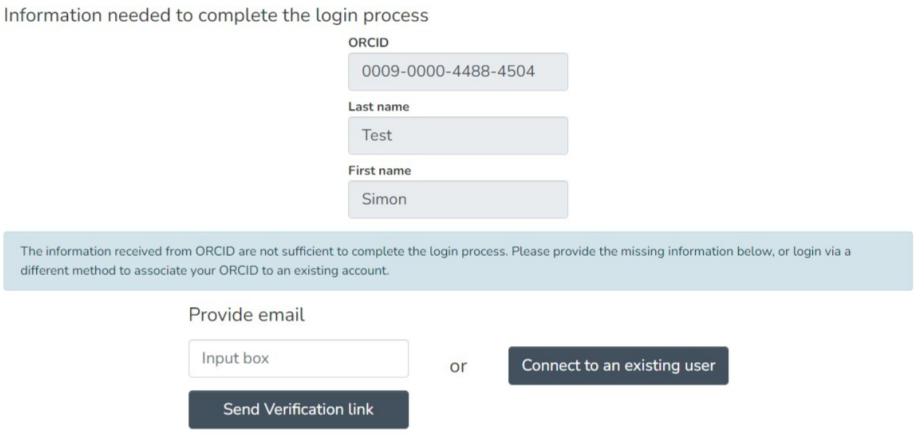
To make the integration work at its best, general ORCID record visibility needs to be set for **Everyone**. (Learn more about ORCID visibility settings.)



At the end of the procedure, access to DSpace for data should be granted. After saving, the user will be redirected to DSpace.

The page will list the data retrieved from ORCID and let the user provide a new email address to register to DSpace or connect the ORCID account to an existing user.

The email is usually hidden on this page if it is not set to public/everyone visibility on ORCID. It will be retrieved and shown as user data if it's public.



For **new registration**, an email should be provided to create a profile after sending the verification link and verifying the email address.

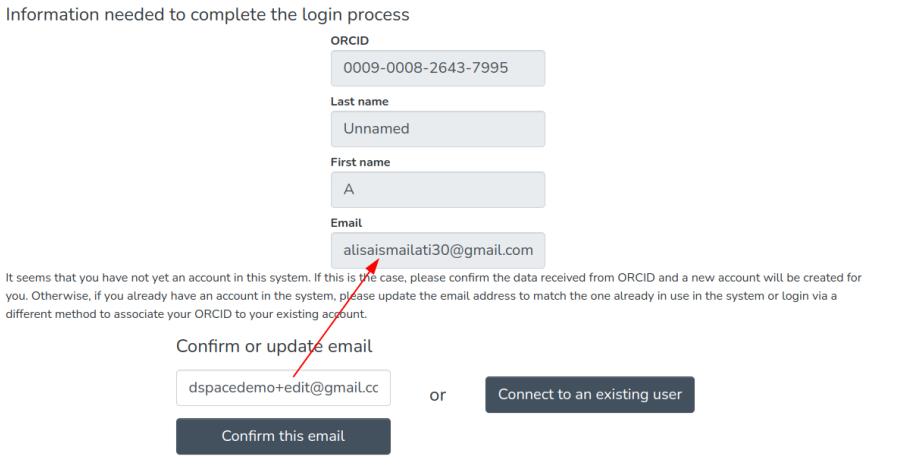
If the user's email address from ORCID is not retrieved or if the user confirms the same email as ORCID, the system will successfully create a new user in DSpace.

Confirming a different email from ORCID

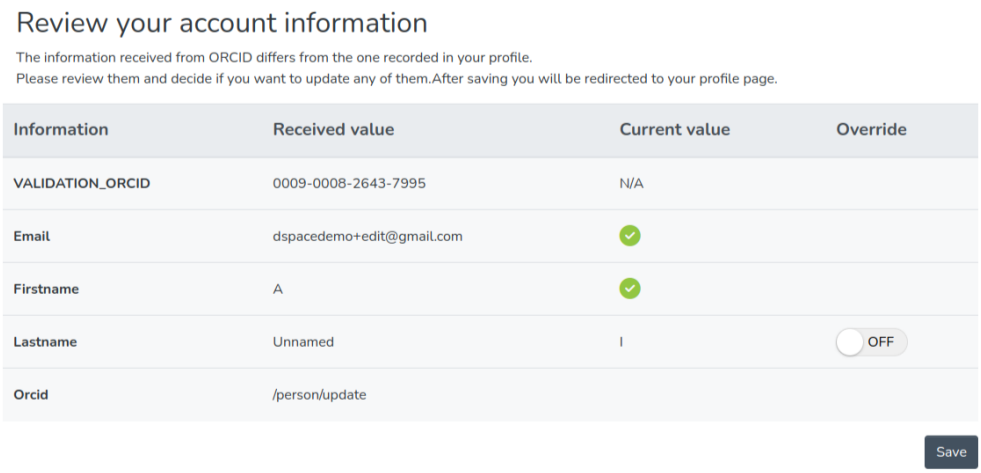
If the user's email address is set as public in ORCID, when logging in to DSpace, the system will also receive and show the email as user data.

In this case, the user can provide DSpace with the same email address and complete the registration.

If the **email address** provided to DSpace **differs** from the one received from **ORCID**, the user, after the email verification, will **review the account information** and choose the data to keep in the system in case there are duplicate values.



With the toggle of the override column, the user can decide whether to keep the existing value by setting it to **OFF** or overwrite the information by setting the toggle to **ON**.



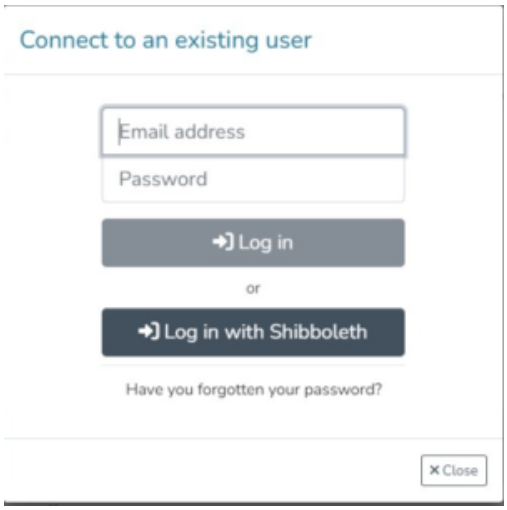
After clicking on **Save** and confirming, the account is successfully created, and the user will be redirected to the Profile page, where, if needed, they can update some of the information.

User registered in DSpace with a different email from ORCID

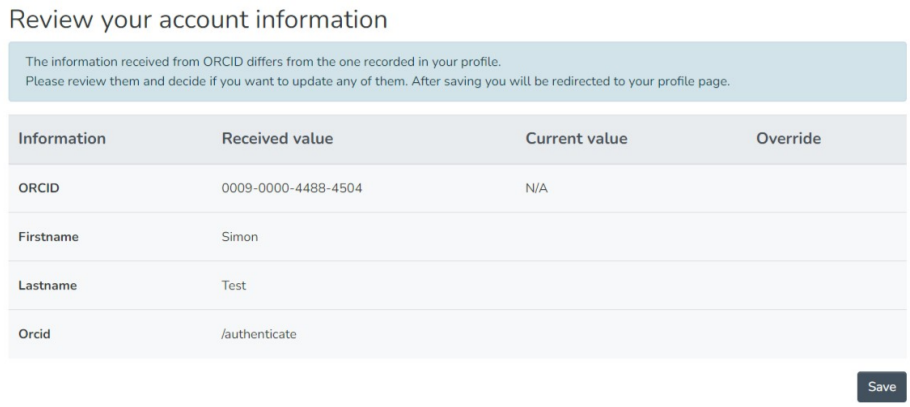
If the user has registered to DSpace and ORCID with different emails, the steps to be followed are similar to the previous.

After logging in with ORCID for the first time with an email never registered to DSpace, the system will retrieve information from ORCID and ask if to create a new account by providing a new email or connect to an existing user.

By clicking on **Connect to an existing user**, the login pop-up will show, in which input the DSpace credentials.



After logging in, the user can review the information that will be associated with the existing account.



By clicking on Save, and confirming the willingness to go on, the user will be redirected to the Profile page in which, if needed, update some of the information.

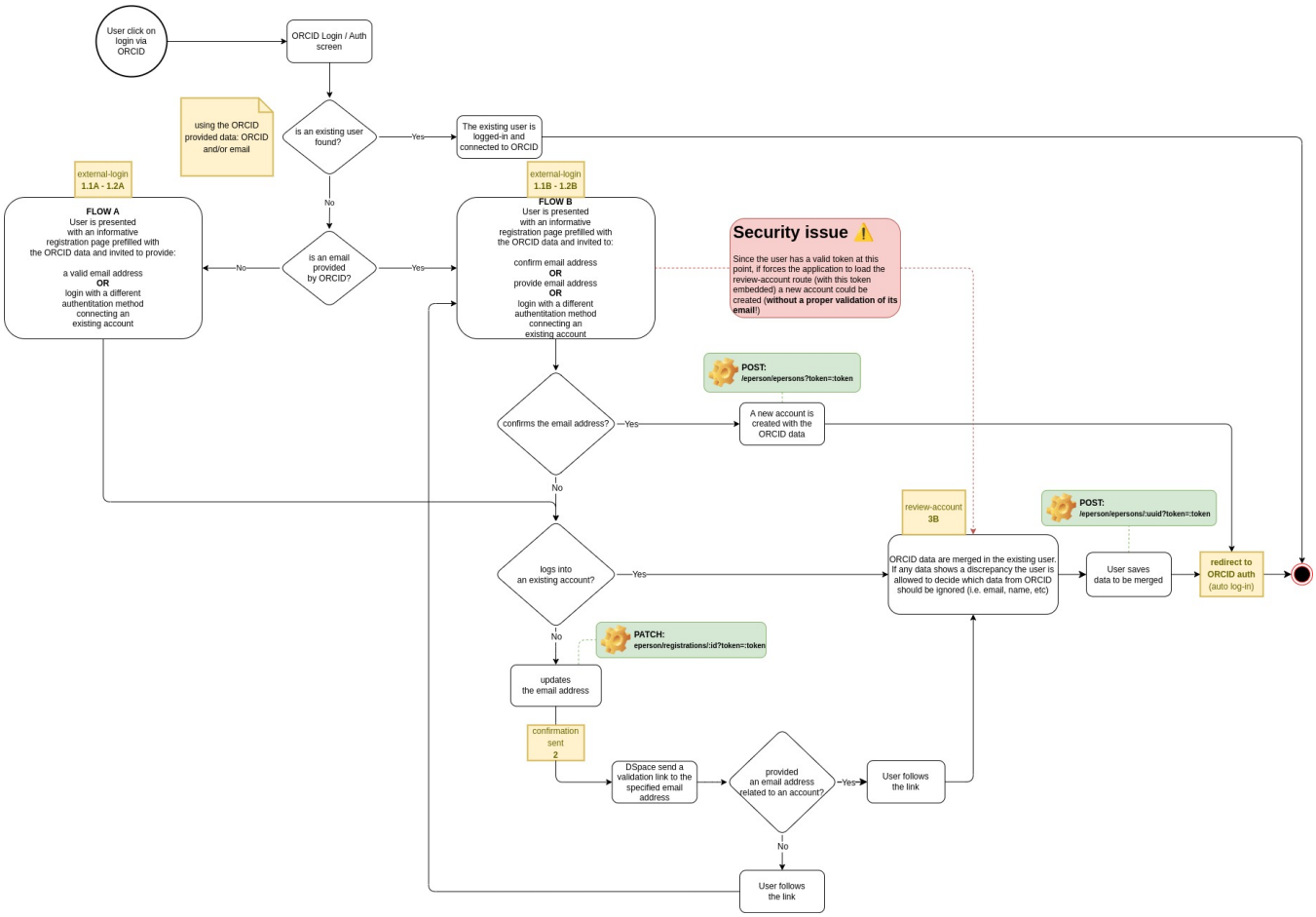
ORCID can be connected to an existing ePerson also from the personal page.

User logging in with same email for ORCID and DSpace

If the user has already registered on DSpace and log in with ORCID with the same email address, the ORCID and DSpace profile will be connected, associating the profile and the ORCID number.

ORCID Login flow

This is the overview of the flow for the ORCID Login.


User click on login via ORCID
ORCID Login / Auth screen
using the ORCID provided data: ORCID and/or email
is an existing user found?
Yes
The existing user is logged-in and connected to ORCID
No
external-login 1.1A - 1.2A
FLOW A
User is presented with an informative registration page prefilled with the ORCID data and invited to provide:
a valid email address
OR
login with a different authentitation method connecting an existing account
No
is an email provided by ORCID?
Yes
external-login 1.1B - 1.2B
FLOW B
User is presented with an informative registration page prefilled with the ORCID data and invited to:
confirm email address
OR
provide email address
OR
login with a different authentitation method connecting an existing account
Security issue
Since the user has a valid token at this point, if forces the application to load the review-account route (with this token embedded) a new account could be created (without a proper validation of its email!)
POST: /eperson/epersons?token=:token
confirms the email address?
Yes
A new account is created with the ORCID data
No
review-account 3B
POST: /eperson/epersons/:uuid?token=:token
logs into an existing account?
Yes
ORCID data are merged in the existing user. If any data shows a discrepancy the user is allowed to decide which data from ORCID should be ignored (i.e. email, name, etc)
User saves data to be merged
redirect to ORCID auth (auto log-in)
No
PATCH: eperson/registrations/:id?token=:token
updates the email address
confirmation sent 2
DSpace send a validation link to the specified email address
provided an email address related to an account?
Yes
User follows the link
No
User follows the link

Researcher Profiles

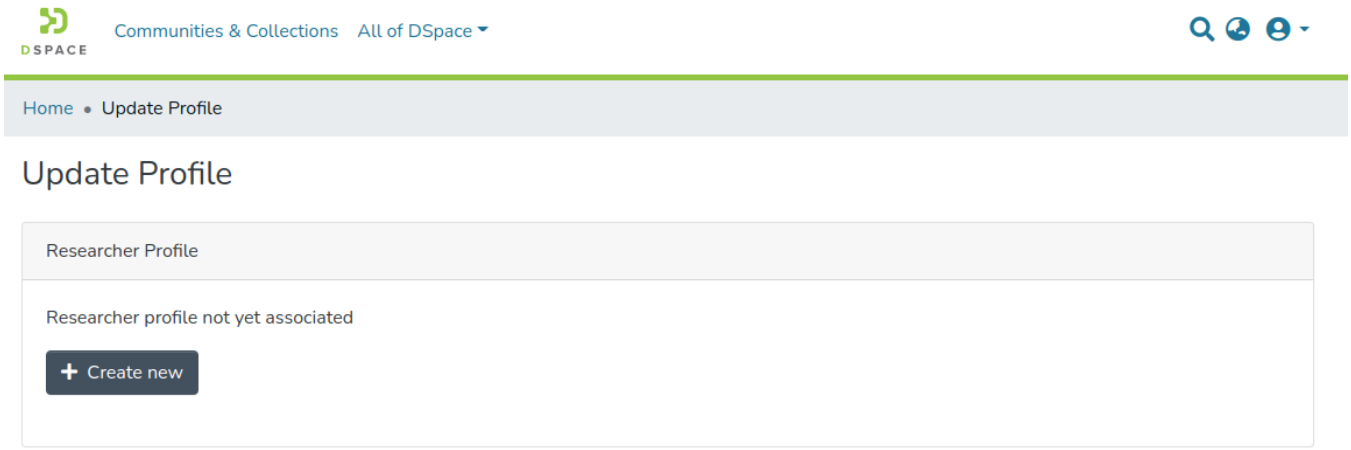
The DSpace Researcher Profile feature has been introduced in DSpace 7.3 to support the ORCID Integration work but can be used alone. It is turned off by default and must be enabled manually

A DSpace Researcher Profile is a special Person Entity (item) that is linked with exactly one EPerson (DSpace account). This linked EPerson owns the profile (Person Item), including having WRITE permission on it. The link between the Person Item and the EPerson is managed in the Person's `dspace.object.owner` metadata field. This field is configured to hold authority values and will contain the `UUID` of the EPerson that owns the profile.

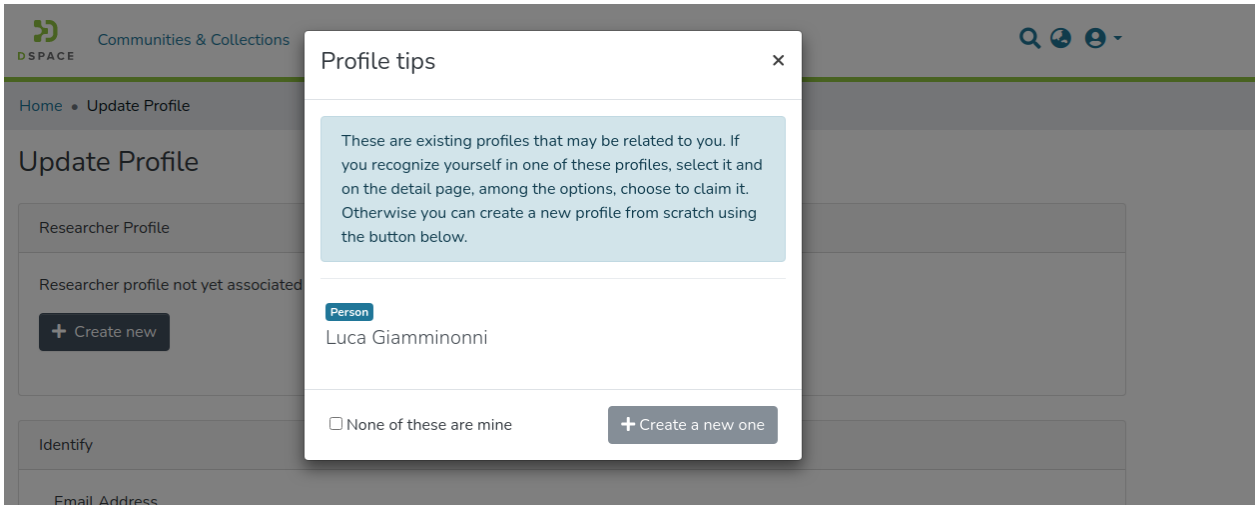
Here a summary of the key concepts & requirements of the feature:

- Profiles require Configurable Entities to be enabled, as every Researcher Profile is represented by a Person Entity.
- A profile can be linked to only one EPerson (user account). That EPerson has full rights to manage the profile, including whether the profile is publicly visible or private.
- Optionally, Profiles can be synchronized (or initially created) via ORCID Integration.

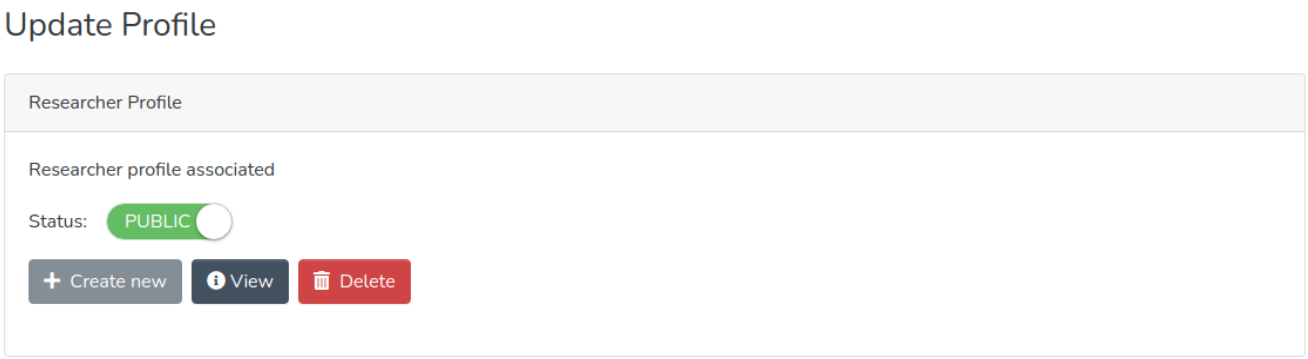
When the feature is enabled, the user can create a researcher profile from his Profile (account page)



If a Person Item already exists in the system, matching the account email address, this Person Item is offered to the user:



Once a profile has been created or claimed, the user can make it public (Anonymous READ) or private:



By default, deletion of the researcher profile does NOT delete the corresponding Person Item. Instead, it just unlinks the Person Item from the EPerson account. This behavior can be changed as specified in the ***Advanced configuration*** section below.

Enable the feature

To enable the feature you need to set the following property (uncommenting it in the `config/modules/researcher-profile.cfg` or adding it to the `config/local.cfg`)

```
researcher-profile.entity-type = Person
```

You can specify a different Entity Type for the item that can be used as profile. This is an advanced setting -- change it only if you know what are you doing and have implemented specific customisation.

You need to enable also the EPerson authority for the `dspace.object.owner`. Uncomment the following lines in the `config/modules/authority.cfg`:

```
# Configuration settings required for Researcher Profiles
# These settings ensure "dspace.object.owner" field are indexed by Authority Control
choices.plugin.dspace.object.owner = EPersonAuthority
choices.presentation.dspace.object.owner = suggest
authority.controlled.dspace.object.owner = true
```

Last, you need to ensure that at least one Collection is configured to accept Person entities. Only EPersons having the submission right in such a Collection will be able to create profiles. There are many possibilities for using these settings to control who may or may not create a profile.

Advanced configuration

You can configure some aspects of the Profile feature in the `config/modules/researcher-profile.cfg`

Property:	`researcher-profile.entity-type`
Example Value:	`Person`
Informational Note:	The type of Entity to use for Researcher Profile items. By default, the Person Entity is used, as this is provided out-of-the-box in DSpace. This would only need to be modified if you have created a heavily customized Configurable Entities data model which does NOT include Person.
Property:	`researcher-profile.collection.uuid`

Example Value:	[collection-uuid]
Informational Note:	UUID of the Collection where all Researcher Profiles should be created by default. This Collection MUST be configured to accept Person Entities (or the entity type specified in "researcher-profile.entity-type"). By default this is UNSPECIFIED. The default behavior is that the person's Researcher Profile will be created in the Collection in DSpace which is configured to accept Person Entities and where the user has permissions to submit. If multiple Collections of this type are available, then the first one found will be used.
Property:	`researcher-profile.hard-delete.enabled`
Example Value:	`false`
Informational note	Whether to enable "hard delete" when a Researcher Profile is deleted by an EPerson. When "hard delete" is enabled (set to true), then anytime an EPerson deletes their Researcher Profile, the underlying Person Entity will be deleted (i.e. this acts as a permanent deletion). When "hard delete" is disabled (set to false, the default value), then anytime an EPerson deletes their Researcher Profile, it will simply be "unlinked". In other words, the underlying Person Entity will be kept in the system.
Property:	`researcher-profile.set-new-profile-visible`
Example Value:	false
Informational note	Whether to make a new Researcher Profile "visible" (i.e. allow anonymous access) on creation. When set to "false" (default value), a newly created Researcher Profile will only be accessible to the EPerson who created it. That EPerson may chose to make to visible (i.e. allow anonymous access) at a later time. When set to "true", a newly created Researcher Profile will be immediately accessible to anonymous users. But, the EPerson who created it may chose to hide it (i.e. disallow anonymous access) at a later time.

Troubleshooting

I cannot find this feature

The feature needs to be enabled explicitly. Please follow the instruction in the ***Enable the feature*** section above.

The users sees an error when they try to create their profile

The feature requires that the Person entity be configured in the data model (see Configurable Entities) and the user must have permission to submit in at least one collection configured to accept Person entities. Please double check that the EPersonAuthority is bound to the `dspace.object.owner` metada ta -- see the ***Enable the feature*** section above.

Sending emails to fixed recipients

DSpace 10 provides configuration options to control or prevent email delivery. These settings are especially useful in development and test environments to avoid sending messages to real users.

Disabling the Mail Server

The mail server can be completely disabled using the following property:

```
mail.server.disabled = true
```

When this option is enabled:

- **No emails are sent by DSpace**
- DSpace **logs the subject of each email** that would have been sent
- No message is delivered to any recipient

By default, the property is set to:

```
mail.server.disabled = false
```

CatchAll Email Configuration

DSpace 10 also supports a *catchall* mechanism that redirects all outgoing emails to a predefined list of recipients.

To enable this feature, the following properties must be configured:

```
mail.server.catchAll.enabled = true
```

```
mail.server.catchAll.recipient = admin@example.org,test@example.org
```

When the catchall system is enabled:

- **All emails are sent only to the addresses specified** in `mail.server.catchAll.recipient`
- **Original recipients do not receive the email**
- The original recipients are **logged for reference**

Important Notes

The two features serve **different purposes**:

- `mail.server.disabled` completely suppresses email delivery
- `mail.server.catchAll.enabled` allows email delivery, but only to controlled recipients

Share Content via Social Network

Work in Progress

By means of the **AddToAny** plugin, it is possible to share DSpace-CRIS content with common Social Media Platforms.

- Configuration
  - Environment properties
    - vailable social networks - First method
    - Available social networks - Second method
  - Angular Router
- GDPR and CCPA compliance

Configuration

The AddToAny plugin can be enabled and configured through the Angular environment property `addToAnyPlugin`. It is also possible to decide on which pages it should be enabled by configuring the Angular router.

The *AddToAny* plugin is enbled by default. In order to disable it you need to set the Angular environment property `addToAnyPlugin.socialNetworksEnabled` to false.

```
addToAnyPlugin:
  socialNetworksEnabled: false
```

Environment properties

The Angular environment property `addToAnyPlugin` provides the following configuration properties:

- **scriptUrl** is the URL of the script that enables the plugin
- **socialNetworksEnabled** is a boolean flag that allow to enable the plugin
- **buttons** is an array of strings that contains the IDs of available buttons
- **showPlusButtons** is a boolean flag that enables the ➕ button that shows all available share options
- **showCounters** is a boolean flag that shows the share count for each service that provides a public API
- **title** is a string that may be used as title, content, or subject for the shared post
- **link** is an optional string containing the address of the DSpace installation,

This is the default plugin configuration in `default-app-config.ts`:

```
addToAnyPlugin: AddToAnyPluginConfig = {
  scriptUrl: 'https://static.addtoany.com/menu/page.js',

  socialNetworksEnabled: false,
  buttons: [],
  showPlusButton: true,
  showCounters: true,
  title: 'DSpace demo',
};
```

This is an example of configuration override in `config.ENVIRONMENT.yml` file:

```
addToAnyPlugin:
  scriptUrl: "https://static.addtoany.com/menu/page.js"

  socialNetworksEnabled: true
  buttons:
    - facebook
    - twitter
    - linkedin
    - email
    - copy_link
  showPlusButton: true
  showCounters: true
  title: DSpace demo
```

vailable social networks - First method

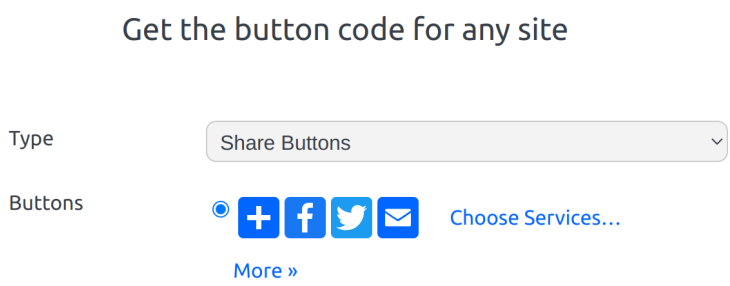
Search the desired service in the following list, then copy the service code and add it to the `buttons` property:

Services - AddToAny

Available social networks - Second method

Another way to get the button IDs is to generate the button code on AddToAny website.

Open Share Buttons Code for Any Website - AddToAny, then click “Choose services” and select the desired services:



Then click “Get button code” to generate the HTML code:

```
<!-- AddToAny BEGIN -->
<div class="a2a_kit a2a_kit_size_32 a2a_default_style">
<a class="a2a_dd" href="https://www.addtoany.com/share"></a>
<a class="a2a_button_facebook"></a>
<a class="a2a_button_twitter"></a>
<a class="a2a_button_email"></a>
</div>
<script async src="https://static.addtoany.com/menu/page.js"></script>

<!-- AddToAny END -->
```

For each button you will get an anchor element containing the button ID:

```
<a class="a2a_button_BUTTON-ID"></a>
```

In order to add the buttons you can copy all the IDs and add them to the `button` array.

Note that you don’t need this HTML code, but only need the button IDs. The HTML code is automatically generated by DSpace CRIS.

Angular Router

Configure the **Angular Router** to activate the feature in every desired route (you may want to include just informative pages in order to exclude administrative pages or edit pages).

Add `showSocialButtons: true` to the data object of each configured route

```
{
  path: ':id',
  data: {
    showSocialButtons: true

  }
}
```

The AddToAny container element will then be positioned inside the page accordingly to the widget configuration.

GDPR and CCPA compliance

Accorting to what reported at https://www.addtoany.com/buttons/faq/#data-privacy, AddToAny does not store personal data and does not require the user to accept any privacy policy.

Shared submission / workspace

Overview

In DSpace, Workspace Items may be visible and accessible to:

- **Administrators** of the collection where the item will be placed
- **The submitter** who created the workspace item
- **Other submitters** of the same collection, but **only** if the Collection has the "shared workspace" setting enabled
- **Other EPersons** according to a configurable set of metadata (co-authors), but **only** if the Collection has the "shared workspace" setting enabled

The submitter finds their Workspace items in the MyDSpace dashboard under the default "Your Submission" option. Other EPersons allowed to see Workspace items submitted by others will find those items under the "Shared submission" option.

Configuration

By default, Workspace items are accessible to administrators of the collection where they are going to be placed.

When a Collection has the "Shared workspace" setting enabled (from the Collection's "Edit Metadata" tab), this allows all Workspace Items within this collection to be **shared** submissions among all other submitters to that collection. For example, if one submitter starts a submission in that collection, all other submitters to that collection will have access to it from the "Shared submission" option on their MyDSpace page.

Additionally, it's possible to provide shared submission access to **authors** of the submission, even if they are not submitters to that Collection. By configuring the `sharedWorkspaceAuthorMetadataFields` bean in `discovery.xml`, it is possible to define which metadata of the Workspace Item must be evaluated to share permissions and give access to other EPersons.

Requirements for the metadata fields:

- The metadata must represent an **authority-controlled Person entity**
- If this Person entity represents a **Researcher Profile**, permission on the Workspace Item will be granted to EPersons owning (`dspace.object.owner` metadata) those Researcher Profiles
- Those EPersons will be able to **edit** Workspace Item metadata

Default Configuration

Out of the box, the logic relies on `dc.contributor.author` and `dc.contributor.editor` metadata:

```
<util:list id="sharedWorkspaceAuthorMetadataFields" value-type="java.lang.String">
    <value>dc.contributor.author</value>
    <value>dc.contributor.editor</value>
</util:list>
```

This configuration is located in `dspace/config/spring/api/discovery.xml`.

Example

A new Workspace item (Publication) is created, submitted by user **John Doe**, having:

- Author "**Jane Smith**" — a Person stored in DSpace, representing the Researcher Profile of EPerson "Jane Smith"
- Author "**Michael Douglas**" — not matching any Researcher Profile
- Author "**Carlos Prontera**" — a Person representing the Researcher Profile of EPerson "Carlos Prontera"

Both "Jane Smith" and "Carlos Prontera" researchers, when logged into the DSpace instance, will see this Workspace Item among "Shared Submissions" in their MyDSpace dashboard.

Permissions Summary

Role	View	Edit Metadata	Delete	Deposit	Upload Files	Accept License
Original submitter	Yes	Yes	Yes	Yes	Yes	Yes
Collection submitter group member (shared workspace)	Yes	Yes	Yes	Yes	Yes	Yes
Co-author + submitter of same collection	Yes	Yes	Yes	Yes	Yes	Yes
Co-author + submitter of different collection	Yes	Yes	**No**	**No**	**No**	**No**
Co-author not in any submitter group	Yes	Yes	**No**	**No**	**No**	**No**

Collection administrator	Yes	Yes	Yes	Yes	Yes	Yes

Known Limitation

Currently co-authors of a Workspace Item **cannot delete or deposit** it. They can only edit its metadata.

Solr Data Management

Solr Core Management

Overview

The **Solr Core Management** script is a utility developed to simplify and standardize the process of exporting and importing data from Solr cores used by DSpace (such as `statistics` or `audit`).

This tool is particularly useful for **maintenance, data migration, and backup/restore operations**, as it provides a consistent way to handle large Solr indexes without requiring direct Solr administrative access.

In production environments, Solr cores can grow significantly over time. Full exports or imports of these cores can be time-consuming and resource-intensive. To address this, the script includes **incremental export capabilities** and **multi-threaded processing**, allowing administrators to manage data more efficiently.

Why This Script Is Used

DSpace installations often rely heavily on Solr for usage statistics, and audit tracking. However, managing Solr cores manually through Solr's admin interface or API can be problematic, especially when dealing with large indexes or multi-year data retention.

This script was designed to:

- **Automate routine Solr maintenance tasks**, such as periodic exports for backup or archival.
- **Enable partial exports/imports** based on time increments (e.g., by week, month, or year), making it possible to handle data in smaller, more manageable chunks.
- **Support parallel processing** to improve performance during heavy operations or large data amount.
- **Standardize import/export formats** (CSV or JSON).

In short, this tool allows administrators to **safely back up, migrate, or rebuild Solr cores** without disrupting DSpace operations or requiring direct low-level Solr commands.

Usage

The script can be executed through the DSpace command-line interface (the script is also available from the UI but requires administrative permission to run the script):

Parameters

```
./dspace solr-core-management
         -m <mode:{export|import}>
         -c <core:{audit|statistics|...}>
         -d <directory>
         [-f <format:{csv|json}>]
         [-t <threads:integer>=1]
         [-s <start-date:yyyy-MM-dd>]
         [-e <end-date:yyyy-MM-dd>]
         [-i <increment:{WEEK|MONTH|YEAR}>]
         [-h]
```

Parameter Description

Parameter	Required	Description
`-m`, `--mode`		Operation mode: either `export` or `import`.
`-c`, `--core`		Name of the Solr core to manage (e.g., `statistics`, `authority`, `audit`).
`-d`, `--directory`		Directory where exported data will be stored or imported from.
`-f`, `--format`		File format for export/import. Supported formats: `csv` (default) or `json`.
`-t`, `--threads`		Number of threads used for parallel processing (default: 1).
`-s`, `--start-date`		Start date (in `yyyy-MM-dd` format) for time-based filtering during export.

`-e`, `--end-date`		End date (in `yyyy-MM-dd` format) for time-based filtering during export.
`-i`, `--increment`		Split the export into time-based chunks: `WEEK`, `MONTH`, or `YEAR`. Useful for very large datasets. (default is `MONTH`)
`-h`, `--help`		Displays help and usage information.

Examples

Export Example

```
./dspace solr-core-management --mode export --core audit --directory /tmp/export --format csv --threads 4 --increment WEEK
```

This command exports the content of the `audit` core into the directory `/tmp/export`, splitting data by **weekly increments**.
The export is performed in **CSV format**, using **4 parallel threads** for faster processing.

Incremental export is useful when the Solr core contains a large volume of records — for example, exporting weekly chunks prevents single massive files and allows resuming operations in case of partial failure.

Import Example

```
./dspace solr-core-management --mode import --core audit --directory /tmp/export --format csv --threads 2
```

This command imports previously exported data (from `/tmp/export`) back into the `audit` Solr core.
It uses **2 threads** to parallelize document ingestion and supports the same format used during export (`csv` or `json`).

This operation is typically used when:

- Rebuilding a Solr core after corruption or reindexing purposes.
- Migrating Solr data between environments (e.g., production test).

Best Practices

- Better **stoping DSpace activities** before performing imports to avoid inconsistencies.
  (this also depends on the data where exporting)
- Run exports with **multiple threads** when working with large datasets to reduce execution time.
  Be aware that multiple thread execution could lead to an increased workload on the Solr installation.

Note

The export process is designed to operate over **date ranges** rather than a single continuous dataset.
This approach serves two purposes:

1. It makes data more **manageable and modular**, allowing administrators to back up or transfer only specific time periods (e.g., weekly or monthly exports).
2. It avoids the need for **deep pagination over very large result sets**, which would require Solr to maintain an explicit sort order (`sort=<sort-field>`), significantly increasing memory usage and query time.

By splitting exports into smaller date-based ranges, the process minimizes Solr load, reduces the risk of timeouts, and ensures that each export segment can be completed efficiently even on heavily populated cores.

Statistics and Metrics

- Exchange usage statistics with IRUS
- DSpace Google Analytics Statistics
- SOLR Statistics
- Matomo Integration

Exchange usage statistics with IRUS

1 Introduction
2 Prerequisite
3 Configuration
4 Re-trying failed attempts

IRUS is being retired at the end of April 2026. See the announcement at https://research.jiscinvolve.org/wp/2025/09/02/preparing-for-changes-to-irus/

Introduction

IRUS (Institutional Repository Usage Statistics) enables Institutional Repositories to share and expose statistics based on the COUNTER standard.
It offers opportunities for benchmarking and acts as an intermediary between repositories and other agencies.

IRUS is currently available in the following areas/countries:

- United Kingdom: https://irus.jisc.ac.uk/
- Australia and New-Zealand: https://irus.jisc.ac.uk/irus-anz/
- United States: https://www.lyrasis.org/programs/Pages/IRUS-US.aspx

Prerequisite

The DSpace server should be able to access the tracker's base production and test URL's.
The tracker's base production URL will depend on the area/country where your repository is located:

- United Kingdom: http://irus.jisc.ac.uk/counter/
- Australia and New-Zealand: https://irus.jisc.ac.uk/counter/anz/
- United States: https://irus.jisc.ac.uk/counter/us/

The tracker's base test URL is common to all areas/countries:

https://irus.jisc.ac.uk/counter/test/

Access to the tracker's base URLs can easily be verified using a *wget* command with the applicable URL, e.g.:

```
wget https://irus.jisc.ac.uk/counter/test/
```

The above command should return a HTTP 200.

Configuration

The IRUS statistics tracker can be configured in the irus-statistics.cfg file which can be found [dspace-src]/dspace/config/modules.

Property	Description	Default
`irus.statistics.tracker.enabled`	Configuration used to enable the IRUS statistics tracker. Set to true to enable.	`false`
`irus.statistics.tracker.type-field`	Metadata field to check if certain items should be excluded from tracking. If empty or commented out, all items are tracked.	
`irus.statistics.tracker.type-value`	The values in the above metadata field that will be considered to be tracked.	
`irus.statistics.tracker.entity-types`	The entity types to be included in the tracking. If left empty, only publication hits will be tracked. If entities are disabled in DSpace (the default in DSpace 7.1), then all Items will be included in tracking.	`Publication`
`irus.statistics.tracker.environment`	The tracker environment determines to which url the statistics are exported (test or prod).	`test`
`irus.statistics.tracker.testurl`	The url to which the trackings are exported when testing. (In theory, this should be https://irus.jisc.ac.uk/counter/test/)	
`irus.statistics.tracker.produrl`	The url to which the trackings are exported in production. (this will depend on your area/country, refer to the Prerequisite section)	
`irus.statistics.tracker.urlversion`	Tracker version	

`irus.statistics.spider.agentregex.url`	External URL pointing to the COUNTER user agents file. The user agents file is downloaded from the provided URL as part of the Apache ant build process. Item views determined by DSpace to have been generated by bots/spiders are not sent to IRUS. Including this additional (and optional) agents file can reduce unnecessary network traffic by reducing the need to transfer view data that will be ignored by IRUS. Example value: https://raw.githubusercontent.com/atmire/COUNTER-Robots/master/generated/COUNTER_Robots_list.txt	
`irus.statistics.spider.agentregex.regexfile`	Location where the user agents file should be downloaded to. The Apache ant build process that retrieves the user agents file from the URL specified above places it in the location specified here. Example value: ${dspace.dir}/config/spiders/agents/COUNTER_Robots_list.txt	

Re-trying failed attempts

If the IRUS tracker is down or some other kind of error should occur preventing DSpace from committing to the tracker, the record is stored in the database in a separate table (`OpenUrlTracker`) that is being created automatically during deployment. Committing these entries can be tried again using the following command.

```
[deployed-dspace]/bin/dspace retry-tracker
```

This will iterate over all the logged entries and retry committing them. If they fail again, they remain in the table, if they succeed, they are removed.

It is strongly advised to schedule this script to be executed daily or weekly (preferable at low load-times during the night or weekend). If there are no failed entries, the script will not perform any actions and exit immediately.

DSpace Google Analytics Statistics

- Google Analytics Support
  - Enabling Google Analytics
  - Configuring Google Analytics
- Google Analytics Reports in DSpace UI
  - Configuration settings for Google Analytics Statistics

Google Analytics development and debugging
- Identifying outgoing events in Chrome Dev Tools
- The GA4 Measurement protocol and event parameters
- Analytics Debug View and Debugger Chrome extension

Google Analytics Support

It is possible to record User Interface traffic by enabling the recording of Google Analytics data within DSpace. DSpace supports Google Analytics 4 and as of July 1, 2024, Google Analytics 4 has replaced Universal Analytics. Also, under GA4 it currently supports the Google Tag (gtag.js), but not Google Tag Manager (GTM). See Where is My Google Tag ID for more information.

Enabling Google Analytics

By default, Google Analytics is disabled in DSpace. To enable it, simply set the value of `google.analytics.key` in either your local.cfg or dspace.cfg:

```
# For Google Analytics 4
google.analytics.key = G-XXXXX
```

When Google Analytics is disabled, you will see 404 responses returned from the REST API whenever the User Interface attempts to access `${dspace.server.url}/api/config/properties/google.analytics.key`. This is expected behavior, as that 404 response is the REST API telling the User Interface that Google Analytics is not configured. When the UI sees that 404 from the REST API, it disables Google Analytics tracking the UI.

Configuring Google Analytics

Additional configuration are provided to allow for enhanced Google Analytics support.

Property:	`google.analytics.buffer.limit`
Example Value:	`google.analytics.buffer.limit = 256`
Informational Note:	Maximum number of events held in the buffer to send to Google Analytics. Used in conjunction with "cron" settings below.
Property:	`google.analytics.cron`
Example Value:	`google.analytics.cron = 0 0/5 * * * ?`
Informational Note:	REQUIRED if you want to send file download events to Google Analytics (where they will be tracked as Google "events"). This defines the schedule for how frequently events tracked on the backend (like file downloads) will be sent to Google Analytics. Syntax is defined at https://www.quartz-scheduler.org/api/2.3.0/org/quartz/CronTrigger.html The above example will run this task every 5 minutes (0 0/5 * * * ?) For Google Analytics 4, you MUST also add the "api-secret" below to support sending download events.
Property:	`google.analytics.api-secret`
Example Value:	`google.analytics.api-secret = mysecret`

Informational Note:	(Only used for Google Analytics 4) Defines a Measurement Protocol API Secret to be used to track interactions which occur outside of the user's browser. This is REQUIRED to track downloads of bitstreams. For more details see https://developers.google.com/analytics/devguides/collection/protocol/ga4 Steps to create your API secret are also available from https://www.monsterinsights.com/docs/how-to-create-your-measurement-protocol-api-secret-in-ga4/
Property:	`google-analytics.bundles`
Example Value:	`google-analytics.bundles = ORIGINAL`
Informational Note:	Which Bundles to include in Bitstream statistics. By default, set to ORIGINAL bundle only.

Google Analytics Reports in DSpace UI

DSpace 7 does not yet support

Google Analytics Reporting is not available in DSpace 7.0. While DSpace 7 can capture statistics via Google Analytics (see above), it is not able to display Google Analytics reports in the DSpace User Interface (like was supported in the XMLUI). It is under discussion as it's unclear how many sites used this feature. See DSpace Release 7.0 Status

As of DSpace version 5.0 it has also become possible to expose that recorded Google Analytics data within DSpace. The data is retrieved from Google using the Google Analytics Reporting API v3. This feature is disabled by default, to enable it please follow the instructions below.

Please read the documentation found at https://developers.google.com/analytics/devguides/reporting/core/v3/ and https://developers.google.com/accounts/docs/OAuth2ServiceAccount. It is the definitive documentation, however, it is over detailed for our purposes so the critical steps are summarised below. The theory is that as a developer you would create a Google project, write your application and store the code in the Google code repository, then create a Google Service Account which your application could use to retrieve data from the Google Analytics API. In our case we already have our application, DSpace, but we still have to go through the motions of creating a project in order to be able to be able to generate the Service Account which we need to allow DSpace to talk to the Google Analytics API.

1. Logon to the Google Developers Console https://console.developers.google.com/project with whatever email address you use to access/manage your existing Google Analytics account(s).
2. Create a new Google Project. The assumption is that you are developing some new software and will make use of the Google code repository. This is not the case but you need to create the skeleton project before you can proceed to the next step.
3. Enable the Analytics API for the project. In the sidebar on the left, expand **APIs & auth**. Next, click **APIs**. In the list of APIs, make sure the status is **ON** for the Analytics API.
4. In the sidebar on the left, select **Credentials**.
5. Select **OAuth / Create new Client ID**, then in the subsequent popup screen select **Service account**. This will automatically generate the required Service Account email address and certificate.
6. Go to your Google Analytics dashboard http://www.google.com/analytics/. Create an account for the newly generated Service Account email address and give it permission to 'Read and Analyze' at account level. See *Note below.
7. The generated certificate needs to be placed somewhere that your DSpace application can access and be referenced as described below in the configuration section..

*Note:- The Google documentation specifies that the Service Account email address should only require 'Read and Analyze' permission. However, it would appear this may not be the case and it may be necessary to grant greater permissions, at least initially.

Configuration settings for Google Analytics Statistics

In the `[dspace.dir]/config/modules/google-analytics.cfg` file review the following fields. These should be either edited directly or overridden in your local.cfg config file (see Configuration Reference).

Property:	google-analytics.application.name
Value:	Dummy Project
Informational Note:	Not sure if this property is required but it was in the example code provided by Google. Please do not delete.
Property:	google-analytics.table.id
Example Value:	ga:12345678

Informational Note:	Logon to the Google Analytics Dashboard and select the Property (or website in plain English) that you wish to target. Then select the Admin section for the property. You should then be able to select the 'view settings' for the view you are interested in. The View ID should replace 12345678 below (note - confusingly the Reporting API documentation refers to the View ID as Table ID).
Property:	google-analytics.account.email
Example Value:	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx@developer.gserviceaccount.com
Informational Note:	The email address automatically generated when you created the Service Account.
Property:	google-analytics.certificate.location
Example Value:	/home/example/dslweb--privatekey.p12
Informational Note:	The certificate file automatically generated when you created the Service Account.
Property:	google-analytics.authorization.admin.usage
Example Value:	true
Informational Note:	Control if the statistics pages should be only shown to authorized users. If enabled, only the administrators for the DSpaceObject will be able to view the statistics. If disabled, anyone with READ permissions on the DSpaceObject will be able to view the statistics.

Google Analytics development and debugging

Identifying outgoing events in Chrome Dev Tools

Through "Inspect Element" in Chrome DevTools, you can identify outgoing requests to Google Analytics filtering on requests starting with "collect?"

The GA4 Measurement protocol and event parameters

Google Analytics 4 Measurement Protocol provides a handy overview of the parameters that are sent along.

Analytics Debug View and Debugger Chrome extension

DebugView documentation: GA4 Monitor events in DebugView

SOLR Statistics

DSpace uses the Apache SOLR application underlying the statistics. SOLR enables performant searching and adding to vast amounts of (usage) data. Unlike previous versions, enabling statistics in DSpace does not require additional installation or customization. All the necessary software is included.

1 What is exactly being logged ?
- 1.1 Common stored fields for all usage events
- 1.2 Unique stored fields for bitstream downloads
- 1.3 Unique stored fields for search queries
- 1.4 Unique stored fields for workflow events

2 Disabling Tracking of Statistics
3 Web User Interface Elements
- 3.1 Pageview and Download statistics
  - 3.1.1 Home page
  - 3.1.2 Community home page
  - 3.1.3 Collection home page
  - 3.1.4 Item home page
- 3.2 Search Query Statistics
- 3.3 Workflow Event Statistics

4 Architecture
5 Configuration settings for Statistics
- 5.1 Pre-1.6 Statistics settings

6 Statistics Administration
- 6.1 Converting older DSpace logs into SOLR usage data
- 6.2 Statistics Client Utility
- 6.3 Anonymizing Statistics

7 Custom Reporting - Querying SOLR Directly
- 7.1 Resources
- 7.2 Examples
  - 7.2.1 Top downloaded items by a specific user

8 Managing the City Database File

What is exactly being logged ?

After the introduction of the SOLR Statistics logging, every pageview and file download is logged in a dedicated SOLR statistics core.

In addition to the already existing logging of pageviews and downloads, DSpace also logs search queries users enter in the DSpace search dialog and workflow events.

DSpace 7.x and 8.x does not yet support all features

In DSpace 7.x & 8.x, only usage statistics (pageview, downloads) are logged. Search statistics and workflow reports (which were available in 6.x and below) are not yet supported. See their related tickets: https://github.com/DSpace/DSpace/issues/2880 and https://github.com/DSpace/DSpace/issues/2851

Workflow Events logging

Only workflow events, initiated and executed by a physical user are being logged. Automated workflow steps or ingest procedures are currently **not** being logged by the workflow events logger.

The logging happens at the server side, and doesn't require a javascript like Google Analytics does, to provide usage data. Definition of which fields are to be stored happens in the file **dspace/solr/statistics/conf/schema.xml**.

Although they are stored in the same index, the stored fields for views, search queries and workflow events are different. A new field, statistics_type determines which kind of a usage event you are dealing with. The three possible values for this field are **view, search** and **workflow.**

```
<field name="statistics_type" type="string" indexed="true" stored="true" required="true" />
```

Common stored fields for all usage events

```
<field name="type" type="integer" indexed="true" stored="true" required="true" />
<field name="id" type="integer" indexed="true" stored="true" required="true" />
<field name="ip" type="string" indexed="true" stored="true" required="false" />
<field name="time" type="date" indexed="true" stored="true" required="true" />
<field name="epersonid" type="integer" indexed="true" stored="true" required="false" />
<field name="continent" type="string" indexed="true" stored="true" required="false"/>
<field name="country" type="string" indexed="true" stored="true" required="false"/>
<field name="countryCode" type="string" indexed="true" stored="true" required="false"/>
<field name="city" type="string" indexed="true" stored="true" required="false"/>
<field name="longitude" type="float" indexed="true" stored="true" required="false"/>
<field name="latitude" type="float" indexed="true" stored="true" required="false"/>
<field name="owningComm" type="integer" indexed="true" stored="true" required="false" multiValued="true"/>
<field name="owningColl" type="integer" indexed="true" stored="true" required="false" multiValued="true"/>
<field name="owningItem" type="integer" indexed="true" stored="true" required="false" multiValued="true"/>
<field name="dns" type="string" indexed="true" stored="true" required="false"/>
<field name="userAgent" type="string" indexed="true" stored="true" required="false"/>
<field name="isBot" type="boolean" indexed="true" stored="true" required="false"/>
<field name="referrer" type="string" indexed="true" stored="true" required="false"/>
<field name="uid" type="uuid" indexed="true" stored="true" default="NEW" />
<field name="statistics_type" type="string" indexed="true" stored="true" required="true" default="view" />
```

The combination of type and id determines which resource (either community, collection, item page or file download) has been requested.

Unique stored fields for bitstream downloads

```
<field name="bundleName" type="string" indexed="true" stored="true" required="false" multiValued="true" />
```

Unique stored fields for search queries

```
<field name="query" type="string" indexed="true" stored="true" required="false" multiValued="true"/>
<field name="scopeType" type="integer" indexed="true" stored="true" required="false" />
<field name="scopeId" type="integer" indexed="true" stored="true" required="false" />
<field name="rpp" type="integer" indexed="true" stored="true" required="false" />
<field name="sortBy" type="string" indexed="true" stored="true" required="false" />
<field name="sortOrder" type="string" indexed="true" stored="true" required="false" />
<field name="page" type="integer" indexed="true" stored="true" required="false" />
```

Unique stored fields for workflow events

```
<field name="workflowStep" type="string" indexed="true" stored="true" required="false" multiValued="true"/>
<field name="previousWorkflowStep" type="string" indexed="true" stored="true" required="false" multiValued="
true"/>
<field name="owner" type="string" indexed="true" stored="true" required="false" multiValued="true"/>
<field name="submitter" type="integer" indexed="true" stored="true" required="false" />
<field name="actor" type="integer" indexed="true" stored="true" required="false" />
<field name="workflowItemId" type="integer" indexed="true" stored="true" required="false" />
```

Disabling Tracking of Statistics

By default, Statistics are captured by the REST API for all visits (page hits) and downloads that occur via both the User Interface and the REST API.

Disabling statistical tracking currently must be done by modifying the backend's Spring configuration in [dspace]/config/spring/rest/event-service-listeners.xml. In that file, you must **comment out** the "SolrLoggerUsageEventListener" in order to disable all tracking

event-service-listeners.xml

```
<beans>
    ...
    <!-- Comment out this bean, as shown below, to disable all tracking of usage statistics in Solr -->
    <!-- Inject the SolrLoggerUsageEventListener into the EventService  -->
    <!--
    <bean class="org.dspace.statistics.SolrLoggerUsageEventListener">
        <property name="eventService" ref="org.dspace.services.EventService"/>
    </bean>
    -->
</beans>
```

After commenting out that bean, you will need to restart Tomcat.

NOTE: This only disables **tracking** statistics in Solr. The "Statistics" link will still appear in the header menu of the User Interface. However, you can limit its visibility by setting it to only be visible to administrative users. Update this configuration in your local.cfg or user-statistics.cfg:

```
# Limit access to Admins, Community/Collection Admins
usage-statistics.authorization.admin.usage = true
```

Statistics Menu link can only be removed by customizing DSpace

At this time, there is no flag to remove the "Statistics" menu link completely. See https://github.com/DSpace/DSpace/issues/9698

Web User Interface Elements

Pageview and Download statistics

In the UI, pageview and download statistics can be accessed from the "Statistics" navigation menu near the header. That statistics page is "context aware", so it will show the usage statistics for whatever page (site, Community, Collection) you are currently on.

If you are not seeing the menu, it's likely that they are only enabled for administrators in your installation. Change the configuration parameter "authorization.admin.usage" in usage-statistics.cfg to false in order to make statistics visible for all repository visitors.

Home page

Starting from the repository homepage, the statistics page displays the top 10 most popular items of the entire repository.

Community home page

The following statistics are available for the community home pages:

- Total visits of the current community home page
- Visits of the community home page over a timespan of the last 7 months
- Top 10 country from where the visits originate
- Top 10 cities from where the visits originate

Collection home page

The following statistics are available for the collection home pages:

- Total visits of the current collection home page
- Visits of the collection home over a timespan of the last 7 months
- Top 10 country from where the visits originate
- Top 10 cities from where the visits originate

Item home page

The following statistics are available for the item home pages:

- Total visits of the item
- Total visits for the bitstreams attached to the item
- Visits of the item over a timespan of the last 7 months
- Top 10 country views from where the visits originate
- Top 10 cities from where the visits originate

Search Query Statistics

Only supported in DSpace 6 and below

Search query statistics are only supported in DSpace 6.x and below at this time. The below screenshots and instructions are for 6.x and will need updating if this feature is ported to later versions of DSpace. See https://github.com/DSpace/DSpace/issues/2880

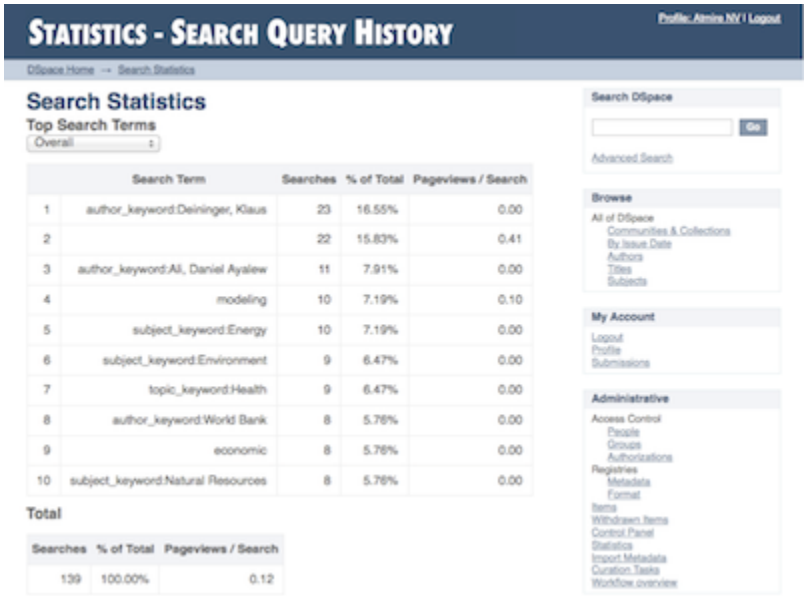
In the UI, search query statistics can be accessed from the lower end of the navigation menu.

If you are not seeing the link labelled "search statistics", it is likely that they are only enabled for administrators in your installation. Change the configuration parameter "authorization.admin.search" in usage-statistics.cfg to false in order to make statistics visible for all repository visitors.

The dropdown on top of the page allows you to modify the time frame for the displayed statistics.

The Pageviews/Search column tracks the amount of pages visited after a particular search term. Therefor a zero in this column means that after executing a search for a specific keyword, not a single user has clicked a single result in the list.

If you are using Discovery, note that clicking the facets also counts as a search, because clicking a facet sends a search query to the Discovery index.



Workflow Event Statistics

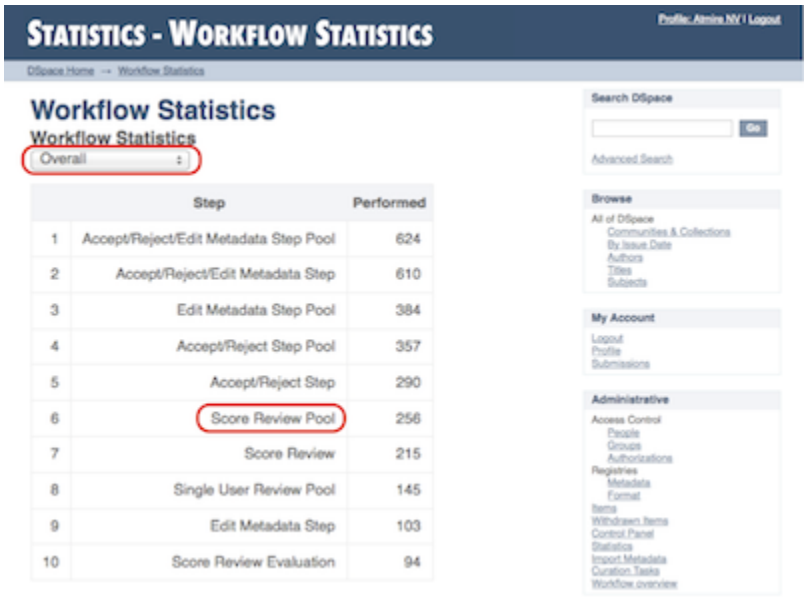
Only supported in DSpace 6 and below

Workflow Event statistics are only supported in DSpace 6.x and below at this time. The below screenshots and instructions are for 6.x and will need updating if this feature is ported to later versions of DSpace. See https://github.com/DSpace/DSpace/issues/2851

In the UI, search query statistics can be accessed from the lower end of the navigation menu.

If you are not seeing the link labelled "Workflow statistics", it is likely that they are only enabled for administrators in your installation. Change the configuration parameter "authorization.admin.workflow" in usage-statistics.cfg to false in order to make statistics visible for all repository visitors.

The dropdown on top of the page allows you to modify the time frame for the displayed statistics.



Architecture

The DSpace Statistics Implementation is a Client/Server architecture based on Solr for collecting usage events in the User Interface or REST API applications of DSpace. Solr must be installed separately from DSpace.

Configuration settings for Statistics

In the `{dspace.dir}/config/modules/solr-statistics.cfg` file review the following fields. These fields can be edited in place, or overridden in your own local.cfg config file (see Configuration Reference).

Property:	solr-statistics.server
Example Values:	solr-statistics.server = http://127.0.0.1/solr/statistics solr-statistics.server = ${solr.server}/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:	solr-statistics.query.filter.bundles
Example Value:	solr-statistics.query.filter.bundles=ORIGINAL
Informational Note:	A comma seperated list that contains the bundles for which the file statistics will be displayed.
Property:	solr-statistics.query.filter.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:	solr-statistics.autoCommit
Example Value:	solr-statistics.autoCommit = true
Informational Note:	If true (default), then all view statistics will be committed to Solr whenever the next autoCommit is triggered. This is recommended behavior. If false, then view statistics will be committed to Solr *immediately* (i.e. via an explicit commit call). This setting is untested in Production scenarios, and is primarily used by automated integration tests (to verify that the statistics engine is working properly).
Property:	solr-statistics.spiderips.urls

Example Value:	solr-statistics.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`

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

In the `{dspace.dir}/config/modules/`**`usage-statistics`**`.cfg` file review the following fields. These fields can be edited in place, or overridden in your own local.cfg config file (see Configuration Reference).

Property:	usage-statistics.dbfile
Example Value:	usage-statistics.dbfile = ${dspace.dir}/config/GeoLite2-City.mmdb
Informational Note:	References the location of the installed GeoLite or DB-IP City "mmdb" database file. This file is utilized by the LocationUtils to calculate the location of client requests based on IP address. NOTE: This database file MUST be downloaded, installed and updated using third-party tools. See the "Managing the City Database File" section below.
Property:	usage-statistics.resolver.timeout
Example Value:	usage-statistics.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 (Set in dspace.cfg)
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:	usage-statistics.authorization.admin.usage
Example Value:	usage-statistics.authorization.admin.usage = true
Informational Note:	When set to true, only general administrators, collection and community administrators are able to access the pageview and download statistics from the web user interface. As a result, the links to access statistics are hidden for non logged-in admin users. Setting this property to "false" will display the links to access statistics to anyone, making them publicly available.
Property:	usage-statistics.authorization.admin.search
Example Value:	usage-statistics.authorization.admin.search = true
Informational Note:	When set to true, only system, collection or community administrators are able to access statistics on search queries.
Property:	usage-statistics.authorization.admin.workflow
Example Value:	usage-statistics.authorization.admin.workflow = true
Informational Note:	When set to true, only system, collection or community administrators are able to access statistics on workflow events.
Property:	usage-statistics.logBots
Example Value:	usage-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:	usage-statistics.shardedByYear
Example Value:	usage-statistics.shardedByYear = false

Information Note:	When set to "true", the DSpace statistics engine will look for additional Solr Shards (per year) when compiling all usage statistics. Therefore, if you are regularly running "stats-utils -s" (as documented in the "Solr Sharding By Year" section of the "SOLR Statistics Maintenance" page), then you should set this to "true". By default, it is "false", which tells the statistics engine to only compile usage statistics based on what is found in the current Solr core.

Pre-1.6 Statistics settings

Only supported in DSpace 6 and below

Search query statistics are only supported in DSpace 6.x and below at this time. See https://github.com/DSpace/DSpace/issues/2852

Older versions of DSpace featured static reports generated from the log files. They still persist in DSpace today but are completely independent from the SOLR based statistics.
The following configuration parameters applicable to these reports can be found in dspace.cfg.

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

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

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

These fields are not used by the new 1.6 Statistics, but are only related to the Statistics from previous DSpace releases

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. As of DSpace 3.0, a script has been added to split up the monolithic SOLR core into individual cores each containing a year of statistics.

Anonymizing Statistics

DSpace provides a commandline script (./dspace anonymize-statistics) which allows you to anonymize your statistics to better comply with GDPR and similar privacy regulations.

The script will anonymise the IP values by rewriting (‘masking’) the last part. This mask is configurable, both for ipv4 and ipv6 addresses.

- For IPv4 addresses, the last number will be replaced by the mask, defined by the configuration key ‘anonymise_statistics.ip_v4_mask’ which defaults to ‘254’.
  For example, 109.74.16.171 is rewritten as 109.74.16.254
- For IPv6 address, the last two numbers will be replaced by the mask, defined by the configuration key ‘anonymise_statistics.ip_v6_mask’ which defaults to ‘FFFF:FFFF’. For example, 2001:0db8:85a3:0000:0000:8a2e:0370:7334 is rewritten as 2001:0db8:85a3:0000:0000:8a2e:FFFF:FFFF

For each anonymised record, the DNS field is also replaced by “anonymised”.

Script options available:

- The program only processes records older than 90 days. This period can be altered with the config ‘anonymise_statistics.time_limit’ (expressed in days) in usage-statistics.cfg.
- "-s [sleep]" : The script takes an optional parameter ‘-s [sleep]’ (expressed in ms), which will make the Java thread sleep between the calls to Solr to reduce the load impact.
- "-t [threads]" : The Solr service commit mechanism is also optimised by adding multi-threading support. The script takes an optional parameter ‘-t [threads]’ to indicate how many threads the Solr service can use for this, if not given the thread count defaults to 2.

Statistical records can also be anonymised the moment they are created. Enabling this feature can be done by setting the configuration parameter "anonymise_statistics.anonymise_on_log" to true in "usage-statististics.cfg" When this configuration property is not set, the feature is disabled by default.

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

- https://www.safaribooksonline.com/library/view/apache-solr-enterprise/9781782161363/
- https://lucidworks.com/blog/faceted-search-with-solr/

Examples

Top downloaded items by a specific user

Query:

```
http://localhost:8983/solr/statistics/select?indent=on&version=2.2&start=0&rows=10&fl=*%
2Cscore&qt=standard&wt=standard&explainOther=&hl.fl=&facet=true&facet.field=epersonid&q=type:0
```

Explained:

facet.field=epersonid — You want to group by epersonid, which is the user id.
type:0 — Interested in bitstreams only

```
<lst name="facet_counts">
    <lst name="facet_fields">
        <lst name="epersonid">
            <int name="66">1167</int>

<int name="117">251</int>

<int name="52">42</int>

<int name="19">36</int>

<int name="88">20</int>

<int name="112">18</int>

<int name="110">9</int>

<int name="96">0</int>

</lst>
    </lst>
</lst>
```

Managing the City Database File

If you wish to record the geographic locations of clients in your DSpace statistics records (e.g. the City or Country where they are accessing your DSpace), you **must** install (and regularly update) one of the following IP to City Database Files (in MMDB format). We recommend installing a City-level database, as it provides more granular location information than a Country-level database (which can only provide the country where the access originated).

- Either install a copy of MaxMind's GeoLite City database (in MMDB format)
  - Installing MaxMind GeoLite2 is *free*. However, you **must** sign up for a (free) MaxMind account in order to obtain a license key to use the GeoLite2 database.
  - You will need to arrange regular downloads of the GeoLite2 database. MaxMind offers an updater tool (geoipupdate) to do the downloading/updating, and a number of Linux distributions package it (as `geoipupdate`). You will still need to configure your license key prior to usage. Use it before restarting DSpace, to get an up-to-date database.
  - Once the "GeoLite2-City.mmdb" database file is installed on your system, you will need to configure its location as the value of `usage-statistics.dbfile` in your `local.cfg` configuration file.
  - NOTE: This file is frequently updated by MaxMind.com, so you will need to refresh it regularly (ideally by scheduling the updater tool via a cron job or similar). As this is written, the database is updated monthly, and to be allowed to obtain it you need to agree to keep your copy updated.
- Or, you can alternatively use/install DB-IP's City Lite database (in MMDB format)
  - This database is also free to use, but does **not** require an account to download.
  - You will need to arrange regular downloads of the City Lite database. DB-IP offers an updater tool (dbip-update) to do the downloading /updating, but it requires PHP to run.
  - Once the "dbip-city-lite.mmdb" database file is installed on your system, you will need to configure its location as the value of `usage-statistics.dbfile` in your `local.cfg` configuration file.
  - NOTE: This file is frequently updated by DB-IP.com, so you will need to refresh it regularly (ideally by scheduling the updater tool via a cron job or similar). As this is written, the database is updated monthly with the latest available at https://db-ip.com/db/download/ip-to-city-lite

SOLR Statistics Maintenance

1 DSpace Log Converter
2 Filtering and Pruning Spiders
3 Export SOLR records to intermediate format for import into another tool/instance
4 Export SOLR statistics, for backup and moving to another server
5 Import SOLR statistics, for restoring lost data or moving to another server
6 Reindex SOLR statistics, for upgrades or whenever the Solr schema for statistics is changed
7 Upgrade Legacy DSpace Object Identifiers (pre-6x statistics) to DSpace 6x UUID Identifiers
8 Solr Sharding By Year
    8.1 Technical implementation details
    8.2 Testing Solr Shards

DSpace Log Converter

The use of Solr for statistics in DSpace makes it possible to have a database of statistics. With this in mind, there is the issue of the older log files and how a site can use them. The following command process is able to convert the existing log files and then import them for Solr use. The user will need to perform this conversion only once.

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

Command used:	`[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 if `-i dspace.log -m` was specified, dspace.log* would be converted. (i.e. all of the following: *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 or newer
`-v` or `--verbose`	Display verbose output (helpful for debugging)
`-h` or `--help`	Help

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

Command used:	`[dspace]/bin/dspace stats-log-importer`
Java class:	org.dspace.statistics.util.StatisticsImporter
Arguments (short and long forms):	Description
`-i` or `--in`	input file
`-m` or `--multiple`	Adds a wildcard at the end of the input, so it would mean dspace.log* would be imported
`-s` or `--skipdns`	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 `--verbose`	Display verbose ouput (helpful for debugging)
`-l` or `--local`	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`	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 for spider removal operations, after converting your old logs.

Filtering and Pruning Spiders

Command used:	`[dspace]/bin/dspace stats-util`
Java class:	org.dspace.statistics.util.StatisticsClient
Arguments (short and long forms):	Description
`-b` or `--reindex-bitstreams`	Reindex the bitstreams to ensure we have the bundle name
`-r` or `--remove-deleted-bitstreams`	While indexing the bundle names remove the statistics about deleted bitstreams
`-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 Configuration settings for Statistics
`-f` or `--delete-spiders-by-flag`	Delete Spiders in Solr By isBot Flag. Will prune out all records that have `isBot:true`
`-m` or `--mark-spiders`	Update isBot Flag in Solr. Marks any records currently stored in statistics that match entries 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 you want to keep spider entries in your repository, you can just mark them using "`-m`" and they will be excluded from statistics queries when "`solr.statistics.query.filter.isBot = true`" in the `dspace.cfg`. If you want to keep the spiders out of the solr repository, just use the "`-f`" option and they will be removed immediately.

Spider IPs are specified in files containing one pattern per line. A line may be a comment (starting with "#" in column 1), empty, or a single IP address or DNS name. If a name is given, it will be resolved to an address. Unresolvable names are discarded and will be noted in the log.

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

Spiders may also be excluded by DNS name or Agent header value. Place one or more files of patterns in the directories `[dspace]/config/spiders/domains` and/or `[dspace]/config/spiders/agents`. Each line in a pattern file should be either empty, a comment starting with "#" in column 1, or a regular expression which matches some names to be recognized as spiders.

Export SOLR records to intermediate format for import into another tool/instance

Command used:	`[dspace]/bin/dspace stats-util`
Java class:	org.dspace.statistics.util.StatisticsClient
Arguments (short and long forms):	Description
`-e` or `--export`	Export SOLR view statistics data to usage statistics intermediate format

This exports the records to `[dspace]/temp/usagestats_0.csv`. This will chunk the files at 10,000 records to new files. This can be imported with `stats-log-importer` to SOLR Statistics

Export SOLR statistics, for backup and moving to another server

Command used:	`[dspace]/bin/dspace solr-export-statistics`
Java class:	org.dspace.util.SolrImportExport
Arguments (short and long forms):	Description
`-i` or `--index-name`	optional, the name of the index to process. "statistics" is the default. "authority" can also be exported.
`-l or --last` *`integer`*	optionally export only *integer* many days worth of statistics
`-d or --directory`	optional, directory to use for storing the exported files. By default, `[dspace]/solr-export` is used. If that is not appropriate (due to storage concerns), we recommend you use this option to specify a more appropriate location.
`-f` or `--force-overwrite`	optional, overwrite export file if it exists (DSpace 6.1 and later)

Import SOLR statistics, for restoring lost data or moving to another server

Command used:	`[dspace]/bin/dspace solr-import-statistics`
Java class:	org.dspace.util.SolrImportExport
Arguments (short and long forms):	Description
`-i` or `--index-name`	optional, the name of the index to process. "statistics" is the default. "authority" can also be imported.
`-c or --clear`	optional, clears the contents of the existing stats core before importing
`-d or --directory`	optional, directory which contains the files for importing. By default, `[dspace]/solr-export` is used. If that is not appropriate (due to storage concerns), we recommend you use this option to specify a more appropriate location.

Reindex SOLR statistics, for upgrades or whenever the Solr schema for statistics is changed

Command used:	`[dspace]/bin/dspace solr-reindex-statistics`
Java class:	org.dspace.util.SolrImportExport
Arguments (short and long forms):	Description
`-i` or `--index-name`	optional, the name of the index to process. "statistics" is the default
`-k or --keep`	optional, tells the script to keep the intermediate export files for possible later use (by default all exported files are removed at the end of the reindex process).
`-d or --directory`	optional, directory to use for storing the exported files (temporarily, unless you also specify `--keep`, see above). By default, `[dspace]/solr-export` is used. If that is not appropriate (due to storage concerns), we recommend you use this option to specify a more appropriate location. Not sure about your space requirements? You can estimate the space required by looking at the current size of `[dspace]/solr/statistics`
`-f` or `--force-overwrite`	optional, overwrite export file if it exists (DSpace 6.1 and later)

NOTE: `solr-reindex-statistics` is safe to run on a live site. The script stores incoming usage data in a temporary SOLR core, and then merges that new data into the reindexed data when the reindex process completes.

Upgrade Legacy DSpace Object Identifiers (pre-6x statistics) to DSpace 6x UUID Identifiers

This command was introduced in **DSpace 7.0** and will be included in the **DSpace 6.4** release as well.

It is recommended that all DSpace instances with legacy identifiers perform this one-time upgrade of legacy statistics records.

This action is safe to run on a live site. As a precaution, it is recommended that you backup you statistics shards before performing this action.

Note: a link to this section of the documentation should be added to the DSpace 6.4 Release Notes. (It is already noted in the DSpace 7.0 Upgrading DSpace page, step 11d)

The DSpace 6x code base changed the primary key for all DSpace objects from an integer id to UUID identifiers. Statistics records that were created before upgrading to DSpace 6x contain the legacy identifiers.

While the DSpace user interfaces make some attempt to correlate legacy identifiers with uuid identifiers, it is recommended that users perform this one time upgrade of legacy statistics records.

If you have sharded your statistics repository, this action must be performed on each shard.

Command used:	`[dspace]/bin/dspace solr-upgrade-statistics-6x`
Java class:	org.dspace.util.SolrUpgradePre6xStatistics
Arguments (short and long forms):	Description
`-i` or `-index-name`	Optional, the name of the index to process. "statistics" is the default

`-n or --num_rec`	Optional. Total number of records to update (defaut=100,000). To process all records, set -n to 10000000 or to 100000000 (10M or 100M) If possible, please allocate 2GB of memory to this process (e.g. -Xmx2000m)
`-b or --batch_size`	Number of records to batch update to SOLR at one time (default=10,000).

NOTE: This process will rewrite most solr statistics records and may temporarily double the size of your statistics repositories.

If a UUID value cannot be found for a legacy id, the legacy id will be converted to the form "xxxx-unmigrated" where xxxx is the legacy id.

Solr Sharding By Year

The DSpace tool described below for managing Solr data through yearly sharding no longer functions in DSpace 7.x (see also https://github.com/DSpace/DSpace/issues/8478). Using these tools to manage Solr shards is no longer recommended. Alternative approaches are being explored and this page will be updated to reflect those findings.

Command used:	`[dspace]/bin/dspace stats-util`
Java class:	org.dspace.statistics.util.StatisticsClient
Arguments (short and long forms):	Description
`-s` or `--shard-solr-index`	Splits the data in the main Solr core up into a separate core for each year. This will upgrade the performance of Solr.

Notes:

Yearly Solr sharding is a routine that can drastically improve the performance of your DSpace SOLR statistics. It was introduced in DSpace 3.0 and is not backwards compatible. The routine decreases the load created by the logging of new usage events by reducing the size of the SOLR Core in which new usage data are being logged. By running the script, you effectively split your current SOLR core, containing all of your usage events, into different SOLR cores that each contain the data for one year. In case your DSpace has been logging usage events for less than one year, you will see no notable performance improvements until you run the script after the start of a new year. Both writing new usage events as well as read operations should be more performant over several smaller SOLR Shards instead of one monolithic one.

It is highly recommended that you execute this script once at the start of every year. To ensure this is not forgotten, you can include it in your crontab or other system scheduling software. Here's an example cron entry (just replace [dspace] with the full path of your DSpace installation):

```
# At 12:00AM on January 1, "shard" the DSpace Statistics Solr index.  Ensures each year has its own Solr index
- this improves performance.
0 0 1 1 * [dspace]/bin/dspace stats-util -s
```

You MUST restart Tomcat after sharding

After running the statistics shard process, the "View Usage Statistics" page(s) in DSpace will **not** automatically recognize the new shard.

Restart tomcat to ensure that the new shard is recognized & included in usage statistics queries.

Repair of Shards Created Before DSpace 5.7 or DSpace 6.1

If you ran the shard process before upgrading to DSpace 5.7 or DSpace 6.1, the multi-value fields such as owningComm and onwningColl are likely be corrupted. Previous versions of the shard process lost the multi-valued nature of these fields. Without the multi-valued nature of these fields, it is difficult to query for statistics records by community / collection / bundle.

You can verify this problem in the solr admin console by looking at the owningComm field on existing records and looking for the presence of "\\," within that field.

The following process may be used to repair these records.

1. Backup your solr statistics-xxxx directories while tomcat is down.
2. Backup and delete the contents of the dspace-install/solr-export directory
3. For each "statistics-xxxx" shard that exists, export the repository

```
dspace solr-export-statistics -i statistics-xxxx -f
```

4. Run the following to repair records in the dspace-install/solr-export directory

```
for file in *
do
sed -E -e "s/[\\]+,/,/g" -i $file
done
```

5. For each shard that was exported, run the following import

```
dspace solr-import-statistics -i statistics-xxxx -f
```

If you repeat the query that was run previously, the fields containing "\\," should now contain an array of owning community ids.

Shard Naming

Prior to the release of DSpace 6.1, the shard names created were off by one year in timezones with a positive offset from GMT.

Shar

See Unable to locate Jira server for this macro. It may be due to Application Link configuration.

Technical implementation details

After sharding, the Solr data cores are located in the [dspace.dir]/solr directory. There is no need to define the location of each individual core in solr.xml because they are automatically retrieved at runtime. This retrieval happens in the *static* method located in the *org.dspace.statistics.SolrLogger* class. These cores are stored in the *statisticYearCores* list. Each time a query is made to Solr, these cores are added as shards by the *addAdditionalSolrYearCores* method. The cores share a common configuration copied from your original *statistics* core. Therefore, no issues should be resulting from subsequent an`t updates`.

The actual sharding of the of the original Solr core into individual cores by year is done in the *shardSolrIndex* method in the *org.dspace.statistics.SolrLogger* class. The sharding is done by first running a facet on the time to get the facets split by year. Once we have our years from our logs we query the main Solr data server for all information on each year & download these as CSVs. When we have all data for one year, we upload it to the newly created core of that year by using the update csv handler. Once all data of one year have been uploaded, those data are removed from the main Solr (by doing it this way if our Solr crashes we do not need to start from scratch).

Multiple Shard Fix (DSpace 6.1)

A bug exists in the DSpace 6.0 release that prevents tomcat from starting when multiple shards are present.

To address this issue, the initialization of SOLR shards is deferred until the first SOLR related requests are processed.

See Unable to locate Jira server for this macro. It may be due to Application Link configuration.

Testing Solr Shards

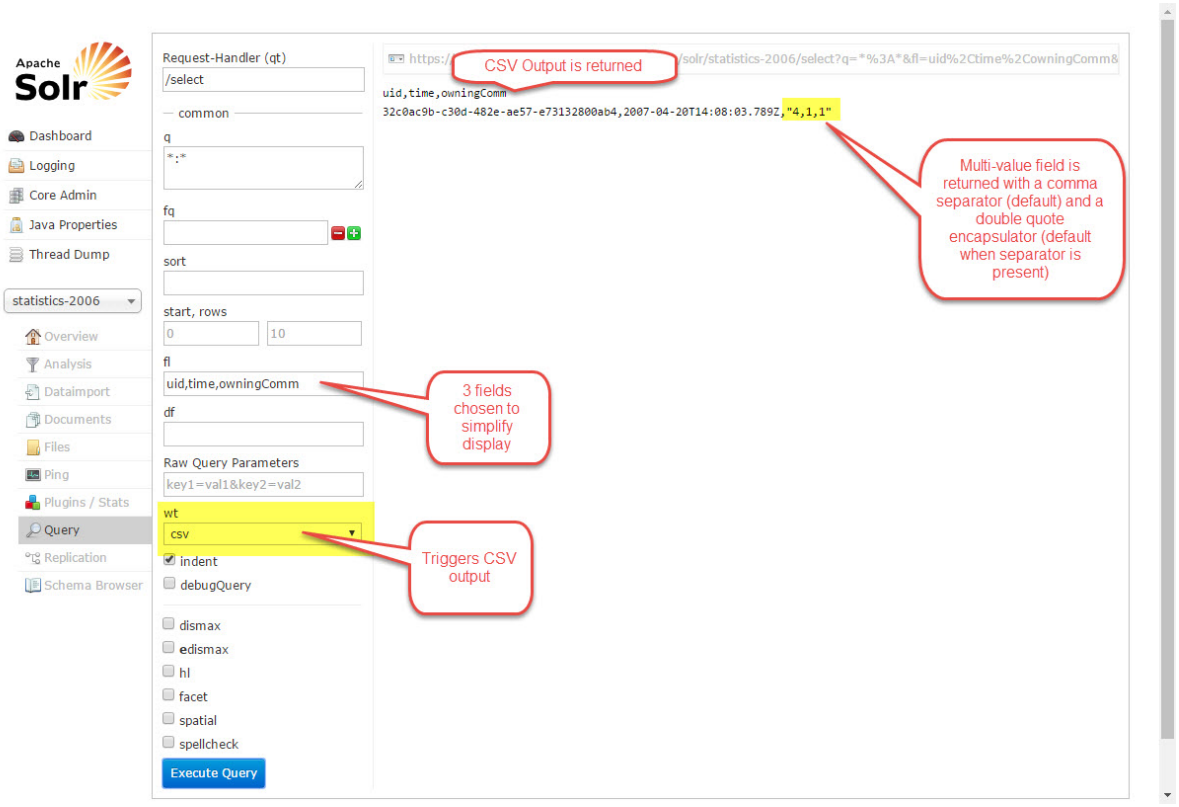
Testing Solr Shards

Testing Solr Shards

These notes detail how to test and manipulate SOLR statistics shards.

Testing CSV Export

The SOLR Admin Console provides a mechanism to test the CSV Export Process and Parameters

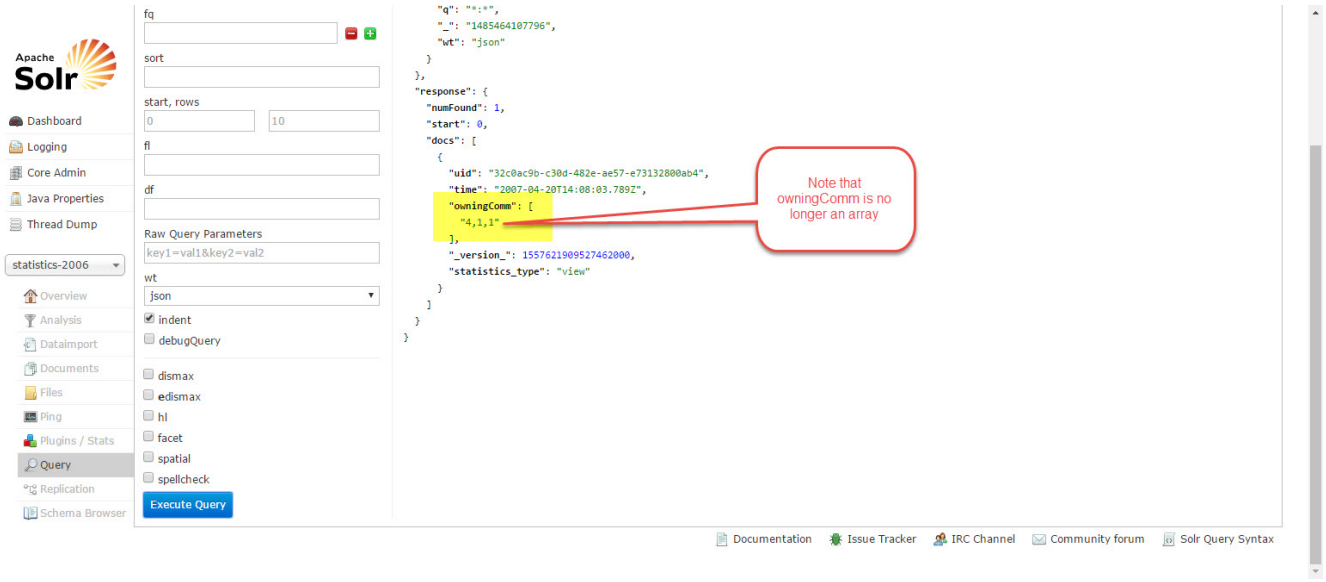


Testing CSV Import

The SOLR Admin Console provides a mechanism to access the CSV Upload process. Unfortunately, it does not all parameters to be provided.



Note that the multi-value field is corrupted if you import by this manner.



It is possible to csv import parameters using curl.

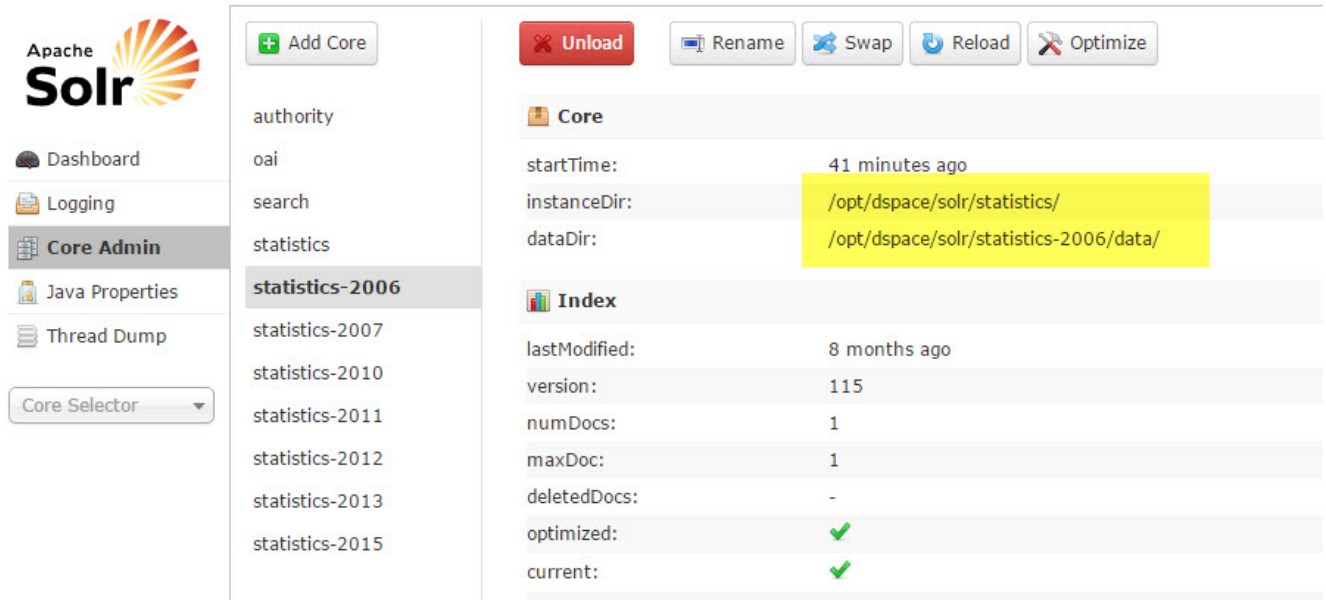
Running CSV Upload with curl

```
curl -F "data=@statistics-2006_export_2007-04.csv" "http://localhost/solr/statistics-2006/update/csv?skip=_version_&csv.mv.escape=%5C&f.owningColl.split=true&f.owningColl.separator=%7C&f.owningComm.split=true&f.owningComm.separator=,&f.owningItem.split=true&f.owningItem.separator=%7C&f.bundleName.split=true&f.bundleName.separator=%7C&stream.contentType=text%2Fcsv%3Bcharset%3Dutf-8&commit=true&softCommit=false&waitSearcher=true&wt=javabin&version=2"
```

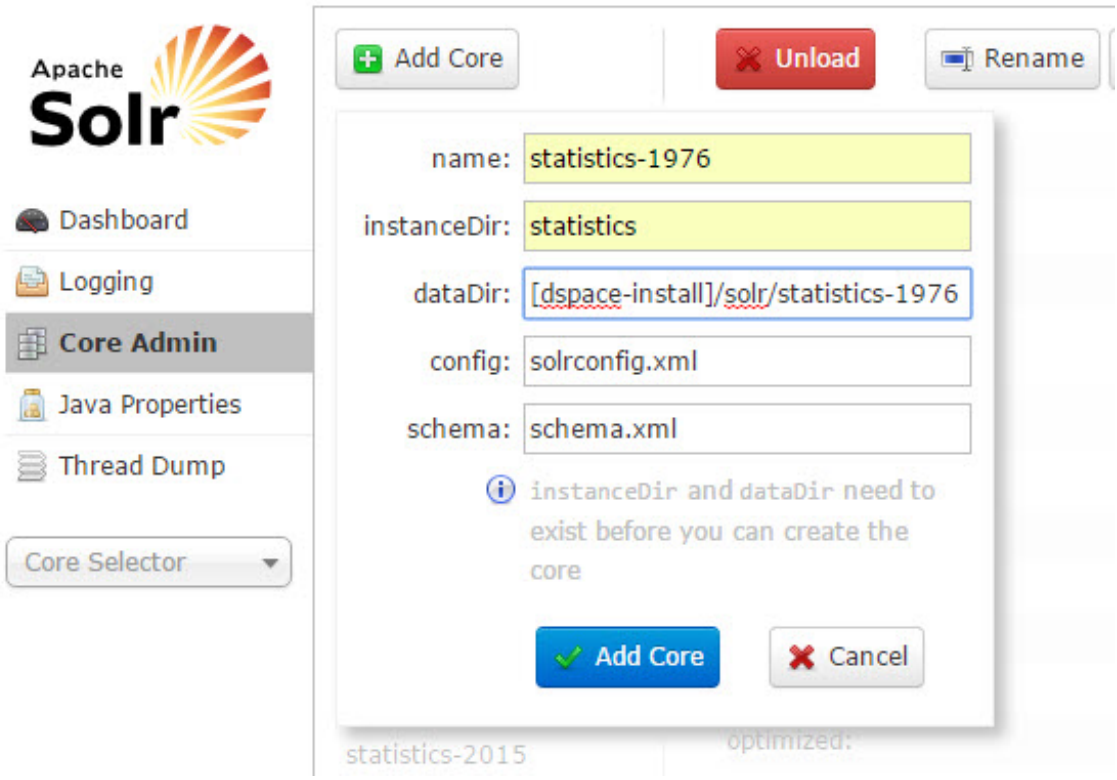
Creating a Shard in the Admin Console

While this is probably not necessary, it is possible to create an empty shard in the Solr Admin console.

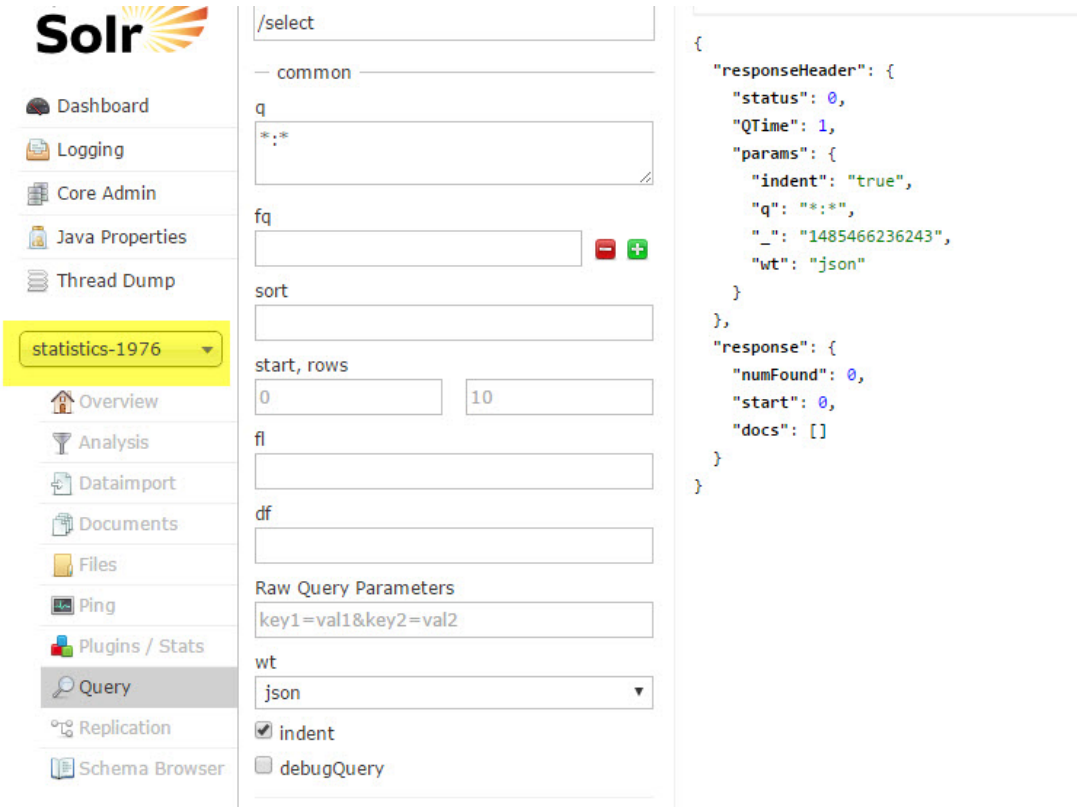
Note that existing shards use the statistics directory as an "instance" directory.



Manually create a new shard



The new shard can be queried like the other ones



Matomo Integration

Integration details
Configuration
- Matomo REST
- Matomo Docker
- Matomo Angular

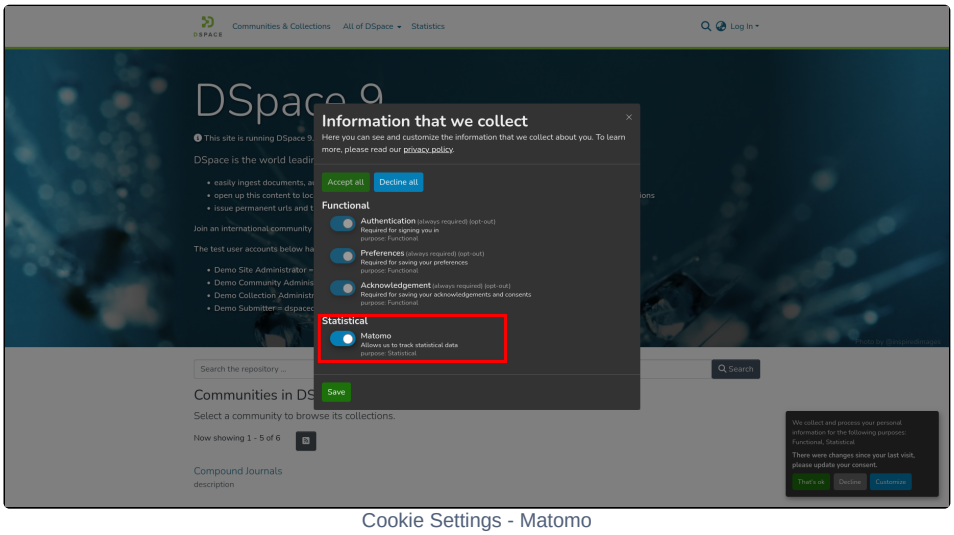
Integration details

This integration allows DSpace to track and record `UsageEvents` with Matomo Analytics, a GDPR-compliant, privacy-first Google Analytics alternative. This allows DSpace to leverage Matomo's capabilities for enhanced tracking and analysis.

There are two main parts of this development:

- the REST part is responsible for tracking `Bitstream` downloads, and links them with:
- the `angular` integration that is responsible for sending the events for the site navigation, `Item` view and search.

On the first navigation you will be notified about cookies. Once opened up to customize them, you can choose which cookies are going to be installed.

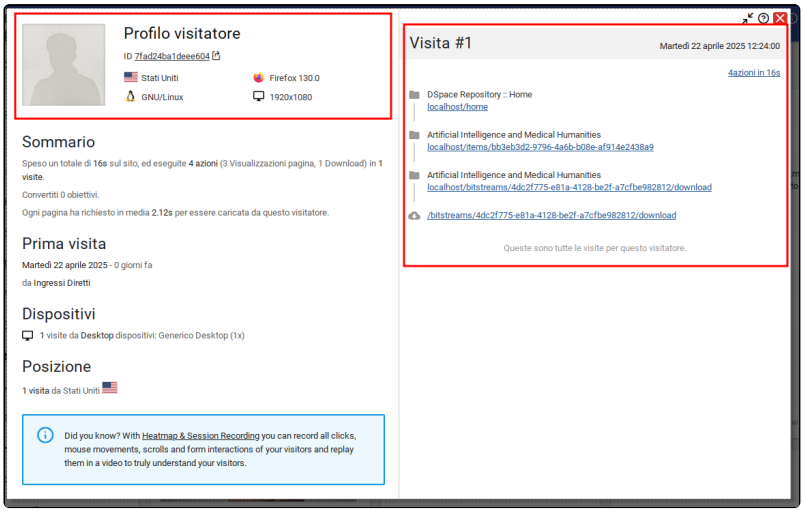


Cookie Settings - Matomo

The integration would track these actions:

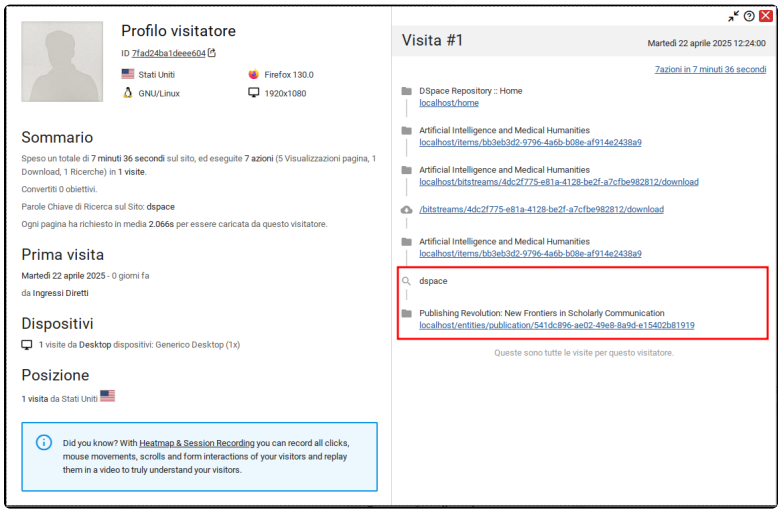
- Page navigation on the Site
- Search performed
- Bitstream download

Take a look at this tracked event:



Here you can see that the user (*profiled on the left side*) has been involved into page navigations and then downloaded a bitstream.

Then performed a search by *"dspace"* keyword, as you can see here:



So, generally once the cookies have been set on the user's browser all the interactions of that user within the system will be tracked. You can see all those interactions by opening the Matomo dashboard (by default if you're using the Docker configuration would be matomo-dashboard)

Configuration

Matomo REST

The configuration can be checked inside the file `matomo.cfg`

matomo.cfg

```
#------------------------------------------------------------------#
#---------------MATOMO TRACKER CONFIGURATION-------------------#
#------------------------------------------------------------------#

# Enables matomo integration mapped inside the MatomoEventListener
matomo.enabled = false
# Configured `siteid` inside the matomo dashboard
matomo.request.siteid = 1
# Configured tracker.url that will be used by the Angular application
matomo.tracker.url = http://localhost:8081
# Specifies bitstream's bundle that will be tracked ( default is ORIGINAL )
# Add 'none' to disable the tracking for bitstreams
# matomo.track.bundles = ORIGINAL

#------------------------------------------------------------------#
#---------------MATOMO CLIENTS CONFIGURATION-------------------#
#------------------------------------------------------------------#
#  Here you will configure settings of beans that will be used  #
#  as client inside matomo.xml                                  #
#                                                               #
#################################################################

#------------------------------------------------------------------#
#---------------DEFAULT CLIENT CONFIGURATION-------------------#
#------------------------------------------------------------------#
# The tracking endpoint of matomo
matomo.default-client.baseurl = ${matomo.tracker.url}/matomo.php
# The auth-token configured inside the matomo instance should have the pattern [a-f0-9]{32}
# matomo.default-client.token = e6c21371cd97bebe45e8e729edfb202c

#------------------------------------------------------------------#
#-----------------ASYNC CLIENT CONFIGURATION-------------------#
#------------------------------------------------------------------#
# The tracking endpoint of matomo
matomo.async-client.baseurl = ${matomo.tracker.url}/matomo.php
# The auth-token configured inside the matomo instance should have the pattern [a-f0-9]{32}
# matomo.async-client.token = e6c21371cd97bebe45e8e729edfb202c
```

The main configuration to enable the all the features is `matomo.enabled` that must be set to `true`.

Once enabled, you should check also the `matomo.siteId` and the `matomo.tracker.url`. Both of them must be aligned with your current installation. By default they're set to a local instance started with the `docker-compose-matomo` file.

If you need to customize something you can have a look to the bean definitions inside `matomo.xml` file:

matomo.xml

```
    <bean id="matomoRequestDetailsBuilder" class="org.dspace.matomo.model.MatomoRequestDetailsBuilder">
        <constructor-arg name="enrichers">
            <list>
                <bean id="userAgentEnricher" class="org.dspace.matomo.factory.
MatomoRequestDetailsEnricherFactory" factory-method="userAgentEnricher"/>
                <bean id="actionNameEnricher" class="org.dspace.matomo.factory.
MatomoRequestDetailsEnricherFactory" factory-method="actionNameEnricher"/>
                <bean id="urlEnricher" class="org.dspace.matomo.factory.MatomoRequestDetailsEnricherFactory"
factory-method="urlEnricher"/>
                <bean id="downloadEnricher" class="org.dspace.matomo.factory.
MatomoRequestDetailsEnricherFactory" factory-method="downloadEnricher"/>
                <bean id="trackerIdEnricher" class="org.dspace.matomo.factory.
MatomoRequestDetailsEnricherFactory" factory-method="trackerIdentifierEnricher"/>
                <!-- As of now these cookie enrichers are not working, because we are not treating cross-domain
Matomo Cookies -->
<!--
                <bean id="cookieIdEnricher" class="org.dspace.matomo.factory.
MatomoRequestCookieIdentifierEnricher"/>
                <bean id="cookieSessionEnricher" class="org.dspace.matomo.factory.
MatomoRequestCookieSessionEnricher"/>
                <bean id="customVariablesEnricher" class="org.dspace.matomo.factory.
MatomoRequestCustomVariablesEnricher"/>
                <bean id="customCookiesEnricher" class="org.dspace.matomo.factory.
MatomoRequestCustomCookiesEnricher">
                        <constructor-arg name="customCookies" value="${matomo.request.customcookies.cookie-
name}"/>
                </bean>
-->
                <!-- These two enrichers requires a valid token set on the client -->
<!--                <bean id="ipEnricher" class="org.dspace.matomo.factory.MatomoRequestIpAddressEnricher"/>-->
<!--                <bean id="requestCountryEnricher" class="org.dspace.matomo.factory.
MatomoRequestCountryEnricher"/>-->
            </list>
        </constructor-arg>
        <constructor-arg name="siteId" value="${matomo.request.siteid}"/>
    </bean>
```

This first portion specifies how the request should be composed, depending on the enrichers configured, and also identifies the siteId to be used.

Each `ViewEvent` received on the REST side would be parsed and then sent to the Matomo tracker by using the following components:

matomo.xml

```
    <bean id="matomoAsyncClient" class="org.dspace.matomo.client.MatomoAsyncClientImpl">
        <constructor-arg name="baseUrl" value="${matomo.async-client.baseurl}"/>
        <!-- Optional, so the default value will be null -->
        <constructor-arg name="token" value="${matomo.async-client.token:}"/>
        <constructor-arg name="matomoRequestBuilder" ref="matomoRequestBuilder"/>
        <constructor-arg name="matomoResponseReader" ref="matomoResponseReader"/>
    </bean>

    <!-- This Sync event handler tracks each view event using the async client that is non-blocking -->
    <!--
        You can use any combination of EventHandler / MatomoClient, all the options available are listed down
here
        but commented, you can choose the one you prefer.
    -->
    <bean id="matomoSyncEventHandler" class="org.dspace.matomo.MatomoSyncEventHandler">
        <constructor-arg name="matomoClient" ref="matomoAsyncClient"/>
        <constructor-arg name="builder" ref="matomoRequestDetailsBuilder"/>
    </bean>
```

event-service-listeners.xml

```
<!-- Matomo events recording -->
<bean class="org.dspace.matomo.MatomoEventListener">
    <property name="eventService" ref="org.dspace.services.EventService"/>
</bean>
```

Each event is intercepted by the `MatomoEventListener` and processed by the `MatomoSyncEventHandler` that generates a proper request using the configured enrichers (`MatomoRequestDetailsBuilder`) and sends them to the Matomo tracker using the `MatomoAsyncClientImpl`

Matomo Docker

If you would like to try it out, you must start the Docker instance of DSpace and Matomo using the proper `docker-compose` commands.

If you need to setup a DSpace from scratch, you must build the `Dockerfile` with the command:

build

```
docker build -t dspace:latest .
```

Then, you must ingest it with some data by running the command:

ingest-dspace

```
docker compose -p d9 -f docker-compose.yml -f dspace/src/main/docker-compose/db.entities.yml up -d
```

So, finally you can startup all together with the Matomo container that uses the port 8081 to expose its API and User Interface.

matomo-compose

```
docker compose -p d9 -f docker-compose.yml -f dspace/src/main/docker-compose/docker-compose-matomo.yml up -d
```

Once started you can complete the Matomo configuration directly from the Matomo Home Page.

1. You need to:
    - Create a proper user
    - Create a proper siteId - The first one should be always set to 1 (`matomo.request.siteId` on `matomo.cfg`)
    - Create a siteId - Matomo-Docker-SiteId and save it property.
    - Create a new authToken - Matomo-Docker-AuthToken and save it properly. Be careful since this token shouldn't set to allow only secured requests otherwise it won't work.
2. You need to set check these properties inside the `matomo.cfg` file:

    matomo.cfg

    ```
    # Enables matomo integration mapped inside the MatomoEventListener
    matomo.enabled = true
    # Configured `siteid` inside the matomo dashboard
    matomo.request.siteid = 1
    # The tracking endpoint of matomo
    matomo.async-client.baseurl = http://localhost:8081/matomo.php
    ```

3. Startup / Restart DSpace and try to download a bitstream that has been placed inside the ORIGINAL bundle (default bundle configured for Matomo integration)
4. You should see a mapped request inside your Matomo dashboard with the bitstream details - Matomo-Docker-Dashboard

Matomo Angular

On the angular side there is a configuration that you would need to apply to your `yml` file:

matomo-angular

```
matomo:
  siteId: 1
  trackerUrl: 'http://localhost:8081'
```

The `siteId` and the `trackerUrl` must be compatible with the one set on the REST side, since they need to communicate with the same Matomo instance.

If you start the env using Docker, those two addresses are different because internally the Docker Matomo container has a different address.

User Interface

- User Interface Configuration
- User Interface Customization
- Accessibility
- Browse
- Contextual Help Tooltips
- Discovery
- IIIF Configuration
- Multilingual Support
- User Interface Debugging

User Interface Configuration

- Overview
- Configuration File Format
- Configuration Override
- Configuration Reference
    - UI Core Settings
    - Server Side Rendering (SSR) Settings
    - REST API Settings
    - Cache Settings - General
    - Cache Settings - Server Side Rendering (SSR)
    - Authentication Settings
    - Form Settings
    - Notification Settings
    - Submission Settings
    - Language Settings
    - Browse By Settings
    - Community-List Settings
    - Homepage Settings
    - Undo Settings
    - Item Access Labels
    - Item Page Settings
    - Community Page Settings
    - Collection Page Settings
    - Theme Settings
    - Bundle Settings
    - Media Viewer Settings
        - Uploading video captioning files
    - Toggle end-user agreement and privacy policy
    - Settings for rendering Markdown, HTML and MathJax in metadata
    - Controlled Vocabularies in Search Filters
    - Search settings
    - Layout settings
    - Geospatial map viewer settings
    - Debug Settings

Overview

As the DSpace User Interface is built on Angular.io, it aligns with many of the best practices of that platform & the surrounding community. One example is that our UI uses the TypeScript language. That said, you do NOT need to be deeply familiar with TypeScript to edit themes or other configuration.

Configuration File Format

The configuration format is YAML and is located at `./config/config.*.yaml`.

Configuration Override

The UI configuration files reside in the `./config/` folder in the Angular UI source code. The default configuration is provided in `config.yml`.

To change the default configuration values, you simply create (one or more) local files that override the parameters you need to modify. You can use `config.example.yml` as a starting point.

- For example, create a new `config.dev.yml` file in `config/` for a `development` environment;
- For example, create a new `config.prod.yml` file in `config/` for a `production` environment;

Configurations can also be overridden via one of the following

- Using Environment variables. All environment variables MUST (1) be prefixed with "DSPACE_", (2) use underscores as separators (no dots allowed), and (3) use all uppercase. Some examples are below:

```
# "ui" settings environment variables
ui.ssl => DSPACE_UI_SSL # Whether the angular application uses SSL [true/false]
ui.host => DSPACE_UI_HOST # The host name
ui.port => DSPACE_UI_PORT # The port number
ui.nameSpace => DSPACE_UI_NAMESPACE # The namespace
ui.baseUrl => DSPACE_UI_BASEURL  # The Base URL for SSR

# "rest" settings environment variables
rest.ssl => DSPACE_REST_SSL # Whether the angular REST uses SSL [true/false]
rest.host => DSPACE_REST_HOST # The host name of the REST application
rest.port => DSPACE_REST_PORT # The port number of the REST application
rest.nameSpace => DSPACE_REST_NAMESPACE # The namespace of the REST application

# Other examples
fallbackLanguage => DSPACE_FALLBACKLANGUAGE
mediaViewer.video => DSPACE_MEDIAVIEWER_VIDEO

# Multi-valued setting examples
# If a setting can have multiple values (e.g. theme names), then use an index number (starting with zero)
# to specify the multiple values.
# The below example is equivalent to:
# themes:
#   - name: 'dspace'
#   - name: 'mytheme'
#     handle: '10673/123'
DSPACE_THEMES_0_NAME = 'dspace'
DSPACE_THEMES_1_NAME = 'mytheme'
DSPACE_THEMES_1_HANDLE = '10673/123'
```

- Or, by creating a `.env` (environment) file in the project root directory and setting the environment variables in that location.

The override priority ordering is as follows (with items listed at the top overriding all other settings)

1. Environment variables
2. The `.env` file
3. The `./config/config.prod.yml`, `./config/config.dev.yml` or `./config/config.test.yml` files (depending on current mode)
4. The `./config/config.yml` file
5. The hardcoded defaults in `./src/config/default-app-config.ts`

Configuration Reference

The following configurations are available in `./config/config.example.yml` These settings may be overridden as described above.

UI Core Settings

The "ui" (user interface) section defines where you want Node.js to run/respond. It may correspond to your primary/public URL, but it also may not (if you are running behind a proxy). In this example, we are setting up our UI to just use localhost, port 4000. This is a common setup for when you want to use Apache or Nginx to handle HTTPS and proxy requests to Node.js running on port 4000.

config.*.yml

```
ui:
  ssl: false
  host: localhost
  port: 4000
  # NOTE: Space is capitalized because 'namespace' is a reserved string in TypeScript
  nameSpace: /
  # Specify the public URL that this user interface responds to. This corresponds to the "dspace.ui.url"
property in your backend's local.cfg.
  # SSR is only enabled when the client's "Host" HTTP header matches this baseUrl. The baseUrl is also used for
redirects and SEO links (in robots.txt).
  baseUrl: http://localhost:4000
  # The rateLimiter settings limit each IP to a 'max' of 500 requests per 'windowMs' (1 minute).
  rateLimiter:
    windowMs: 60000 # 1 minute
    max: 500 # limit each IP to 500 requests per windowMs
```

The "rateLimiter" sub-section can be used to protect against a DOS (denial of service) attack when the UI is processed on the server side (i.e. server-side rendering). Default settings are usually OK. In Angular, server-side rendering occurs to support better Search Engine Optimization (SEO), as well as to support clients which cannot use Javascript. See the next section for more details.

Sub-path in frontend URL: When using a subpath (nameSpace) in your UI server base URL (e.g. "http://localhost:4000/mysite/" instead of "http://localhost:4000/"), you must make sure that the URL **without** the subpath is added to the `rest.cors.allowed-origins` list in `[dspace]/config/modules/rest.cfg` or the `local.cfg` override. The default value used for this configuration assumes that Origin and DSpace URL are identical, but CORS origins do not contain a subpath. Without this change you will see CORS policy errors preventing communication between the frontend and backend servers.

Server Side Rendering (SSR) Settings

In DSpace 7, this section was called "universal", which referenced "Angular Universal", the original name of Angular's Server Side Rendering (SSR). As of DSpace 8, this section has been renamed "ssr", but the settings are the same.

The "ssr" section defines configurations related to Angular's Server Side Rendering (SSR). DSpace enables Angular SSR by default as required for Search Engine Optimization (SEO), as well as to support clients which cannot use Javascript. The default settings attempt to provide a balance between SEO and performance. However, you may choose to customize this behavior based on your local needs.

config.*.yml

```
ssr:
  # Whether to tell Angular to inline "critical" styles into the server-side rendered HTML.
  # Determining which styles are critical is a relatively expensive operation; this option is
  # disabled (false) by default to boost server performance at the expense of loading smoothness.
  inlineCriticalCss: false
  # Path patterns to *exclude* from SSR. By default, excludes community and collection browse, global browse,
  # global search, community list, statistics and various administrative tools.
  # The defined patterns will be run as regexes against the path of the page to check if SSR is allowed.
  # If the path matches any of the regexes it will be served directly in CSR (client side rendering).
  # NOTE: This configuration *replaces* the "paths" setting that existed in versions 7.6.3 and 8.1.
  excludePathPatterns:
    - pattern: "^/communities/[a-f0-9-]{36}/browse(/.*)?$"
      flag: "i"
    - pattern: "^/collections/[a-f0-9-]{36}/browse(/.*)?$"
      flag: "i"
    - pattern: "^/browse/"
    - pattern: "^/search$"
    - pattern: "^/community-list$"
    - pattern: "^/admin/"
    - pattern: "^/processes/?"
    - pattern: "^/notifications/"
    - pattern: "^/statistics/?"
    - pattern: "^/access-control/"
    - pattern: "^/health$"
  # Whether to enable rendering of Search component in SSR.
  # If set to true the component will be included in the HTML returned from the server side rendering.
  # If set to false the component will not be included in the HTML returned from the server side rendering.
  enableSearchComponent: false
  # Whether to enable rendering of Browse component on SSR.
  # If set to true the component will be included in the HTML returned from the server side rendering.
  # If set to false the component will not be included in the HTML returned from the server side rendering.
  enableBrowseComponent: false
  # Enable state transfer from the server-side application to the client-side application. (Defaults to true)
  # Note: When using an external application cache layer, it's recommended not to transfer the state to avoid
caching it.
  # Disabling it ensures that dynamic state information is not inadvertently cached, which can improve security
and
  # ensure that users always use the most up-to-date state.
  transferState: true
  # When a different REST base URL is used for the server-side application, the generated state contains
references to
  # REST resources with the internal URL configured. By default, these internal URLs are replaced with public
URLs.
  # Disable this setting to avoid URL replacement during SSR. In this the state is not transferred to avoid
security issues.
  replaceRestUrl: true
  # Enable request performance profiling data collection and printing the results in the server console.
  # Defaults to false. Enabling in production is NOT recommended
  enablePerformanceProfiler: false
```

The "paths" setting defines which DSpace pages will be processed in server-side rendering. For proper Search Engine Optimization, you should ensure this "paths" setting includes all URL paths that you want to be indexed by search engine bots. Any paths **not** listed in this setting will be inaccessible to most bots/crawlers (at least any that cannot process Javascript). The default "paths" settings are listed above, and they correspond to every page which is listed in the Sitemaps or may be of interest to search engines. DSpace purposefully does not add "/search" or "/browse" pages to these paths because bot requests to those pages may result in performance issues or require a larger amount of CPU to process the SSR. Keep in mind that values of "paths" are matched against the start of a path, so "/items/" will match every Item page in the system.

The "enableSearchComponent" and "enableBrowseComponent" settings control whether the Angular Components related to search/browse are enabled for SSR or not. These components are used in many different pages within the DSpace site, and these configurations are obeyed site wide. For example, while the "/communities/" and "/collections/" pages do undergo SSR by default, both of these pages also include embedded browse options as separate tabs. Setting "enableBrowseComponent: false" will ensure that the browse tabs for all Communities and Collections are **not** processed by SSR. Similarly, some "/entities/" pages (e.g. Person Entities) may include an embedded search section on the entity page. Setting "enableSearchComponent: false" will ensure that embedded search section for all Entities is **not** processed by SSR.

The "transferState" and "replaceRestUrl" settings are ONLY used if you've set the "ssrBaseUrl" setting the REST API Settings below (see next section). When the "ssrBaseUrl" is specified, then a different REST API URL is being used for all SSR requests. Setting "transferState" to true (default vaule) will ensure that the state of the server-side application will be transferred to the client-side after SSR completes. This is recommended behavior unless you are using an external cache layer. Setting "replaceRestUrl" to true (default value) will ensure that all URLs returned in the JSON response of an SSR request to the "ssrBaseUrl" will be updated to use the public REST API URL (set by the "ssl", "host", "port" and "nameSpace" in the REST API Settings). This is recommended behavior to ensure the client-side application has access to the public URLs you can use. Setting "replaceRestUrl" to false will disable this URL replacement, but will also disable "transferState" to ensure the state is not transferred while including potentially "private" URLs.

The "enablePerformanceProfiler" setting should always remain false in production environments. Setting it to "true" will enable request performance profiling data collection and print those results to the server console. This corresponds to the Angular SSR setting for performance profiling.

REST API Settings

The "rest" (REST API) section defines which REST API the UI will use. The REST settings MUST correspond to the primary URL of the backend. Usually, this means they must be kept in sync
with the value of `dspace.server.url` in the backend's `local.cfg`. See Configuration Reference.

This example is valid if your Backend is publicly available at https://mydspace.edu/server/ . Keep in mind that the "port" must always be specified even if it's a standard port (i.e. port 80 for HTTP and port 443 for HTTPS).

config.*.yml

```
rest:
  ssl: true
  host: mydspace.edu
  port: 443
  # NOTE: Space is capitalized because 'namespace' is a reserved string in TypeScript
  nameSpace: /server
  # OPTIONAL: Provide a different REST API URL to be used during SSR execution.
  # It must contain the whole URL including protocol, server port and server namespace
  ssrBaseUrl: http://localhost:8080/server
```

The "ssl", "host", "port" and "nameSpace" settings are all **required** , and are used to construct the URL used to contact the REST API. As noted above, these four settings should be kept in sync with the value of `dspace.server.url` in the backend's `local.cfg`

The "ssrBaseUrl" provides the option to use a **different** URL for the REST API when performing Server Side Rendering (SSR) (see prior section). This is not enabled by default, but it can provide potential SSR performance benefits, as it allows all SSR requests to bypass DNS lookup. For example, the primary REST API settings (ssl, host, port, nameSpace) should always reference a public URL like "https://mydspace.edu/server". But, you could set the "ssrBaseUrl" to a localhost URL (e.g. "http://localhost:8080/server") if your REST API is running on the same machine as the User Interface. This would result in all client-side code (running in the user's browser) accessing the REST API via the public URL, while the server-side code (triggered by SSR) would access the REST API via the localhost URL. A few important tips:

- The "ssrBaseUrl" need not always be a localhost URL, but it **must** be accessible to the machine where the User Interface's server-side code is running.
- When the "ssrBaseUrl" is specified, you **must also set** the corresponding `dspace.server.ssr.url` in the backend's local.cfg. See Configuration Reference.
- When the "ssrBaseUrl" is specified, there are additional "ssr" settings available including "transferState" and "replaceRestUrl". See Server Side Rendering (SSR) above.

Cache Settings - General

The "cache" section controls how long objects/responses will remain in the UI cache. The defaults should be OK for most sites.

config.*.yml

```
cache:
  # NOTE: how long should objects be cached for by default
  msToLive:
    default: 900000 # 15 minutes
  # Default 'Cache-Control' HTTP Header to set for all static content (including compiled *.js files)
  # Defaults to one week. This lets a user's browser know that it can cache these files for one week,
  # after which they will be "stale" and need to be redownloaded.
  control: max-age=604800 # one week
  # These static files should not be cached (paths relative to dist/browser, including the leading slash)
  noCacheFiles:
    - '/index.html'
  autoSync:
    defaultTime: 0
    maxBufferSize: 100
    timePerMethod:
      PATCH: 3 # time in seconds
```

Cache Settings - Server Side Rendering (SSR)

Caching options are also available for the User Interface's "server-side rendering" (which uses Angular Universal). Server-side rendering is used to pre-generate full HTML pages and pass those back to users. This is necessary for Search Engine Optimization (SEO) as some web crawlers cannot use Javascript. It also can be used to immediately show the first HTML page to users while the Javascript app loads in the user's browser.

While server-side-rendering is highly recommended on all sites, it can result in Node.js having to pre-generate many HTML pages at once when a site has a large number of simultaneous users/bots. This may cause Node.js to spend a lot of time processing server-side-rendered content, slowing down the entire site.

Therefore, DSpace provides some basic caching of server-side rendered pages, which allows the same pre-generated HTML to be sent to many users /bots at once & decreases the frequency of server-side rendering.

Two cache options are provide: `botCache` and `anonymousCache`. As the names suggest, the botCache is used for known web crawlers / bots, while the anonymousCache may be used for all anonymous (non-authenticated) users. By default, only the botCache is enabled. But highly active sites may wish to enable the anonymousCache as well, since it can provide users with a more immediate response when they encounter cached pages.

Keep in mind, when the "anonymousCache" is enabled, this means that all non-authenticated users will utilize this cache. This cache can result in massive speed improvements (for initial page load), as the majority of users may be interacting with cached content. However, these users may occasionally encounter cached pages which are outdated or "stale" (based on values of "timeToLive" and "allowStale"). This means that these users will not immediately see new updates or newly added content (Communities, Collections, Items) until the cache has refreshed itself. That said, when "timeToLive" is set to a low value (like 10 seconds), this risk is minimal for highly active pages/content.

config.*.yml

```
cache:
  ...
  serverSide:
    # Set to true to see all cache hits/misses/refreshes in your console logs. Useful for debugging SSR caching
issues.
    debug: false
    # When enabled (i.e. max > 0), known bots will be sent pages from a server side cache specific for bots.
    # (Keep in mind, bot detection cannot be guarranteed. It is possible some bots will bypass this cache.)
    botCache:
      # Maximum number of pages to cache for known bots. Set to zero (0) to disable server side caching for
bots.
      # Default is 1000, which means the 1000 most recently accessed public pages will be cached.
      # As all pages are cached in server memory, increasing this value will increase memory needs.
      # Individual cached pages are usually small (<100KB), so max=1000 should only require ~100MB of memory.
      max: 1000
      # Amount of time after which cached pages are considered stale (in ms). After becoming stale, the cached
      # copy is automatically refreshed on the next request.
      # NOTE: For the bot cache, this setting may impact how quickly search engine bots will index new content
on your site.
      # For example, setting this to one week may mean that search engine bots may not find all new content for
one week.
      timeToLive: 86400000 # 1 day
      # When set to true, after timeToLive expires, the next request will receive the *cached* page & then re-
render the page
      # behind the scenes to update the cache. This ensures users primarily interact with the cache, but may
receive stale pages (older than timeToLive).
      # When set to false, after timeToLive expires, the next request will wait on SSR to complete & receive a
fresh page (which is then saved to cache).
      # This ensures stale pages (older than timeToLive) are never returned from the cache, but some users will
wait on SSR.
      allowStale: true
    # When enabled (i.e. max > 0), all anonymous users will be sent pages from a server side cache.
    # This allows anonymous users to interact more quickly with the site, but also means they may see slightly
    # outdated content (based on timeToLive)
    anonymousCache:
      # Maximum number of pages to cache. Default is zero (0) which means anonymous user cache is disabled.
      # As all pages are cached in server memory, increasing this value will increase memory needs.
      # Individual cached pages are usually small (<100KB), so a value of max=1000 would only require ~100MB of
memory.
      max: 0
      # Amount of time after which cached pages are considered stale (in ms). After becoming stale, the cached
      # copy is automatically refreshed on the next request.
      # NOTE: For the anonymous cache, it is recommended to keep this value low to avoid anonymous users seeing
outdated content.
      timeToLive: 10000 # 10 seconds
      # When set to true, after timeToLive expires, the next request will receive the *cached* page & then re-
render the page
      # behind the scenes to update the cache. This ensures users primarily interact with the cache, but may
receive stale pages (older than timeToLive).
      # When set to false, after timeToLive expires, the next request will wait on SSR to complete & receive a
fresh page (which is then saved to cache).
      # This ensures stale pages (older than timeToLive) are never returned from the cache, but some users will
wait on SSR.
      allowStale: true
```

Authentication Settings

The "auth" section provides some basic authentication-related settings. Currently, it's primarily settings related to when a session timeout warning will be showed to your users, etc.

config.*.yml

```
auth:
  # Authentication UI settings
  ui:
    # the amount of time before the idle warning is shown
    timeUntilIdle: 900000 # 15 minutes
    # the amount of time the user has to react after the idle warning is shown before they are logged out.
    idleGracePeriod: 300000 # 5 minutes
  # Authentication REST settings
  rest:
    # If the rest token expires in less than this amount of time, it will be refreshed automatically.
    # This is independent from the idle warning. Defaults to automatic refresh when the token will
    # expire within 2 minutes. Because token expires after 30 minutes by default, this means automatic
    # refresh would occur every ~28 minutes.
    timeLeftBeforeTokenRefresh: 120000 # 2 minutes
```

Form Settings

The "form" section provides basic settings for any forms displayed in the UI. At this time, these settings only include a validatorMap, which is not necessary to modify for most sites

config.*.yml

```
form:
  # Whether to enable "spellcheck" attribute of textareas in forms.
  spellCheck: true
  # NOTE: Map server-side validators to comparative Angular form validators
  validatorMap:
    required: required
    regex: pattern
```

Notification Settings

The "notifications" section provides options related to where user notifications will appear in your UI. By default, they appear in the top right corner, and timeout after 5 seconds.

config.*.yml

```
notifications:
  rtl: false
  position:
    - top
    - right
  maxStack: 8
  # NOTE: after how many seconds notification is closed automatically. If set to zero notifications are not
closed automatically
  timeOut: 5000 # 5 second
  clickToClose: true
  # NOTE: 'fade' | 'fromTop' | 'fromRight' | 'fromBottom' | 'fromLeft' | 'rotate' | 'scale'
  animate: scale
```

The set of valid animations can be found in the NotificationAnimationsType, and are implemented in `./src/shared/animations/`

Submission Settings

The "submission" section provides some basic Submission/Deposit UI options. These allow you to optionally enable an autosave (disabled by default), and custom styles/icons for metadata fields or authority confidence values.

config.*.yml

```
submission:
  autosave:
    # NOTE: which metadata trigger an autosave
    metadata: []
    # NOTE: after how many time (milliseconds) submission is saved automatically
    # eg. timer: 300000 # 5 minutes
    timer: 0
  icons:
    metadata:
      # NOTE: example of configuration
      #   # NOTE: metadata name
      # - name: dc.author
      #   # NOTE: fontawesome (v6.x) icon classes and bootstrap utility classes can be used
      #   style: fas fa-user
      - name: dc.author
        style: fas fa-user
      # default configuration
      - name: default
        style: ''
    authority:
      confidence:
        # NOTE: example of configuration
        #   # NOTE: confidence value
        # - value: 600
        #   # NOTE: fontawesome (v6.x) icon classes and bootstrap utility classes can be used
        #   style: text-success
        #   icon: fa-circle-check
        #   # NOTE: the class configured in property style is used by default, the icon property could be used
in component
        #           configured to use a 'icon mode' display (mainly in edit-item page)
        - value: 600
          style: text-success
          icon: fa-circle-check
        - value: 500
          style: text-info
          icon: fa-gear
        - value: 400
          style: text-warning
          icon: fa-circle-question
        - value: 300
          style: text-muted
          icon: fa-thumbs-down
        - value: 200
          style: text-muted
          icon: fa-circle-exclamation
        - value: 100
          style: text-muted
          icon: fa-circle-stop
        - value: 0
          style: text-muted
          icon: fa-ban
        - value: -1
          style: text-muted
          icon: fa-circle-xmark
        # default configuration
        - value: default
          style: text-muted
          icon: fa-circle-xmark
```

Language Settings

The "fallbackLanguage" and "languages" sections allow you to customize which languages to support in your User Interface. See also Multilingual Support.

config.*.yml

```
# Fallback or default Language in which the UI will be rendered if the user's browser language is not an active
language
# NOTE: in DSpace 7.x-9.1 this was called "defaultLanguage"
fallbackLanguage: en

# Languages. DSpace Angular holds a message catalog for each of the following languages.
# When set to active, users will be able to switch to the use of this language in the user interface.
# All out of the box language packs may be found in the ./src/assets/i18n/ directory
languages:
  - code: en
    label: English
    active: true
  - code: cs
    label: eština
    active: true
  - code: de
    label: Deutsch
    active: true
  - ...
```

The DSpace UI requires that a corresponding language pack file (named with the language code and ending in ".json5") be placed in `./src/assets/i18n/`. See also DSpace 7 Translation - Internationalization (i18n) - Localization (l10n) for information about how to create and contribute these files.

Browse By Settings

The "browseBy" section provides basic UI configurations for "Browse by" pages (/browse path). The "Browse by" options that appear in the "All of DSpace" header menu *are determined dynamically from the REST API*. This allows the UI to change dynamically based on the configured browse indexes in Discovery.

config.*.yml

```
browseBy:
  # Amount of years to display using jumps of one year (current year - oneYearLimit)
  oneYearLimit: 10
  # Limit for years to display using jumps of five years (current year - fiveYearLimit)
  fiveYearLimit: 30
  # The absolute lowest year to display in the dropdown (only used when no lowest date can be found for all
items)
  defaultLowerLimit: 1900
  # If true, thumbnail images for items will be added to BOTH search and browse result lists. (default: true)
  showThumbnails: true
  # The number of entries in a paginated browse results list.
  # Rounded to the nearest size in the list of selectable sizes on the settings menu.
  pageSize: 20

# NOTE: The "types" section no longer exists, as it is determined dynamically via the REST API
```

NOTE: The "pageSize" configuration will always round to the closest "pageSizeOptions" value listed in "page-component-options.model.ts"

Community-List Settings

The "communityList" section allows you to configure the behavior of the "Communities & Collections" page (/community-list path), which is linked in the header.

config.*.yml

```
communityList:
  # Number of communities to list per expansion (i.e. each time you click "show more")
  pageSize: 20
```

NOTE: The "pageSize" configuration will always round to the closest "pageSizeOptions" value listed in "page-component-options.model.ts"

Homepage Settings

The "homePage" section allows you to configure the behavior of the DSpace homepage (/ path).

config.*.yml

```
homePage:
  recentSubmissions:
    # The number of item showing in recent submissions list. Set to "0" to hide all recent submissions
    pageSize: 5
    # Date field to use to sort recent submissions
    sortField: 'dc.date.accessioned'
  topLevelCommunityList:
    # Number of communities to list (per page) on the home page
    # This will always round to the nearest number from the list of page sizes. e.g. if you set it to 7 it'll
use 10
    pageSize: 5
  # Enable or disable the Discover filters on the homepage
  showDiscoverFilters: false
```

NOTE: The "pageSize" configuration will always round to the closest "pageSizeOptions" value listed in "page-component-options.model.ts"

Undo Settings

Both the "item" edit and "collection" edit screens allow you to undo changes within a specific time. This is controlled by these settings:

config.*.yml

```
item:
  edit:
    undoTimeout: 10000 # 10 seconds

collection:
  edit:
    undoTimeout: 10000 # 10 seconds
```

Item Access Labels

Item access labels allow to display for each item in search results if it is Open Access, under embargo, restricted or metadata only (does not contain any file/bitstream). This feature is disabled by default, but can be enabled in your config.*.yml.

config.*.yml

```
# Item Config
item:
  # Show the item access status label in items lists (default=false)
  showAccessStatuses: true
```

Item Page Settings

The "item" section allows you to configure the behavior of the Item pages.

config.*.yml

```
item:
  ...
  bitstream:
    # Number of entries in the bitstream list in the item view page.
    pageSize: 5
    # Show the bitstream access status label on the item page (default=false)
    # When enabled, embargoed bitstreams will show their embargo date in a label.
    showAccessStatuses: false
```

NOTE: The "pageSize" configuration will always round to the closest "pageSizeOptions" value listed in "page-component-options.model.ts"

Community Page Settings

The "community" section allows you to configure the behavior of the Community pages *(Path: /community/[uuid])*.

config.*.yml

```
community:
  # Search tab config
  searchSection:
    # When set to "true", the search filter sidebar will be displayed on the "Search" tab
    showSidebar: true
  # Default tab to be shown when browsing a Community. Valid values are: comcols, search, or browse_<field>
  # <field> must be any of the configured "browse by" fields, e.g., dateissued, author, title, or subject
  # When the default tab is not the 'search' tab, the search tab is moved to the last position
  defaultBrowseTab: search
```

Collection Page Settings

The "collection" section allows you to configure the behavior of the Collection pages *(Path: /collection/[uuid])*.

config.*.yml

```
collection:
  # Search tab config
  searchSection:
    # When set to "true", the search filter sidebar will be displayed on the "Search" tab
    showSidebar: true
  # Default tab to be shown when browsing a Collection. Valid values are: search, or browse_<field>
  # <field> must be any of the configured "browse by" fields, e.g., dateissued, author, title, or subject
  # When the default tab is not the 'search' tab, the search tab is moved to the last position
  defaultBrowseTab: search
```

Theme Settings

The "themes" section allows you to configure which theme(s) are enabled for your DSpace site (with the default theme being the "dspace" one). You can enable a single theme across all pages, and/or enable specific alternative themes based on a specific Community, Collection or Item (by UUID or Handle), or based on a Regex match of a URL pattern. This allows you fine grained control over how your site looks, including the ability to customize it per Community or Collection or even per specific pages in the site. *See User Interface Customization for details of how to create a new, custom theme.*

config.*.yml

```
themes:
  # Add additional themes here. In the case where multiple themes match a route, the first one
  # in this list will get priority. It is advisable to always have a theme that matches
  # every route as the last one
  #
  # # A theme with a handle property will match the community, collection or item with the given
  # # handle, and all collections and/or items within it
  # - name: 'custom',
  #   handle: '10673/1233'
  #
  # # A theme with a regex property will match the route using a regular expression. If it
  # # matches the route for a community or collection it will also apply to all collections
  # # and/or items within it
  # - name: 'custom',
  #   regex: 'collections\/e8043bc2.*'
  #
  # # A theme with a uuid property will match the community, collection or item with the given
  # # ID, and all collections and/or items within it
  # - name: 'custom',
  #   uuid: '0958c910-2037-42a9-81c7-dca80e3892b4'
  #
  # # The extends property specifies an ancestor theme (by name). Whenever a themed component is not found
  # # in the current theme, its ancestor theme(s) will be checked recursively before falling back to default.
  # - name: 'custom-A',
  #   extends: 'custom-B',
  #   # Any of the matching properties above can be used
  #   handle: '10673/34'
  #
  # - name: 'custom-B',
  #   extends: 'custom',
  #   handle: '10673/12'
  #
  # # A theme with only a name will match every route
  # name: 'custom'
  #
  # # This theme will use the default bootstrap styling for DSpace components
  # - name: BASE_THEME_NAME
  #
  - name: dspace
    # This theme's CSS will be prefetched via https://developer.mozilla.org/en-US/docs/Web/HTML/Reference
/Attributes/rel/prefetch
    # to improve peformance & style stability on initial load.
    # Can be omitted for themes that only apply to a specific handle, regex or uuid to save bandwidth.
    prefetch: true
    # Whenever this theme is active, the following tags will be injected into the <head> of the page.
    # Example use case: set the favicon based on the active theme.
    headTags:
    - tagName: link
      attributes:
        rel: icon
        href: assets/dspace/images/favicons/favicon.ico
        sizes: any
    - tagName: link
      attributes:
        rel: icon
        href: assets/dspace/images/favicons/favicon.svg
        type: image/svg+xml
    - tagName: link
      attributes:
        rel: apple-touch-icon
        href: assets/dspace/images/favicons/apple-touch-icon.png
    - tagName: link
      attributes:
        rel: manifest
        href: assets/dspace/images/favicons/manifest.webmanifest
```

Bundle Settings

The "bundle" section allows you to customize which bundles will be displayed as suggestions whenever you upload a new Bitstream:

```
bundle:
  standardBundles: [ ORIGINAL, THUMBNAIL, LICENSE ]
```

Media Viewer Settings

The DSpace UI comes with a basic, out-of-the-box Media Viewer (disabled by default). This media viewer can support any files which have a MIME Type that *begins with* either "image/*", "video/*", or "audio/*".

config.*.yml

```
# Whether to enable media viewer for image and/or video Bitstreams (i.e. Bitstreams whose MIME type starts with
'image' or 'video').
# When "image: true", this enables a gallery viewer where you can zoom or page through images.
# When "video: true", this enables embedded video streaming.  This embedded video streamer also supports audio
files.
mediaViewer:
  image: false
  video: false
```

Uploading video captioning files

The Video viewer also supports WebVTT (or VTT) Captioning. Video captioning requires that a WebVTT Caption file (.vtt) be uploaded into the DSpace Item (DSpace is not able to create or generate these .vtt files). Here's an example of how to setup captioning:

- The Item must already have a Bitstream which is a video file (in a "video/*" format) in the ORIGINAL bundle. In this example, we'll assume it is named "myVideo.mp4"
- Upload a corresponding WebVTT Caption file named "`[video-filename]-[languageCode].vtt`" to the ORIGINAL bundle.
  - For a video named "myVideo.mp4", an English caption file would be named "myVideo.mp4-en.vtt".
  - If an additional Spanish language caption file was uploaded, it should be named "myVideo.mp4-es.vtt".
  - All WebVTT Caption files MUST use two-letter ISO 639-1 Language Codes. A list of all supported Language Codes can be found in "src/app/item-page/media-viewer/media-viewer-video/language-helper.ts"
- Once the Caption file is uploaded, reload the video viewer (on the Item page). You should now see the "Captions" (or CC) option is now available. (Depending on the browser you use, this option may appear in the lower menu of the video, or require you to open an options menu.) Selecting it will enable captioning in your language of choice.

Toggle end-user agreement and privacy policy

The DSpace UI comes with basic end-user agreement and privacy policy functionality. These features can be disabled in a configuration file. More information on what disabling on of these features results in is documented in the default app configuration (see code snippet below).

config.*.yml

```
info:
  # Whether the end user agreement is required before users may use the repository.
  # If enabled, the user will be required to accept the agreement before they can use the repository.
  # If disabled, the page will not exist and no agreement is required to use the repository
  enableEndUserAgreement: false
  # Whether the privacy statement should exist or not.
  enablePrivacyStatement: false
```

By default the features are enabled.

Settings for rendering Markdown, HTML and MathJax in metadata

The DSpace UI can support Markdown (using https://commonmark.org/) and MathJax (https://www.mathjax.org) in metadata field values. Both Markdown and MathJax are disabled by default.

HTML is a part of markdown, so enabling the markdown option will ensure HTML tags in metadata field values get rendered as well

config.*.yml

```
# Whether to enable Markdown (https://commonmark.org/) and MathJax (https://www.mathjax.org/)
# display in supported metadata fields. By default, only dc.description.abstract is supported.
markdown:
  enabled: false
  mathjax: false
```

Mathjax will only be rendered if markdown is enabled, so configuring 'markdown.mathjax = true' with 'markdown.enabled = false' will have no effect.

By default, only the "dc.description.abstract" metadata supports these formats when enabled. To enable markdown for other metadata fields, a custom sub-component of the ItemPageFieldComponent has to be created for that metadata field, with the enableMarkdown field set to true. Refer to the ItemPageAbstractFieldComponent component for an example.

Controlled Vocabularies in Search Filters

When using hierarchical controlled vocabularies (e.g. SRSC as described in Authority Control of Metadata Values), it's possible to search using the controlled vocabulary hierarchy via the search filters. To enable this feature, you must specify the filter and vocabulary to enable as follows:

config.*.yml

```
# Which vocabularies should be used for which search filters
# and whether to show the filter in the search sidebar
# Take a look at the filter-vocabulary-config.ts file for documentation on how the options are obtained
vocabularies:
  - filter: 'subject'
    vocabulary: 'srsc'
    enabled: true
```

Keep in mind, the "filter" MUST be a valid search filter (e.g. subject, author) as seen on the "/api/discover/facets" REST API endpoint. The "vocabulary" MUST be a valid controlled vocabulary installed in your DSpace backend (under "[dspace]/config/controlled-vocab/" folder based on the documentation at Authority Control of Metadata Values.

When this feature is enabled, you should see a "Browse [filter] tree" link in the search filter on the search results page (and anywhere search filters are shown). This "Browse [filter] tree" link will allow you to select a search filter from within the configured hierarchical vocabulary.

Search settings

The "search" section allows you to customize how the Search page works *(Path: /search)*

config.*.yml

```
search:
  # Settings to enable/disable or configure Advanced Search filters.
  advancedFilters:
    enabled: false
    # List of filters to enable in "Advanced Search" dropdown
    filter: [ 'title', 'author', 'subject', 'entityType' ]
```

Layout settings

Configuration for layout customization of metadata rendering in Item page. Currently only the authority reference config is available, more will follow with the integration of the so called CRIS layout.

```
layout:
  # Configuration of icons and styles to be used for each authority controlled link
  authorityRef:
    - entityType: DEFAULT
      entityStyle:
        default:
          icon: fa fa-user
          style: text-info
    - entityType: PERSON
      entityStyle:
        person:
          icon: fa fa-user
          style: text-success
        default:
          icon: fa fa-user
          style: text-info
    - entityType: ORGUNIT
      entityStyle:
        default:
          icon: fa fa-university
          style: text-success
    - entityType: PROJECT
      entityStyle:
        default:
          icon: fas fa-project-diagram
          style: text-success
  # If true the download link in item page will be rendered as an advanced attachment, the view can be then
configured with the layout.advancedAttachmentRendering config
  showDownloadLinkAsAttachment: false
  # Configuration for advanced attachment rendering in item pages. This controls how files are displayed when
showDownloadLinkAsAttachment is enabled.
  # Defines which metadata/attributes to display for bitstream attachments.
  advancedAttachmentRendering:
    # Metadata and attributes to display for each attachment
    metadata:
      - name: dc.title
        type: metadata
        truncatable: false
      - name: dc.type
        type: metadata
        truncatable: false
      - name: dc.description
        type: metadata
        truncatable: true
      - name: size
        type: attribute
      - name: format
        type: attribute
      - name: checksum
        type: attribute
```

Geospatial map viewer settings

The "geospatialMapViewer" section allows you to enable the display of geospatial maps on item pages, search result views, and as a "browse by" page. These maps are driven by Leaflet.js with the OpenStreetMaps Mapnik tile provider (by default).

Take note of the default field used for geospatial metadata - this may need to be changed to fit your specific repository metadata usage.

config.*.yml

```
# Geospatial Map display options
geospatialMapViewer:
  # Which fields to use for parsing as geospatial points in search maps
  # (note, the item page field component allows any field(s) to be used
  # and is set as an input when declaring the component)
  spatialMetadataFields:
    - 'dcterms.spatial'
  # Which discovery configuration to use for 'geospatial search', used
  # in the browse map
  spatialFacetDiscoveryConfiguration: 'geospatial'
  # Which filter / facet name to use for faceted geospatial search
  # used in the browse map
  spatialPointFilterName: 'point'
  # Whether item page geospatial metadata should be displayed
  # (assumes they are wrapped in a test for this config in the template as
  # per the default templates supplied with DSpace for untyped-item and publication)
  enableItemPageFields: false
  # Whether the browse map should be enabled and included in the browse menu
  enableBrowseMap: false
  # Whether a 'map view' mode should be included alongside list and grid views
  # in search result pages
  enableSearchViewMode: false
  # The tile provider(s) to use for the map tiles drawn in the leaflet maps.
  # (see https://leaflet-extras.github.io/leaflet-providers/preview/) for a full list
  tileProviders:
    - 'OpenStreetMap.Mapnik'
  # Starting centre point for the map, as lat and lng coordinates. This is useful
  # to set the centre of the map when the map is first loaded and if there are no
  # points, shapes or markers to display.
  # Defaults to the centre of Istanbul
  defaultCentrePoint:
    lat: 41.015137
    lng: 28.979530
```

Additional information about this feature can be found at https://github.com/kshepherd/dspace-geospatial-maps-doc/blob/main/README.md

Debug Settings

The "debug" property allows you to turn on debugging in the Angular UI. When enabled, your environment and all Redux actions/transfers are logged to the console. This is only ever needed if you are debugging a tricky issue.

config.*.yml

```
# NOTE: will log all redux actions and transfers in console
debug: false
```

User Interface Customization

- Angular Overview
- Theme Technologies
- Running the UI in Developer Mode
- Creating a Custom Theme
    - Theme Directories & Design Principles
    - Getting Started
    - Global style/font/color customizations
    - Customize Logo in Header
    - Customize Navigation Links in Header
    - Customize Footer
    - Customize Favicon for site or theme
    - Customize Home Page News
    - Customize the simple Item page
    - Customize other Components in your Theme
    - Customize UI labels using Internationalization (i18n) files
    - Extending other Themes
    - Adding Component Directories to your Theme
    - Removing Component Directories from your Theme
    - Adding routes to your theme's new pages
    - Debugging which theme is being used
    - Finding which component is generating the content on a page
- Upgrading an Existing Theme
    - Upgrading from 9.x to 10.x
- Additional Theming Resources

Angular Overview

The DSpace User Interface (UI) is built on the Angular framework. All data comes from the REST API (DSpace Backend), and the final HTML pages are generated via TypeScript.

Before getting started in customizing or branding the UI, there are some basic Angular concepts to be aware of. Y*ou do not need to know Angular or TypeScript to theme or brand the UI.* But, understanding a few basic concepts will allow you to better understand the folder structure / layout of the codebase.

Angular Components: In Angular, every webpage consists of a number of "components" which define the structure of a page. They are the main "building block" of any Angular application. Components are reusable across many pages. So, for example, there's only one "header" and "footer" component, even though they appear across all pages.

Each Component has:

- A `*.component.ts` (TypeScript) file which contains the logic & name ("selector") of the component
- A `*.component.html` (HTML) file which contains the HTML markup for the component (and possibly references to other embedded components). This is also called the "template".
    - In HTML files, components are named/referenced as HTML-like tags (e.g. `<ds-header>`, `<ds-footer>`). In DSpace's UI, every component starts with "ds-" in order to distinguish it as an out-of-the-box DSpace component.
- A `*.component.scss` (Sass / CSS) file which contains the style for the component.

If you want a deeper dive into Angular concepts of Components and Templates, see https://angular.io/guide/architecture-components

Theme Technologies

The DSpace UI uses:

- Bootstrap (v5.x) website framework for general layout & webpage components (buttons, alerts, etc)
- Sass, a CSS preprocessor, for stylesheets. Sass is very similar to CSS (an in fact, any CSS is valid Sass). But, Sass allows you to nest CSS rules & have variables and functions. For a brief overview on Sass, see https://sass-lang.com/guide
- HTML5, the latest specification of the HTML language

Familiarity with these technologies (or even just CSS + HTML) is all you need to do basic theming of the DSpace UI.

Running the UI in Developer Mode

Whenever you are testing changes in the User Interface, may wish to see you changes "live" instead of rebuilding after each change. The easiest way to achieve this is to run the User Interface locally (i.e. on localhost) in developer mode by running:

```
npm run start:dev
```

This mode has several development-specific advantages:

- UI starts more rapidly
- UI will use a separate "config.dev.yml" configuration file. This lets you have development specific configs, separate from your Production settings in "config.prod.yml"
- UI will automatically reload anytime you modify a file. Essentially the UI will constantly "watch" for changes (as you make them) & will reload anytime you modify a file. This lets you find issues/bugs more rapidly and also test more rapidly.

Keep in mind, you should NEVER run the UI in developer mode in production scenarios. Production mode provides much better performance and ensures your site fully supports SEO, etc.

Creating a Custom Theme

Theme Directories & Design Principles

A theme's directory should include the following files and directories

- `app/` contains the theme's Angular components and should mirror the structure of `src/app/`
- `assets/` contains the theme's custom assets, such as fonts or images
- `styles/` contains the theme's global styles
- `eager-theme-components.ts` references the components that should be available immediately when first loading, such as the main parts of the homepage and components that are present on every page.
- `lazy-listable-components.ts` components that use the `@listableObjectComponent decorator`. This will ensure that those decorators get picked up when the app loads.

Out of the box, there are three theming layers/directories to be aware of:

- **Base Theme** (`src/app/` directories): The primary look & feel of DSpace (e.g. HTML layout, header/footer, etc) is defined by the HTML5 templates under this directory. Each HTML5 template is stored in a subdirectory named for the Angular component where that template is used. The base theme includes very limited styling (CSS, etc), based heavily on *default Bootstrap (5.x) styling*, and only allowing for minor tweaks to improve WCAG 2.1 AA accessibility.
- **Custom Theme** (`src/themes/custom` directories): This directory acts as the scaffolding or template for creating a new custom theme. It provides (empty) Angular components/templates which allow you to change the theme of individual components. Since all files are empty by default, if you enable this theme (without modifying it), it will look *identical* to the Base Theme.
- **DSpace Theme** (`src/themes/dspace` directories): This is the default theme for DSpace. It's a very simple example theme providing a custom color scheme, header & homepage on top of the Base Theme. It's important to note that this theme ONLY provides custom CSS/images to override our Base Theme. All HTML5 templates are included at the Base Theme level, as this ensures those HTML5 templates are also available to the Custom Theme.

The DSpace UI design principles & technologies are described in more detail at DSpace UI Design principles and guidelines

Getting Started

1. *Choose a theme to start from:* As documented above, there are two "src/theme/" directories provided out of the box: "custom" or "dspace". You should select one to use as the basis for your theme. Which you choose is up to you, but here are a few things to consider:
    a. *DSpace Theme* (`src/themes/dspace`): This is a simple, example theme for novice users. Primarily, in this theme, you can immediately customize the CSS, header & homepage components. You can add other components as needed (see "Adding Component Directories to your Theme" below).
        i. Advantages: This theme is small and simple. It provides an easy starting point / example for basic themes. Future User Interface upgrades (e.g. from 7.1 7.2) are likely to be easier because the theme is smaller in size.
        ii. Disadvantages: It has very few component directories by default. But you can always add more. See "Adding Component Directories to your Theme" below.
    b. *Custom Theme* (`src/themes/custom`): This theme provides all available theme-able components for more advanced or complex theming options. This provides you full control over everything that is theme-able in the User Interface
        i. Advantages: All theme-able components are provided in subdirectories. This makes it easier to modify the look and feel of any area of the User Interface.
        ii. Disadvantages: After creating your theme, you may wish to remove any component directories that you didn't modify (see "Removing Component Directories from your Theme" below). Generally speaking, upgrades (e.g. from 7.1 7.2) are often easier if your theme includes fewer components (as your theme may require updates if any component it references change significantly).
2. *Create your own theme folder OR edit the existing theme folder:* Either edit the theme directory in place, or copy it (and all its contents) into a new folder under `src/themes/` (choose whatever folder name you want)
3. Import the new theme's `eager-theme-components.ts` in `themes/eager-themes-components.ts`. If you're switching from one theme to another, remove the old theme from the imports. Below is an example for a theme named "my-theme":

eager-themes-components.ts

```
// COMMENT out the imports for any themes you are NOT using
// import { COMPONENTS as CUSTOM_THEME_EAGER_COMPONENTS } from './custom/eager-theme-components';
// import { COMPONENTS as DSPACE_THEME_EAGER_COMPONENTS } from './dspace/eager-theme-components';

// Add a new import for your custom theme. Give its COMPONENTS variable a unique name (e.g. "as [choose-
a-unique-name]").
// Make sure the path points at its "eager-theme-components" (see 'from' portion of the import
statement).
// NOTE: You can import multiple themes if you plan to use multiple themes
import { COMPONENTS as MY_THEME_THEME_EAGER_COMPONENTS } from './my-theme/eager-theme-components';

...
export const EAGER_THEME_COMPONENTS = [
  // Again, comment out any themes you are NOT using
  // ...CUSTOM_THEME_EAGER_COMPONENTS,
  // ...DSPACE_THEME_EAGER_COMPONENTS,

  // Add your custom theme's COMPONENTS to this list
  // NOTE: you can add multiple theme's to this list if you plan to use multiple themes.
  MY_THEME_THEME_EAGER_COMPONENTS,
];
```

4. Import the new theme's `lazy-listable-components.ts` in `themes/themes-listable-components.ts`. If you're switching from one theme to another, remove the old theme from the imports. Below is an example for a theme named "my-theme":

themes-listable-components.ts

```
// COMMENT out the imports for any themes you are NOT using
// import { LISTABLE_COMPONENTS as CUSTOM_LISTABLE_COMPONENTS } from './custom/lazy-listable-components';
// import { LISTABLE_COMPONENTS as DSPACE_LISTABLE_COMPONENTS } from './dspace/lazy-listable-components';

// Add a new import for your custom theme. Give its LISTABLE_COMPONENTS variable a unique name (e.g. "as
[choose-a-unique-name]").
// Make sure the path points at its "lazy-listable-components" (see 'from' portion of the import
statement).
// NOTE: You can import multiple themes if you plan to use multiple themes
import { LISTABLE_COMPONENTS as MY_THEME_LISTABLE_COMPONENTS } from './my-theme/lazy-listable-
components';

...
export const THEME_LISTABLE_COMPONENTS = [
  // Again, comment out any themes you are NOT using
  // ...CUSTOM_LISTABLE_COMPONENTS,
  // ...DSPACE_LISTABLE_COMPONENTS,

  // Add your custom theme's LISTABLE_COMPONENTS to this list
  // NOTE: you can add multiple theme's to this list if you plan to use multiple themes.
  MY_THEME_LISTABLE_COMPONENTS,
];
```

5. *Register your theme folder (only necessary if you create a new folder in previous step):* Now, we need to make the UI aware of this new theme folder, before it can be used in configuration.
    a. Modify `angular.json` (in the root folder), adding your theme folder's main "theme.scss" file to the "styles" list. The below example is for a new theme folder named `src/themes/mydspacesite/`

```
"styles": [
  "src/styles/startup.scss",
  {
     "input": "src/styles/base-theme.scss",
     "inject": false,
     "bundleName": "base-theme"
  },
  ...
  {
     "input": "src/themes/mydspacesite/styles/theme.scss",
     "inject": false,
     "bundleName": "mydspacesite-theme"
  },
]
```

NOTE: the "bundleName" for your custom them MUST use the format "${folder-name}-theme". E.g. if the folder is named "src/themes/amazingtheme", then the "bundleName" MUST be "amazingtheme-theme"

6. *Enable your theme:* Modify your `config/config.*.yml` configuration file, adding your new theme to the "themes" array in that file. Pay close attention to modify the correct configuration file (e.g. modify config.dev.yml if running in dev mode, or config.prod.yml if running in prod mode). We recommend starting in "dev mode" (config.dev.yml) as this mode lets you see your changes immediately in a browser without a full rebuild of the UI – see next step.

config.*.yml

```
# In this example, we only show one theme enabled. It's possible to enable multiple (see below note)
themes:
  - name: 'mydspacesite'
```

NOTE: The "name" used is the name of the theme's folder, so the example is for enabling a theme at `src/themes/mydspacesite/` globally. You should also comment out the default "dspace" theme, if you intend to replace it entirely.
NOTE #2: You may also choose to enable multiple themes for your site, and even specify a different theme for different Communities, Collections, Items or URL paths. See User Interface Configuration for more details on "Theme Settings

7. *Verify your settings by starting the UI* (ideally in Dev mode): At this point, you should verify the basic settings you've made all "work". We recommend doing your theme work while running the UI in "dev mode", as the UI will auto-restart anytime you save a new change. This will allow you to quickly see the impact of each change in your browser.

```
# Start in dev mode (which uses config.dev.yml)
npm run start:dev
```

8. At this point, you can start making changes to your theme. See the following sections for examples of how to make common changes.

Global style/font/color customizations

Changes to the global Bootstrap variables or styles will apply to all pages / Angular components across the entire site.

1. *Global style changes:* All global style changes can be made in your theme's `styles` folder (e.g. `src/themes/mydspacesite/styles`). There are four main files in that folder:
   a. `_theme_sass_variable_overrides.scss` - May be used to override Bootstrap's default Sass variables. This is the file you may wish to use for **most** style changes. There are a large number of Bootstrap variables available which control everything from fonts, colors and the base style for all Bootstrap web components. For a full list of Bootstrap variables you can override in this file, see the `node_modules/bootstrap/scss/_variables.scss` file (which is installed in your source directory when you run "npm install"). More information may also be found in the Bootstrap Sass documentation at https://getbootstrap.com/docs/5.3/customize/sass/
   b. `_theme_css_variable_overrides.scss` - May be used to override DSpace's default CSS variables. DSpace's UI uses CSS variables for all its components. These variables all start with "--ds-*", and are listed in `src/styles/_custom_variables.scss`. You can also use this file to add your own, custom CSS variables which you want to use for your theme. If you create custom variables, avoid naming them with a "--ds-*" or a "--bs-*" prefix, as those are reserved for DSpace and Bootstrap variables.
   c. `_global-styles.scss` - May be used to modify the global CSS/SCSS for the site. This file may be used to override the default global style contained in `src/styles/_global-styles.scss`. Keep in mind, many styles can be more quickly changed by simply updating a variable in one of the above "*_variable_overrides.scss" files. So, it's often easier to use those first, where possible.
   d. `theme.scss` - This just imports all the necessary Sass files to create the theme. It's unnecessary to modify directly, unless you with to add new Sass files to your theme.
2. *Modifying the default font:* By default, DSpace uses Bootstrap's "native font stack", which just uses system UI fonts. However, the font used in your site can be quickly updated via Bootstrap variables in your theme's `_theme_sass_variable_overrides.scss` file.
   a. One option is to add a new import statement and modify the "$font-family-sans-serif" variable:

```
// Import the font (from a URL)
@import url('https://fonts.googleapis.com/css?family=Source+Sans+Pro');

// Configure Bootstrap to use this font (and list a number of backup fonts to use on various
systems)
$font-family-sans-serif: 'Source Sans Pro', -apple-system, BlinkMacSystemFont, "Segoe UI",
"Roboto", "Helvetica Neue", Arial, sans-serif, "Apple Color Emoji", "Segoe UI Emoji", "Segoe UI
Symbol" !default;
```

b. If your font requires installing local files, you can do the following
   i. Copy your font file(s) in your theme's `assets/fonts/` folder
   ii. Create a new ".scss" file specific to your font in that folder, e.g. `assets/fonts/open-sans.scss`, and use the "@font-face" CSS rule to load that font:

open-sans.scss

```
@font-face {
  font-family: "Open Sans";
  src: url("/assets/fonts/OpenSans-Regular-webfont.woff2") format("woff2"),
       url("/assets/fonts/OpenSans-Regular-webfont.woff") format("woff");
}
```

   iii. Then, import that new "open-sans.scss" file and use it in the "$font-family-sans-serif" variable

```
// Import the font via the custom SCSS file
@import '../assets/fonts/open-sans';

// Configure Bootstrap to use this font (and list a number of backup fonts to use on
various systems)
$font-family-sans-serif: 'Open Sans', -apple-system, BlinkMacSystemFont, "Segoe UI",
"Roboto", "Helvetica Neue", Arial, sans-serif, "Apple Color Emoji", "Segoe UI Emoji",
"Segoe UI Symbol" !default;
```

   c. Keep in mind, as changing the font just requires adjusting Bootstrap Sass variables, there are a lot of Bootstrap guides out there that can help you make more advanced changes

3. *Modifying default color scheme:* The colors used in your site can be quickly updated via Bootstrap variables in your theme's `_theme_sass_variable_overrides.scss` file.
   a. Again, you can use entirely Bootstrap variables to adjust color schemes. See the Bootstrap Theming Colors documentation
   b. A list of all Bootstrap color variables can be found in the `node_modules/bootstrap/scss/_variables.scss` file
   c. Additional examples can be found in the out-of-the-box "dspace" theme, which adjusts the default Bootstrap colors slightly for both accessibility & to match the DSpace logo.
4. Any changes require rebuilding your UI. If you are running in "dev mode" (npm run start:dev), then the UI will restart automatically whenever changes are detected.

CSS custom properties vs. SASS variables

⚠

Support for theme switching at runtime requires that components use CSS custom properties (which vary at runtime) rather than SASS variables (which are fixed at build time). Thus, SASS variables will be undefined in individual components' stylesheets. The Bootstrap SASS variables are mapped to CSS properties for use in these places. For example, `$red` is mapped to `--bs-red` and may be referenced as `var("--bs-red")`.

Customize Logo in Header

1. Copy your logo to your theme's `assets/images/` folder. Anything in this theme folder will be deployed to `/assets/[theme-name]/images/` URL path.
2. Edit your theme's `app/header/header.component.ts` file. Swap the "templateUrl" property that your theme is using the local copy of "header.component.html"

header.component.ts

```
@Component({
  selector: 'ds-header',
  // styleUrls: ['header.component.scss'],
  styleUrls: ['../../../../app/header/header.component.scss'],
  // Uncomment the templateUrl which references the "header.component.html" file in your theme directory
  templateUrl: 'header.component.html',
  // Comment out the templateUrl which references the default "src/app/header/header.component.html"
file.
  //templateUrl: '../../../../app/header/header.component.html',
})
```

3. Your theme's version of the `header.component.html` file will be empty by default. Copy over the default HTML code from `src/app/header/header.component.html` into your version of this file.
4. Then, modify your copy of header.component.html to use your logo. In this example, we're assuming your theme name is "mytheme" and the logo file is named "my-logo.svg"

```
<header>
  <div class="container">
    <div class="d-flex flex-row justify-content-between">
      <a class="navbar-brand my-2" routerLink="/home">
        <!-- Modify the logo on the next line -->
        <img src="/assets/mytheme/images/my-logo.svg" [attr.alt]="'menu.header.image.logo' | translate"/>
      </a>
      ...
</header>
```

5. Obviously, you can also make additional modifications to the HTML of the header in this file! You'll also see that the header references several other DSpace UI components (e.g. `<ds-search-navbar>` is the search icon in the header). You can easily comment out these tags to disable them, or move them around to change where that component appears in the header.
6. Any changes require rebuilding your UI. If you are running in "dev mode" (npm run start:dev), then the UI will restart automatically whenever changes are detected.
7. NOTE: If you have a theme based on the "dspace" theme, be aware that theme places the header logo in two locations. This allows the "dspace" theme to support a single-line header (whereas the "custom" theme's header is multi-line):
    a. The Header component (*as described above*) is only used on user profile pages
    b. The Navbar component (`src/app/navbar/navbar.component.html`) is used everywhere else. The Navbar component can be customized in the same way as the Header Component. Just edit the logo path in the "navbar.component.html".

Customize Navigation Links in Header

This provides a basic example of adding in a hardcoded link to the header menu (displayed on every page in DSpace).

1. Edit your theme's existing `app/navbar/navbar.component.html` file. This file defines the entire <nav> which displays the navigation header across the entire DSpace site. While much of the content in this <nav> is loaded dynamically via other Angular components, it is possible to easily add a hardcoded link to the existing header. Find the section of this <nav> which is the , as that's where you'll want to add your new link. See inline comments in the example below.

navbar.component.html

```
<!-- The header links are all embedded under the role="menubar" div tag -->
<div role="menubar" class="navbar-nav" ... >
    ...
    <!-- Add your new link at the end (or beginning) of this DIV tag in a new DIV tag -->
    <!-- NOTE: All classes used below are the same Bootstrap CSS classes used by our 'dspace' and
'custom' themes.
         You can modify them if the link doesn't look correct in your theme. -->
    <div class="ds-menu-item-wrapper text-md-center">
      <a role="menuitem" href="http://dspace.org" class="ds-menu-item">DSpace.org</a>
    </div>
</div>
```

2. Obviously, you can also make additional modifications to the HTML of the header in this file, as necessary for your navigation header. Keep in mind though that anything you remove may impact the dynamic content that is pulled into this navigation header.
    a. An example is that the header logo for the "dspace" theme also exists in this same file.

3. Any changes require rebuilding your UI. If you are running in "dev mode" (npm run start:dev), then the UI will restart automatically whenever changes are detected.

Customize Footer

1. First, you'll want to decide if you want to modify just the footer's HTML, or the footer's styles (CSS/Sass), or both.
    a. If you want to modify the HTML, you'll need to create a copy of "footer.component.html" in your theme, where you place your changes.
    b. If you want to modify the styles, you'll need to create a copy of "footer.component.scss" in your theme, where you place your changes.
2. Edit your theme's `app/footer/footer.component.ts` file. Swap the "templateUrl" and "styleUrls" properties, based on which you want to modify in your theme.

footer.component.ts

```
@Component({
  selector: 'ds-footer',
  // If you want to modify styles, then...
  // Uncomment the styleUrls which references the "footer.component.scss" file in your theme's directory
  // and comment out the one that references the default "src/app/footer/footer.component.scss"
  styleUrls: ['footer.component.scss'],
  //styleUrls: ['../../../../app/footer/footer.component.scss'],
  // If you want to modify HTML, then...
  // Uncomment the templateUrl which references the "footer.component.html" file in your theme's
directory
  // and comment out the one that references the default "src/app/footer/footer.component.html"
  templateUrl: 'footer.component.html'
  //templateUrl: '../../../../app/footer/footer.component.html'
})
```

3. Now, based on what you want to modify, you will need to either update your theme's copy of `footer.component.html` or `footer.component.scss` or both.
    a. To change footer HTML: Your theme's version of the `footer.component.html` file will be empty by default. Copy over the default HTML code from `src/app/footer/footer.component.html` into your version of this file.
    b. To change footer Styles: Your theme's version of the `footer.component.scss` file will be empty by default. Copy over the default Sass code from `src/app/footer/footer.component.scss` into your version of this file.
4. Modify the HTML or Sass as you see fit.
    a. If you want to add images, add them to your theme's `assets/images` folder. Then reference them at the `/assets/[theme-name]/images/` URL path.
    b. Keep in mind, all Bootstrap variables, utility classes & styles can be used in these files. Take advantage of Bootstrap when you can do so.
5. DSpace also has a option to display a two-level footer, which is becoming more common these days. By default. DSpace just displays a small, bottom footer. But, you can enable a top footer (above that default footer) by add this line into your theme's `footer.component.ts`

footer.component.ts

```
export class FooterComponent extends BaseComponent {
   // This line will enable the top footer in your theme
   showTopFooter = true;
}
```

This top footer appears in the `footer.component.html` within a div. Notice the "`@if(showTopFooter)`", which only shows that div when that variable is set to "true"

footer.component.html

```
<footer class="text-lg-start">
  <!-- This div and everything within it only displays if showTopFooter=true -->
  @if (showTopFooter) {
      <div class="top-footer">
        ...
      </div>
  }
  <!-- The bottom footer always displays -->
  <div class="bottom-footer ...">
    ...
  </div>
</footer>
```

6. Any changes require rebuilding your UI. If you are running in "dev mode" (npm run start:dev), then the UI will restart automatically whenever changes are detected.

Customize Favicon for site or theme

Each theme has the ability to add a set of (attribute-only) HTML tags in the `<head>` section of the page. This is useful for example to change the **favicon** based on the active theme. Whenever the theme changes, the head tags are reset. A theme can inherit head tags from the parent theme **only** if it doesn't have any head tags itself. (E.g. theme B extends theme A; if theme B does not have head tags, the head tags of theme A will be used (if any). However, if theme B does have head tags, only the tags from theme B will be used.) If none of the themes in the inheritance hierarchy have head tags configured, the head tags of the default theme (if any) will be used.

Note that a simple hardcoded favicon is set in case no head tags are currently active. The hardcoded favicon is stored at `src/assets/images/favicon.ico`. This implies that if head tags are added to a theme, the favicon should also be configured explicitly for that theme, else the behavior is undefined.

1. In the "themes" section of your `config/config.*.yml` configuration file, add (one or more) "headTags", pointing at the favicon file you want to use. For example:

```
themes:
  # The default dspace theme
  - name: dspace
    # Whenever this theme is active, the following tags will be injected into the <head> of the page.
    # Example use case: set the favicon based on the active theme.
    headTags:
      # Insert <link rel="icon" href="assets/dspace/images/favicons/favicon.ico" sizes="any"/> into the
<head> of the page.
      - tagName: link
        attributes:
          rel: icon
          href: assets/dspace/images/favicons/favicon.ico
          sizes: any
      # Insert <link rel="icon" href="assets/dspace/images/favicons/favicon.svg" type="image/svg+xml"/>
into the <head> of the page.
      - tagName: link
        attributes:
          rel: icon
          href: assets/dspace/images/favicons/favicon.svg
          type: image/svg+xml
      # Insert <link rel="apple-touch-icon" href="assets/dspace/images/favicons/apple-touch-icon.png"/>
into the <head> of the page.
      - tagName: link
        attributes:
          rel: apple-touch-icon
          href: assets/dspace/images/favicons/apple-touch-icon.png
      # Insert <link rel="manifest" href="assets/dspace/images/favicons/manifest.webmanifest"/> into the
<head> of the page.
      - tagName: link
        attributes:
          rel: manifest
          href: assets/dspace/images/favicons/manifest.webmanifest
```

2. Any changes to this configuration just require restarting your site (no rebuild necessary).
3. NOTE: If you specify multiple formats for your favicon (e.g. favicon.svg and favicon.ico), then your browser will select which one it prefers (e.g. Chrome seems to favor SVG over ICO). However, if you want to force all browser to use a single favicon, then you may wish to only specify one "icon" format in your `headTags` section.

Customize Home Page News

The primary Home page content is all included in the source code under "src/app/home-page". The News section is specifically under "src/app/home-page/home-news".

1. First, you'll want to decide if you want to modify just the Home Page News HTML, or styles (CSS/Sass), or both.
    a. If you want to modify the HTML, you'll need to create a copy of the HTML in "app/home-page/home-news/home-news.component.html" in your theme. This is where you place your changes.
    b. If you want to modify the styles, you'll need to create a copy of the CSS in "app/home-page/home-news/home-news.component.scss" in your theme. This is where you place your changes.
2. Edit your theme's `app/home-page/home-news/home-news.component.ts` file. Swap the "templateUrl" and "styleUrls" properties, based on which you want to modify in your theme.

home-news.component.ts

```
@Component({
  selector: 'ds-home-news',
  // If you want to modify styles, then...
  // Uncomment the styleUrls which references the "home-news.component.scss" file in your theme's
directory
  // and comment out the one that references the default "src/app/home-page/home-news/home-news.
component.scss"
  styleUrls: ['./home-news.component.scss'],
  //styleUrls: ['../../../../../app/home-page/home-news/home-news.component.scss'],
  // If you want to modify HTML, then...
  // Uncomment the templateUrl which references the "home-news.component.html" file in your theme's
directory
  // and comment out the one that references the default "src/app/home-page/home-news/home-news.
component.html"
  templateUrl: './home-news.component.html'
  //templateUrl: '../../../../../app/home-page/home-news/home-news.component.html'
})
```

3. Now, based on what you want to modify, you will need to either update your theme's copy of `home-news.component.html` or `home-news.component.scss` or both.
    a. To change HTML: Your theme's version of the `home-news.component.html` file will be empty by default. Copy over the default HTML code from `src/app/home-page/home-news/home-news.component.html` into your version of this file.
    b. To change Styles: Your theme's version of the `home-news.component.scss` file will be empty by default. Copy over the default Sass code from `src/app/home-page/home-news/home-news.component.scss` into your version of this file.
4. Modify the HTML or Sass as you see fit.
    a. If you want to add images, add them to your theme's `assets/images` folder. Then reference them at the `/assets/[theme-name]/images/` URL path.
    b. Keep in mind, all Bootstrap variables, utility classes & styles can be used in these files. Take advantage of Bootstrap when you can do so.
5. Any changes require rebuilding your UI. If you are running in "dev mode" (npm run start:dev), then the UI will restart automatically whenever changes are detected.

Customize the simple Item page

The "simple" Item page is the default display for an Item (when you visit any item via a URL like [dspace.ui.url]/items/[uuid]). If you want to modify the metadata fields displayed on that page by default, that can be done quite easily.

- Normal Item: The code for the simple Item page for a normal Item (i.e. not an Entity) can be found in the source code at "`src/app/item-page/simple/item-types/untyped-item/`"
- Publication Entity: If you are wanting to modify the display of Publication Entities, it has separate source code under "`src/app/item-page/simple/item-types/publication/`"

Here's the basics of modifying this page. The below examples assume you are working with a normal Item. But the same logic would work for modifying the Publication pages (you'd just need to modify it's HTML/CSS instead)

1. First, you'll want to decide if you want to modify just the Item Page HTML, or styles (CSS/Sass), or both.
    a. If you want to modify the HTML, you'll need to create a copy of the HTML in "`src/app/item-page/simple/item-types/untyped-item/untyped-item.component.html`" in your theme. This is where you place your changes.
    b. If you want to modify the styles, you'll need to create a copy of the CSS in "`src/app/item-page/simple/item-types/untyped-item/untyped-item.component.scss`" in your theme. This is where you place your changes.
2. Edit your theme's `app/item-page/simple/item-types/untyped-item/untyped-item.component.ts` file. Swap the "templateUrl" and "styleUrls" properties, based on which you want to modify in your theme. Also, MAKE SURE the "@listableObjectComponent" is using your theme... the last parameter should be the name of your theme!

untyped-item.component.ts

```
// MAKE SURE that the final parameter here is the name of your theme. This one assumes your theme is
named "custom".
@listableObjectComponent(Item, ViewMode.StandalonePage, Context.Any, 'custom')
@Component({
  selector: 'ds-untyped-item',
  // If you want to modify styles, then...
  // Uncomment the styleUrls which references the "untyped-item.component.scss" file in your theme's
directory
  // and comment out the one that references the default in "src/app/"
  styleUrls: ['./untyped-item.component.scss'],
  //styleUrls: ['../../../../../../../app/item-page/simple/item-types/untyped-item/untyped-item.
component.scss'],
  // If you want to modify HTML, then...
  // Uncomment the templateUrl which references the "untyped-item.component.html" file in your theme's
directory
  // and comment out the one that references the default "src/app/"
```

```
  templateUrl: './untyped-item.component.html',
  //templateUrl: '../../../../../../../app/item-page/simple/item-types/untyped-item/untyped-item.
component.html',
})
```

3. Now, based on what you want to modify, you will need to either update your theme's copy of `untyped-item.component.html` or `untyped-item.component.scss` or both.
    a. To change HTML: Your theme's version of the `untyped-item.component.html` file may be empty by default. Copy over the default HTML code from `src/item-page/simple/item-types/untyped-item/untyped-item.component.html` into your version of this file.
    b. To change Styles: Your theme's version of the `untyped-item.component.scss` file may be empty by default. Copy over the default Sass code from `src/item-page/simple/item-types/untyped-item/untyped-item.component.scss` into your version of this file
4. In the HTML of the Simple Item page, you'll see a number of custom "ds-" HTML-like tags which make displaying individual metadata fields easier. These tags make it easier to add/remove metadata fields on this page.
    a. `<ds-generic-item-page-field>` - This tag can be used to display the value of any metadata field (as a string).
        i. Put the name of the metadata field in the "[fields]" attribute... see existing fields as an example.
        ii. Put the i18n label you want to use for this field in the "[label]" attribute. Again, see existing fields as an example. This i18n tag MUST then be added to your "`src/assets/i18n/en.json5`" file (or corresponding file depending on your language)
        iii. For example, here's a new ds-generic-item-page-field which displays the value of the "dc.title.alternative" field with a label defined in the "
    b. `<ds-item-page-uri-field>` - This tag can be used to display the value of any metadata field as an HTML link. (The value must already be a URL)
        i. This field has the same attributes at `<ds-generic-item-page-field>` above.
    c. Some specific tags exist for other fields (e.g. `<ds-item-page-date-field>` and `<ds-item-page-abstract-field>`). These are currently separate components under "`src/app/item-page/simple/field-components/specific-field/`" directories. They are hardcoded to use a specific metadata field and label, but you could customize the components in that location.
5. To add new fields, just add new "<ds-generic-item-page-field>" tags (or similar). To remove fields, just comment them out or remove the HTML. You can also restructure the columns on that page using simple HTML and Bootstrap CSS.
6. Any changes require rebuilding your UI. If you are running in "dev mode" (npm run start:dev), then the UI will restart automatically whenever changes are detected.

NOTE: If your changes to the simple item page don't appear to be working, it can be helpful to restart your dev server to ensure nothing is being cached.

Customize other Components in your Theme

By now, if you've followed this entire guide, you'll notice a pattern! Customizing specific DSpace UI components requires just three steps:

1. *Configure your theme to use its copies of files:* Modify the corresponding `*.component.ts` in your theme.
    a. If you want to modify component style, replace the "styleUrls" in that file to point at the copy of `*.component.scss` in your theme.
    b. If you want to modify component HTML, replace the "template" in that file to point at the copy of `*.component.html` in your theme.
2. *Copy the default UI code into your theme file(s)*
    a. If you want to modify component style, copy the default `*.component.scss` code (from `src/app/`) into your theme's `component.scss` file.
    b. If you want to modify component HTML, copy the default `*.component.html` code (from `src/app/`) into your theme's `component.html` file.
3. *Modify those theme-specific files*
    a. If you want to add images, add them to your theme's `assets/images` folder. Then reference them at the `/assets/[theme-name]/images/` URL path.
    b. Keep in mind, all Bootstrap variables, utility classes & styles can be used in these files. Take advantage of Bootstrap when you can do so.
4. Remember to either rebuild the UI after each change, or run in dev mode (npm run start:dev) while you are doing theme work.

Customize UI labels using Internationalization (i18n) files

Much of the text (like headers and labels) displayed by the DSpace UI is captured in translation (language pack) files to support the use of DSpace in multiple languages. This model of multi-language support is called internationalization (abbreviated i18n). The default set of these i18n files are stored in `src/assets/i18n` and named using the language code, so `en.json5` is the English translation, `fr.json5` is the French translation, etc. In each of these files, a set of "keys" is mapped to text in the given language.

If you would like to change the text displayed in the UI, you will need to edit the i18n translation files. There are two approaches you can take:

1. You can edit the `src/assets/i18n/*.json5` file(s) directly
    a. For example, to change the label for the browse menu when viewing the UI in English (which defaults to "All of DSpace"), you would edit `src/assets/i18n/en.json5` and change the value for `menu.section.browse_global` .
2. And/or, you can create a separate *.json5 file in your theme which only lists the keys you have changed. This can keep your language changes in your theme, and will override the default keys in the `src/assets/i18n/` files. However, a specific setup is necessary, see the "Theme override approach" instructions below.

While editing the default i18n files directly is effective, the recommended approach is to capture i18n changes in your theme. This ensures that your changes to the default values are easy to find and review and also removes the risk of losing your changes when upgrading to newer versions of DSpace.

To capture i18n changes in your theme, you will need to:

1. Create an i18n directory under `src/themes/[theme-name]/assets`

2. For each language you would like to update, add a file to the new i18n directory following the naming scheme in the default i18n directory (en.json5 for English, fr.json5 for French, etc)
3. In each translation file add only the settings that you wish to add or override

There is an example of this configuration in the custom theme, which you can find in `src/themes/custom/assets/i18n` .

Once you have changes in place within your theme, they need to be applied by executing a script:

```
npm run merge-i18n -- -s src/themes/[theme-name]/assets/i18n
```

The `merge-i18n` script will merge the changes captured in your theme with the default settings, resulting in updated versions of the default i18n files. Any setting you included in your theme will override the default setting. Any new properties will be added. Files will be merged based on file name, so en.json5 in your theme will be merged with the en.json5 file in the default i18n directory.

Extending other Themes

Themes can extend other themes using the "extends" configuration. See User Interface Configuration for more examples.

Extending another theme means that you inherit all the settings of the extended theme. So, if the current theme does NOT specify a component style, its ancestor theme(s) will be checked recursively for their styles before falling back to the default. In other words, this "extends" setting allows for a theme to **inherit** all styles/components from the extended theme, and only override those styles/components you wish to override.

Here's a basic example:

Format for 7.2 or above (config.*.yml)

```
themes:
  # grandchild theme
  - name: custom-A
    extends: custom-B
    handle: '10673/34'
  # child theme
  - name: custom-B
    extends: custom
    handle: 10673/2
  # default theme
  - name: custom
```

Format for 7.1 or 7.0 (environment.*.ts)

```
themes: [
  // grandchild theme
  {
    name: 'custom-A',
    extends: 'custom-B',
    handle: '10673/34',
  },
  // child theme
  {
    name: 'custom-B',
    extends: 'custom',
    handle: '10673/2',
  },
  // default theme
  {
    name: 'custom',
  },
],
```

In the above examples:

- When the object at Handle '10673/2' (and any child objects) is viewed, the 'custom-B' theme will be used. By default, you'll have the same styles as the extended 'custom' theme. However, you can override individual styles in your 'custom-B' theme.
- When the object at Handle '10673/34' (and any child objects) is viewed, the 'custom-A' theme will be used. By default, your overall theme will be based on the 'custom' theme (in this case a "grandparent" theme). But, you can override those styles in your 'custom-B' theme or 'custom-A' theme.

- The order of priority is 'custom-A', then 'custom-B', then 'custom'. If a style/component is in 'custom-A' it will be used. If not, 'custom-B' will be checked and if it's there, that version will be used. If not in either 'custom-A' or 'custom-B', then the style/component from 'custom' will be used. If the style/component is not in ANY of those themes, then the default (base theme) style will be used.

Adding Component Directories to your Theme

If you come across an Angular Component which is NOT in your theme but want to customize it, you can add it into your theme directory. This involves copying the component from the "Custom" theme over into your theme.

You can add/copy over a Component Directory as follows:

1. First, copy the Angular Component directory in question from the "Custom" theme folder (src/themes/custom) into your theme's folder. *NOTE: at this time, not all components are theme-able. So, if it doesn't exist in the "Custom" theme folder, then it may not be possible to theme.*
    a. For example, if you wanted to add the Footer Component to your theme, it can be found in the "Custom" theme at "src/themes/custom/app/footer".
    b. Copy that entire folder into your theme folder, retaining the same relative path. For example, to add the Footer Component, copy "src/themes/custom/app/footer" (and all contents) into "src/themes/[your-theme]/app/footer".
2. If you wich to force the standalone API to always include certain components in the initial JS download, you can add a reference to that component in `eager-theme-components.ts`. This should only be done for components that are necessary on every page, such as the header and the footer.
3. If your component uses the `@listableObjectComponent` decorator you should add a reference to that component in `lazy-listable-components.ts`, because those decorators need to be registered when the app starts to be able to be picked up.
4. At this point, you should rebuild/restart your UI to ensure nothing has broken. If you did everything correctly, no build errors will occur. Generally speaking, it's best to add Components one by one, rebuilding in between.
5. Now, you can customize your newly added Component by following the "Customizing Other Components in your Theme" instructions above.

Removing Component Directories from your Theme

While there is no harm in keeping extra, unmodified component directories in your theme, it can be beneficial to *remove component directories which are unchanged.*

The main advantage to keeping your theme simple/small is that it can make future upgrades easier. Generally speaking, the fewer components you have in your theme, the less likely your theme will need modification in a future upgrade (as generally your theme may require updates if one of the components it references underwent structural/major changes).

You can remove a Component directory as follows:

1. First you MUST remove all references to that directory/component from your theme's `lazy-listable-components.ts` and `eager-theme-components.ts` files (if present).
    a. For example, to delete the "./app/header" directory, you'd want to find which component(s) use that directory in your `eager-theme-components.ts` file.
    b. If you search that file, you'd fine this reference:

    ```
    import { HeaderComponent } from './app/header/header.component';
    ```

    c. That means you not only need to remove that "import" statement. You'd also need to remove all other references to "HeaderComponent" in that same `eager-theme-components.ts` file. So, you'd also need to remove it from the COMPONENTS section:

    ```
    export const COMPONENTS = [
      ....
      HeaderComponent,
      ....
    ];
    ```

2. Finally, delete the directory in question from your theme.
3. At this point, you should rebuild/restart your UI to verify nothing has broken. If you did everything correctly, no build errors will occur.
    a. If you failed to edit the `lazy-listable-components.ts` or `eager-theme-components.ts` correctly, you may see "Cannot find module [path-to-module]" errors which reference the directories that Angular/Node can no longer find in your theme. Either restore those directories, or remove the reference(s) from the `lazy-listable-components.ts`/`eager-theme-components.ts` similar to step 1 above.

Adding routes to your theme's new pages

If you create a new "page" in your theme, you will need to link to it from somewhere. This will typically be an `<a/>` element with the `routerLink` attribute giving the name of the Angular route to the component that creates the page. This requires you to define a new route in the **stock** `src/app/app-routes.ts` and `src/app/app-routing-paths.ts`. The route cannot be declared in your theme.

Debugging which theme is being used

While you are working on themes, sometimes you may discover that it's difficult to tell which theme is being used to generate specific HTML elements. Luckily, there's an easy way to determine which theme is used on every HTML element.

Simply view the HTML source of the page, and look for the "data-used-theme" attribute. This attribute will tell you which named theme matched that HTML element. By default, a name of "base" references the core or "base" code (under ./src/app) was used.

For example:

HTML source

```
<!-- This example shows the theme named "dspace" was used for the "themed-header-navbar-wrapper.component.ts" --
>
<ds-themed-header-navbar-wrapper ... data-used-theme="dspace"></ds-themed-header-navabar-wrapper>

<main>
        <!-- But, on the same page, the theme named "base" (core code) was used for the "themed-breadcrumbs.
component.ts" -->
    <ds-themed-breadcrumbs ... data-used-theme="base"></ds-themed-breadcrumbs>
</main>
```

Finding which component is generating the content on a page

See "Finding which component is generating the content on a page" in our User Interface Debugging guide.

Upgrading an Existing Theme

Upgrading from 9.x to 10.x

In 10.0, the following changes were made which will impact your custom theme:

- All "eager-theme.module.ts" and "lazy-theme.module.ts" files were removed in favor of new "eager-theme-components.ts" and "lazy-listable-components.ts". See #4446

In order to upgrade your theme(s) to be compatible with 10.0, you will need to make the following changes:

1. Delete all all the `src/themes/*/lazy-theme.module.ts` files, as these are now unnecessary. (There will be one of these files in each of your existing themes)
2. Create a new `src/themes/*/eager-theme-components.ts` file in each of your themes. Use the `eager-theme-components.ts` file provided with the `dspace` theme as a template/reference.
    a. In that file, update the `COMPONENTS` array by copying over all the components from your `DECLARATIONS` from your existing "eager-theme.module.ts" file. However, exclude the `ENTRY_COMPONENTS` entry. For example.:

```
// If your 9.x theme's existing eager-theme.module.ts includes these declarations:

const DECLARATIONS = [
  ...ENTRY_COMPONENTS,
  HomeNewsComponent,
  HeaderComponent,
  HeaderNavbarWrapperComponent,
  NavbarComponent,
];

// Then your new 10.x eager-theme-components.ts file should contain:
// (NOTE: Make sure to import each of these components to the file)

export const COMPONENTS = [
  HomeNewsComponent,
  HeaderComponent,
  HeaderNavbarWrapperComponent,
  NavbarComponent,
];
```

3. Update the global `src/themes/eager-themes-components.ts` to reference all the `src/themes/*/eager-theme-components.ts` of all of your active themes. For example:

src/themes/eager-themes-components.ts

```
// This example assumes you have a custom theme named "mytheme"
// and have chosen to disable the default "dspace" and "custom" themes.
```

```
import { COMPONENTS as MY_THEME_EAGER_COMPONENTS } from './mytheme/eager-theme-components';

export const EAGER_THEME_COMPONENTS = [
  // ...CUSTOM_THEME_EAGER_COMPONENTS,
  //...DSPACE_THEME_EAGER_COMPONENTS,
  ...MY_THEME_EAGER_COMPONENTS,
];
```

4. Create new `src/themes/*/lazy-listable-components.ts` files in each of your themes. Again, use the `lazy-listable-components.ts` file provided with the `dspace` theme as a template/reference.
    a. In that file, update the `LISTABLE_COMPONENTS` array by copying over all the components from your `ENTRY_COMPONENTS` from your existing "eager-theme.module.ts" file. (NOTE: This list may be empty, as it is in the default "dspace" theme.) For example:

```
// If your 9.x theme's existing eager-theme.module.ts includes these entry components:

const ENTRY_COMPONENTS = [
  JournalComponent,
  JournalIssueComponent,
  JournalVolumeComponent,
];

// Then your new 10.x lazy-listable-components.ts file should contain:
// (NOTE: Make sure to import each of these components to the file)

export const LISTABLE_COMPONENTS = [
  JournalComponent,
  JournalIssueComponent,
  JournalVolumeComponent,
];
```

5. Update the global `src/themes/themes-listable-components.ts` to reference all the `src/themes/*/lazy-listable-components.ts` of all of your active themes. For example:

```
// This example assumes you have a custom theme named "mytheme"
// and have chosen to disable the default "dspace" and "custom" themes.

import { LISTABLE_COMPONENTS as MY_THEME_LISTABLE_COMPONENTS } from './mytheme/lazy-listable-components';

export const THEME_LISTABLE_COMPONENTS = [
  // ...CUSTOM_LISTABLE_COMPONENTS,
  //...DSPACE_LISTABLE_COMPONENTS,
  ...MY_THEME_LISTABLE_COMPONENTS,
];
```

6. Finally, delete all all the `src/themes/*/eager-theme.module.ts` files, as these have now been replaced by the changes you made above. (There will be one of these files in each of your existing themes)

Additional Theming Resources

- User Interface Debugging - Our guide to debugging basic behaviors in the User Interface
- "Getting Started with DSpace 7.0" Basic Workshop at OR2021 Conference
- Bootstrap Documentation - DSpace's UI strives to be compliant with "out-of-the-box" Bootstrap as much as possible. Therefore, Bootstrap knowledge is very beneficial in customizing DSpace.
- Sass Documentation - Bootstrap and DSpace both use Sass to enhance your ability to customize styles quickly via variables, etc. Some familiarity with Sass is recommended, though you need not be an expert.

Accessibility

- Accessibility Statement
- Conformance status
- How we test for accessibility
- Accessibility Compliance Report (VPAT)
- Known limitations
- Report accessibility issues

Accessibility Statement

DSpace is an international, open-source digital repository application that aspires to be as inclusive as possible for all users, including people with disabilities. As a community of users and developers who build and maintain this application, we are dedicated to creating an accessible and interoperable user interface. We are guided by the recommendations of the Web Content Accessibility Guidelines (WCAG) and we continually strive to meet and exceed these standards.

Conformance status

The Web Content Accessibility Guidelines (WCAG) defines requirements for designers and developers to improve accessibility for people with disabilities. It defines three levels of conformance: Level A, Level AA, and Level AAA.

DSpace strives to conform with the current version of WCAG level AA. However, we acknowledge that achieving full accessibility is a work-in-progress at this time.

How we test for accessibility

Development on DSpace is active and ongoing and we use several methods to ensure accessibility for both existing and new development.

- We use design principles and coding standards informed by accessibility concerns as documented in User Interface Design Principles & Accessibility.
- We run automated accessibility scanning tools (Axe by Deque) across the user interface in our end-to-end tests (run via Cypress). These automated tests run for every GitHub pull request submitted to our user interface codebase.
- We ask institutions who use DSpace to share any of their own accessibility testing results with DSpace developers. Accessibility issues discovered are turned into bug tickets for developers to address in upcoming DSpace releases.
    - If your institution has accessibility testing results to share, please contact Tim Donohue or anyone on our DSpace Steering Group.
- In 2021, we conducted an accessibility audit of the DSpace application with Deque to get specific feedback on our accessibility conformance. Their feedback has guided our design and coding standards mentioned above.

Accessibility Compliance Report (VPAT)

Some organizations require a Voluntary Product Accessibility Template (VPAT) for software applications in use or being considered. VPATs are documents that describe how well a product meets accessibility standards. To access the latest VPAT for DSpace (based on version 7.6.2 and 8.0), please visit the DSpace website..

Known limitations

Despite our best efforts to ensure accessibility of DSpace, there may be some limitations. Below is a description of known limitations:

1. We track all known DSpace accessibility issues in our GitHub issue tracker with the "accessibility" label.
2. DSpace development is primarily volunteer-based, and therefore some accessibility tickets may be waiting on a volunteer to claim them. While we do our best to ensure critical issues are addressed quickly, non-critical issues may not receive attention until a volunteer gets to them. We accept code contributions from anyone (in the form of GitHub Pull Requests).
    a. If an issue is important to you and you have developers on staff (or can hire a service provider), please consider contributing a fix back to DSpace. Please claim open tickets by commenting on the issue ticket - this ensures that no other institutions will duplicate efforts.
3. Since the DSpace User Interface allows users to upload content, we cannot ensure the accessibility of user contributions. DSpace has some features that allow administrators to make uploaded content more accessible, but some limitations exist
    a. The MediaViewer (used to view video/audio content) supports subtitles/captioning. However, at this time, the WebVTT captioning files must be uploaded separately alongside the original video.
    b. At this time, DSpace does not support custom alternative text (alt text) for either thumbnail images (generated from uploaded files) or Community/Collection logos.

Report accessibility issues

We consider all accessibility issues to be bugs. Please report any accessibility issues as a GitHub issue ticket. We will prioritize accessibility issues for all of our future releases.

In the ticket, please include the following details:

- What is the accessibility issue you've found? If you know of a way to fix the issue, please include it as well.

- Which page(s) of the DSpace web application can this issue be found on? For example, provide the URL of the page or a description of how to get to that page.
- How could someone reproduce this issue? For example, what tool or browser plugin did you use when you found this issue? If the issue is browser-specific, also note which browser(s) are affected.
- If possible, provide links/screenshots to document the issue or potential fixes. This might include a screenshot showing the issue, a link to WCAG describing the issue or a description from an internal accessibility audit.

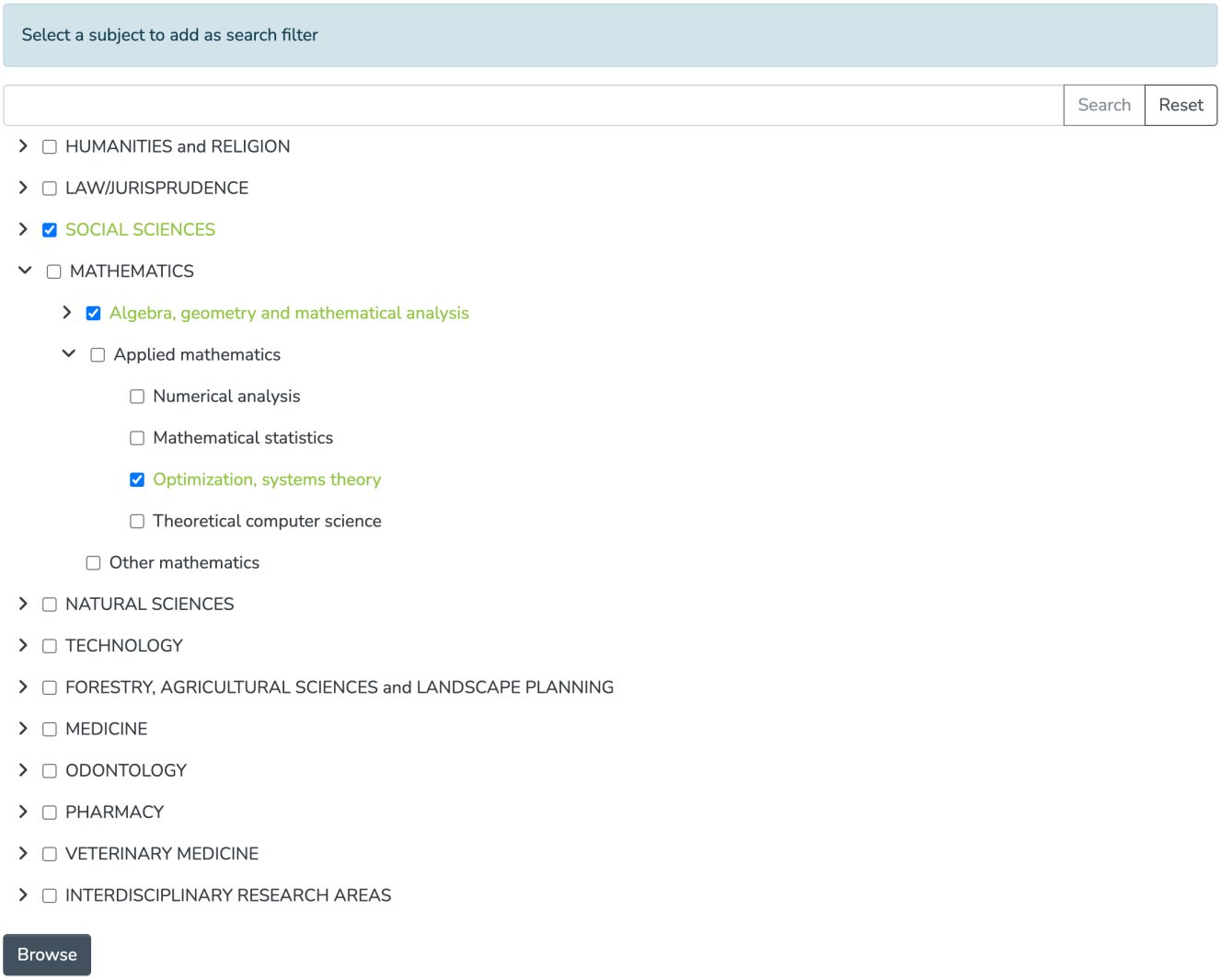
We also welcome contributions / accessibility fixes from anyone. If you've found a way to fix the issue, please submit a GitHub pull request to our codebase . Service providers are also available for hire to fix issues and donate them back to the DSpace codebase.

Browse

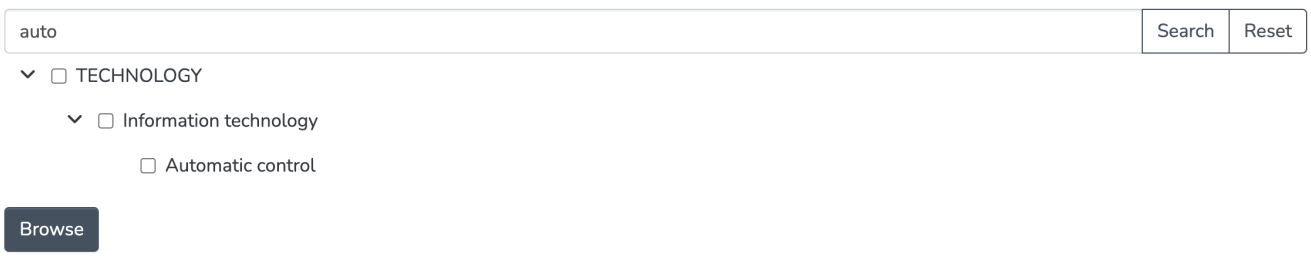
- Browse By Subject Category

Browse By Subject Category

You can find the link to this page by hovering over the "All of DSpace" menu option. On the page, you are presented with the values of the "srsc" vocabulary in a hierarchical tree structure. You can open/close different "branches" of the tree and you can select/deselect values you wish to search on.



You can search for specific values by using the search bar on top of the tree and clicking "Search". Clicking "Reset" will not only reset the tree itself, but also the values you previously selected.



After you're done (de)selecting values, click "Browse". This will redirect you to the search page, where your selected values are used as search filters:

- If one value was selected, the search results will consist of every item which has that value in their `dc.subject` metadata field.
- If multiple values were selected, the search results will consist of the items which have all of the values in their `dc.subject` metadata field. (E.g. if you selected TECHNOLOGY and MEDICINE, only items with *both subjects* will show up.)

To configure Browse by Subject Category options, see "Hierarchical Browse Indexes" in the Configuration Reference.

Contextual Help Tooltips

Available in 7.5 or later.

Contextual help tooltips are a feature to provide additional information about how to use DSpace to less experienced users without cluttering the interface for more advanced users who do not need additional instruction.

- User perspective
- Adding new tooltips

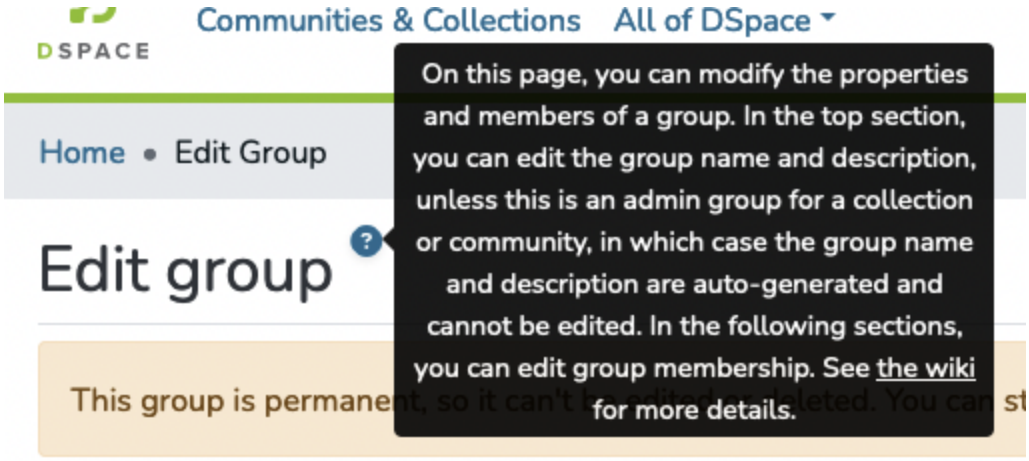
User perspective

If the user visits a page where contextual help tooltips are available, a "toggle context help" button appears in the header, in between the language switch menu and user profile menu. Clicking this button toggles the visibility of the tooltips on the page (by default, they are invisible).



When tooltip visibility is turned on, similar looking buttons appear on the page where ever a tooltip is available.

Clicking any of these buttons makes a text bubble appear containing the contextual help; clicking anywhere outside of the bubble makes it disappear again.



Adding new tooltips

Any HTML element can be given a tooltip by setting the `*dsContextHelp` attribute.

The value assigned to the attribute should be an object of type `ContextHelpDirectiveInput`:

```
export interface ContextHelpDirectiveInput {
  content: string;
  id: string;
  tooltipPlacement?: PlacementArray;
  iconPlacement?: PlacementDir;
}
```

- The mandatory `content` field represents a key in i18n files (src/assets/i18n/*.json5). You will need to add a new key to this file to store the help text.
- `id` should be a unique identifier for this tooltip, to distinguish it from other tooltips on the page.
- `tooltipPlacement` (optional) determines where the text bubble appears relative to the help button. Its type is an array of `Placements`; see the ng-bootstrap documentation for more information.
- `iconPlacement` (optional) should be assigned either 'left' or 'right', and determines whether the tooltip will be placed on the left or on the right of the element.

This is what the template looks like for the "Edit group" example in the "User Perspective" picture above:

```
<h2 class="border-bottom pb-2">
          <span
            *dsContextHelp="{
              content: 'admin.access-control.groups.form.tooltip.editGroupPage',
              id: 'edit-group-page',
              iconPlacement: 'right',
              tooltipPlacement: ['right', 'bottom']
            }"
          >
            {{messagePrefix + '.head.edit' | translate}}
          </span>
</h2>
```

A few important notes:

- Note the use of the `span` tags: setting `*dsContextHelp` directly on the `h2` element makes the help button appear all the way on the right of the page, instead of directly to the right of the "Edit group" text.
- The 'content' field maps to the i18n key which is used to display the help text. This i18n key's value may include markdown-style **links** (only)**.** At this time, other formatting is not supported. To display a link, use the following markdown syntax in your i18n value:

  ```
  "admin.access-control.groups.form.tooltip.editGroupPage": "This is the text that would be displayed in the tooltip. And here is a link to the [DSpace 7 documentation](https://wiki.lyrasis.org/display/DSDOC7x/)."
  ```

Discovery

1 What is DSpace Discovery
    1.1 What is a Sidebar Facet
    1.2 What is a Search Filter
    1.3 What is a tag cloud facet
2 Configuration files
3 General Discovery settings (config/modules/discovery.cfg)
4 Browse settings (config/dspace.cfg)
5 Modifying the Discovery User Interface (config/spring/api/discovery.xml)
    5.1 Structure Summary
    5.2 Default settings
    5.3 Non indexed metadata fields
    5.4 Search filters & sidebar facets Customization
        5.4.1 Hierarchical (taxonomies based) sidebar facets
    5.5 Sort option customization for search results
    5.6 DiscoveryConfiguration
        5.6.1 Configuring lists of sidebarFacets and searchFilters
        5.6.2 Configuring and customizing search sort fields
        5.6.3 Adding default filter queries (OPTIONAL)
        5.6.4 Access Rights Awareness
            5.6.4.1 Access Rights Awareness - technical details
        5.6.5 Customizing the Recent Submissions display
        5.6.6 Customizing hit highlighting & search snippets
            5.6.6.1 Hit highlighting technical details
        5.6.7 "More like this" configuration
            5.6.7.1 "More like this" technical details
        5.6.8 "Did you mean" spellcheck aid for search configuration
            5.6.8.1 "Did you mean" spellcheck aid for search technical details
        5.6.9 Customizing the "Tag Cloud" facet
        5.6.10 Disabling the "Has file(s)" facet
6 Discovery Solr Index Maintenance
7 Advanced Solr Configuration

What is DSpace Discovery

The Discovery Module enables faceted searching & browsing for your repository.

Although these techniques are new in DSpace, they might feel familiar from other platforms like Aquabrowser or Amazon, where facets help you to select the right product according to facets like price and brand. DSpace Discovery offers very powerful browse and search configurations that were only possible with code customization in the past.

Watch the DSpace Discovery introduction video

Since 6.0, Discovery is the only out-of-the-box Search and Browse infrastructure provided in DSpace.

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.



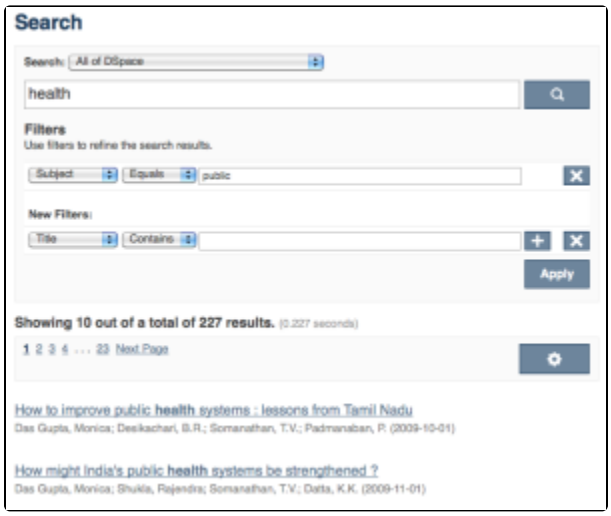
In this example, there are 3 Sidebar Facets: Author, Subject and Date Issued. It's important to know that multiple metadata fields can be included in one facet. For example, the Author facet above includes values from both dc.contributor.author as well as dc.creator.

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.

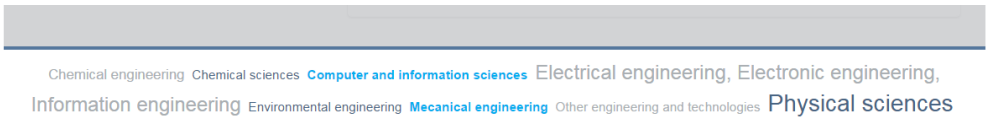
In a faceted search, a user can modify the list of displayed search results by specifying additional "filters" that will be applied on the list of search results. In DSpace, a filter is a contain condition applied to specific facets. In the example below, a user started with the search term "health", which yielded 500 results. After applying the filter "public" on the facet "Subject", only 227 results remain. Each time a user selects a sidebar facet it will be added as a filter. Active filters can be altered or removed in the 'filters' section of the search interface.



Another example: Using the standard search, a user would search for something like [**wetland + "dc.author=Mitsch, William J" + dc.subject="water quality"**]. With filtered search, they can start by searching for [**wetland**], and then filter the results by the other attributes, author and subject.

What is a tag cloud facet

Tag cloud facet is another way to display facets of your repository in a "tag cloud" form in which the importance of each tag is show with font size or color. This format is useful for quickly perceiving the most prominent terms.



This is a classic "tag cloud" facet in a DSpace repository.

Configuration files

The configuration for discovery is located in 2 separate files.

- General settings: The `discovery.cfg` file located in the `[dspace-install-dir]/config/modules directory`.
- Browse settings: The dspace.cfg file located in `[dspace-install-dir]/config/` includes "webui.browse.index.*" settings
- User Interface Configuration: The `discovery.xml` file is located in `[dspace-install-dir]/config/spring/api/` directory.

General Discovery settings (`config/modules/discovery.cfg`)

The `discovery.cfg` file is located in the `[dspace]/config/modules` directory and contains following properties. Any of these properties may be overridden in your `local.cfg` (see Configuration Reference):

Property:	**discovery.search.server**
Example Value:	`discovery.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. If you are uncertain whether this property is set correctly, you can use a commandline tool like "wget" to perform a query against the Solr index (and ensure Solr responds). For example, the below query searches the Solr index for "test" and returns the response on standard out: `wget -O - http://localhost:8080/solr/search/select?q=test`
Property:	**discovery.index.authority.ignore[.field]**

Example Value:	`discovery.index.authority.ignore=true`
	
	`discovery.index.authority.ignore.dc.contributor.author=false`

Informational Note:	By default, Discovery will use the authority information in the metadata to disambiguate homonyms. Setting this property to false will make the indexing process the same as the metadata doesn't include authority information. The configuration can be different on a field (<schema>.<element>.<qualifier>) basis. Setting the property without a field will change the default value.
Property:	**discovery.browse.authority.ignore[.browse-index]**

Example Value:	`discovery.browse.authority.ignore=true`
	
	`discovery.browse.authority.ignore.author=false`

Informational Note:	*Similar property to "discovery.index.authority.ignore", except specific to the "Browse By" indexes.* By default, Discovery will use the authority information in the metadata to disambiguate homonyms. Setting this property to false will make the indexing process the same as the metadata doesn't include authority information. The configuration can be different on a browse index basis. Setting the property without a browse index will change the default value.
Property:	**discovery.index.authority.ignore-prefered[.field]**

Example Value:	`discovery.index.authority.ignore-prefered=true`
	
	`discovery.index.authority.ignore-prefered.dc.contributor.author=false`

Informational Note:	By default, Discovery will use the authority information in the metadata to query the authority for the preferred label. Setting this property to false will make the indexing process the same as the metadata doesn't include authority information (i.e. the preferred form is the one recorded in the metadata value). The configuration can be different on a field (<schema>.<element>.<qualifier>) basis. Setting the property without a field will change the default value. **If the authority is a remote service, disabling this feature can greatly improve performance.**
Property:	**discovery.browse.authority.ignore-prefered[.browse-index]**

Example Value:	`discovery.browse.authority.ignore-prefered=true`
	
	`discovery.browse.authority.ignore-prefered.author=false`

Informational Note:	*Similar property to "discovery.index.authority.ignore-prefered", except specific to the "Browse By" indexes.* By default, Discovery will use the authority information in the metadata to query the authority for the preferred label. Setting this property to false will make the indexing process the same as the metadata doesn't include authority information (i.e. the preferred form is the one recorded in the metadata value). The configuration can be different on a browse index basis. Setting the property without a browse index will change the default value. **If the authority is a remote service, disabling this feature can greatly improve performance.**
Property:	**discovery.index.authority.ignore-variants[.field]**

Example Value:	`discovery.index.authority.ignore-variants=true`
	
	`discovery.index.authority.ignore-variants.dc.contributor.author=false`

Informational Note:	By default, Discovery will use the authority information in the metadata to query the authority for variants. Setting this property to false will make the indexing process the same, as the metadata doesn't include authority information. The configuration can be different on a per-field (<schema>.<element>.<qualifier>) basis. Setting the property without a field will change the default value. **If authority is a remote service, disabling this feature can greatly improve performance.**
Property:	**discovery.browse.authority.ignore-variants[.browse-index]**

Example Value:	`discovery.browse.authority.ignore-variants=true`
	
	`discovery.browse.authority.ignore-variants.author=false`

Informational Note:	*Similar property to "discovery.index.authority.ignore-variants", except specific to the "Browse By" indexes.* By default, Discovery will use the authority information in the metadata to query the authority for variants. Setting this property to false will make the indexing process the same, as the metadata doesn't include authority information. The configuration can be different on a browse index basis. Setting the property without a browse index will change the default value. **If authority is a remote service, disabling this feature can greatly improve performance.**

Browse settings (`config/dspace.cfg`)

See the "Browse Index Configuration" section of the Configuration Reference for all "Browse By" configurations. These Configurations control both which fields are indexed for browsing, as well as which Browse by options appear in the user interface. Changing these configurations requires reindexing.

If you add new browse fields then you should coordinate changes here with the message catalog(s) in the UI. You will need to add several entries:

key	value
browse.comcol.by.*	label the browse field in the Browse menu of a community or collection page
menu.section.browse_global_by_*	label the browse field in the "browse" dropdown at the top of the page
browse.metadata.*	label the browse field in the body of the browsing page
browse.metadata.*.breadcrumbs	label the browse field in the page's "breadcrumb trail"

Modifying the Discovery User Interface (`config/spring/api/discovery.xml`)

The `discovery.xml` file is located in the `[dspace]/config/spring/api` directory. Modifying these settings can change the behavior of the Search pages in the user interface, allowing you to customize the facets, filters, sort options, etc.

If you add new search fields, sorts, etc. then you should coordinate changes here with the message catalog(s) in the UI. You will need to add e.g. `search.filters.filter.NAME.head` (where NAME is the `indexFieldName` of the field definition) to label a new search field.

Structure Summary

This file is in XML format. You should be familiar with XML before editing this file. The configurations are organized together in beans, depending on the purpose these properties are used for.
This purpose can be derived from the class of the beans. Here's a short summary of classes you will encounter throughout the file and what the corresponding properties in the bean are used for.

Download the configuration file and review it together with the following parameters

Class:	**DiscoveryConfigurationService**
Purpose:	Defines the mapping between separate Discovery configurations and individual collections/communities
Default:	All communities, collections and the homepage (key=default) are mapped to defaultConfiguration. Also controls the metadata fields that should not be indexed in the search core (item provenance for example).
Class:	**DiscoveryConfiguration**
Purpose:	Groups configurations for sidebar facets, search filters, search sort options and recent submissions
Default:	There is one configuration by default called defaultConfiguration
Class:	**DiscoverySearchFilter**
Purpose:	Defines that specific metadata fields should be enabled as a search filter
Default:	dc.title, dc.contributor.author, dc.creator, dc.subject.* and dc.date.issued are defined as search filters
Class:	**DiscoverySearchFilterFacet**
Purpose:	Defines which metadata fields should be offered as a contextual sidebar browse options, each of these facets has also got to be a search filter
Default:	dc.contributor.author, dc.creator, dc.subject.* and dc.date.issued
Class:	**HierarchicalSidebarFacetConfiguration**
Purpose:	Defines which metadata fields contain hierarchical data and should be offered as a contextual sidebar option
Class:	**DiscoverySortConfiguration**
Purpose:	Further specifies the sort options to which a DiscoveryConfiguration refers
Default:	dc.title and dc.date.issued are defined as alternatives for sorting, other than Relevance (hard-coded)
Class:	**DiscoveryHitHighlightingConfiguration**
Purpose:	Defines which metadata fields can contain hit highlighting & search snippets
Default:	dc.title, dc.contributor.author, dc.subject, dc.description.abstract & full text from text files.
Class:	**TagCloudFacetConfiguration**
Purpose:	Defines the tag cloud appearance configuration bean and the search filter facets to appear in the tag cloud form. You can have different "**TagCloudFacetConfiguration**" per community or collection or the home page

Default settings

In addition to the summarized descriptions of the default values, following details help you to better understand these defaults. If you haven't already done so, download the configuration file and review it together with the following parameters.
The file contains one default configuration that defines following sidebar facets, search filters, sort fields and recent submissions display:

- Sidebar facets
  - **searchFilterAuthor:** groups the metadata fields dc.contributor.author & dc.creator with a facet limit of 10, sorted by occurrence count
  - **searchFilterSubject:** groups all subject metadata fields (dc.subject.*) with a facet limit of 10, sorted by occurrence count
  - **searchFilterIssued:** contains the dc.date.issued metadata field, which is identified with the type "date" and sorted by specific date values

- Search filters
  - **searchFilterTitle:** contains the dc.title metadata field
  - **searchFilterAuthor:** contains the dc.contributor.author & dc.creator metadata fields
  - **searchFilterSubject:** contains the dc.subject.* metadata fields
  - **searchFilterIssued:** contains the dc.date.issued metadata field with the type "date"
- Sort fields
  - **sortTitle**: contains the dc.title metadata field
  - **sortDateIssued:** contains the dc.date.issued metadata field, this sort has the type date configured.
- defaultFilterQueries
  - The default configuration contains no defaultFilterQueries
  - The default filter queries are disabled by default but there is an example in the default configuration in comments which allows discovery to only return items (as opposed to also communities/collections).
- Recent Submissions
  - The recent submissions are sorted by dc.date. accessioned which is a date and a maximum number of 5 recent submissions are displayed.
- Hit highlighting
  - The fields dc.title, dc.contributor.author & dc.subject can contain hit highlighting.
  - The dc.description.abstract & full text field are used to render search snippets.
- Non indexed metadata fields
  - **Community/Collections:** dc.rights (copyright text)
  - **Items:** dc.description.provenance

Many of the properties contain lists that use references to point to the configuration elements. This way a certain configuration type can be used in multiple discovery configurations so there is no need to duplicate them.

Non indexed metadata fields

The discovery.xml file has configuration to not index certain metadata fields for communities/collections/items. The configuration is handled in the "toIgnoreMetadataFields" property located in the "org.dspace.discovery.configuration.DiscoveryConfigurationService" bean. Below is an example configuration that excludes dc.description.provenance for items & dc.rights for communities/collections:

```
<property name="toIgnoreMetadataFields">
    <map>
        <entry>
            <key><util:constant static-field="org.dspace.core.Constants.COMMUNITY"/></key>
            <list>
                <!--Introduction text-->
                <!--<value>dc.description</value>-->
                <!--Short description-->
                <!--<value>dc.description.abstract</value>-->
                <!--News-->
                <!--<value>dc.description.tableofcontents</value>-->
                <!--Copyright text-->
                <value>dc.rights</value>
                <!--Community name-->
                <!--<value>dc.title</value>-->
            </list>
        </entry>
        <entry>
            <key><util:constant static-field="org.dspace.core.Constants.COLLECTION"/></key>
            <list>
                <!--Introduction text-->
                <!--<value>dc.description</value>-->
                <!--Short description-->
                <!--<value>dc.description.abstract</value>-->
                <!--News-->
                <!--<value>dc.description.tableofcontents</value>-->
                <!--Copyright text-->
                <value>dc.rights</value>
                <!--Collection name-->
                <!--<value>dc.title</value>-->
            </list>
        </entry>
        <entry>
            <key><util:constant static-field="org.dspace.core.Constants.ITEM"/></key>
            <list>
                <value>dc.description.provenance</value>
            </list>
        </entry>
    </map>
</property>
```

By adding additional values to the appropriate lists additional metadata can be excluded from the search core, a reindex is required after altering this file to ensure that the values are removed from the index.

Search filters & sidebar facets Customization

This section explains the properties for search filters & sidebar facets. Each sidebar facet must occur in the reference list of the search filters. Below is an example configuration of a search filter that is not used as a sidebar facet.

```
<bean id="searchFilterTitle" class="org.dspace.discovery.configuration.DiscoverySearchFilter">
    <property name="indexFieldName" value="title"/>
    <property name="metadataFields">
        <list>
            <value>dc.title</value>
        </list>
    </property>
    <property name="pageSize" value="10"/>
</bean>
```

The id & class attributes are mandatory for this type of bean. The properties that it contains are discussed below.

- **indexFieldName** (Required)**:** A unique search filter name, the metadata will be indexed in Solr under this field name.
- **metadataFields** (Required): A list of the metadata fields that need to be included in the facet.

Sidebar facets extend the search filter and add some extra properties to it. Below is an example of a search filter that is also used as a sidebar facet.

```
<bean id="searchFilterAuthor" class="org.dspace.discovery.configuration.DiscoverySearchFilterFacet">
        <property name="indexFieldName" value="author"/>
        <property name="metadataFields">
            <list>
                <value>dc.contributor.author</value>
                <value>dc.creator</value>
            </list>
        </property>
        <property name="facetLimit" value="5"/>
        <property name="sortOrderSidebar" value="COUNT"/>
        <property name="sortOrderFilterPage" value="COUNT"/>
        <property name="isOpenByDefault" value="true"/>
        <property name="type" value="text"/>
    </bean>
```

Note that the class has changed from **DiscoverySearchFilter** to **DiscoverySearchFilterFacet**. This is needed to support the extra properties.

- **facetLimit** (optional)**:** The maximum number of values to be shown by default. This property is optional, if none is specified the default value "10" will be used. If the filter has the type **date**, this property will not be used since dates are automatically grouped together.
- **sortOrder** (optional)**:**The sort order for the sidebar facets, it can either be COUNT or VALUE. The default value is COUNT.
    - **COUNT** Facets will be sorted by the number of times they appear in the repository
    - **VALUE** Facets will be sorted alphabetically
- **type** (optional): the type of the sidebar facet it can either be "date" or "text". "text" is the default value.
    - **text:** The facets will be treated as is (DEFAULT)
    - **date**: Only the year will be stored in the Solr index. These years are automatically displayed in ranges that get smaller when you select one.

Hierarchical (taxonomies based) sidebar facets

Discovery supports specialized drill down in hierarchically structured metadata fields. For this drill down to work, the metadata in the field for which you enable this must be composed out of terms, divided by a splitter. For example, you could have a dc.subject.taxonomy field in which you keep metadata like "CARTOGRAPHY::PHOTOGRAMMETRY", in which Cartography and Photogrammetry are both terms, divided by the splitter "::". Initially the sidebar will only display the top level facets. When clicking on "view more" all the facet options will be displayed.

```
<bean id="searchFilterSubject" class="org.dspace.discovery.configuration.HierarchicalSidebarFacetConfiguration">
    <property name="indexFieldName" value="subject"/>
    <property name="metadataFields">
        <list>
            <value>dc.subject</value>
        </list>
    </property>
    <property name="sortOrder" value="COUNT"/>
    <property name="splitter" value="::"/>
    <property name="skipFirstNodeLevel" value="false"/>
</bean>
```

Note that the class has changed from **SidebarFacetConfiguration** to **HierarchicalSidebarFacetConfiguration**. This is needed to support the extra properties.

- **splitter** (required)**:** The splitter used to split up the separate nodes
- **skipFirstNodeLevel** (optional)**:** Whether or not to show the root node level. For some hierarchical data there is a single root node. In most cases it doesn't need to be shown since it isn't relevant. **This property is true by default.**

Sort option customization for search results

This section explains the properties of an individual SortConfiguration, like sortTitle and sortDateIssued from the default configuration. In order to create custom sort options, you can either modify specific properties of those that already exist or create a totally new one from scratch.

Here's what the sortTitle SortConfiguration looks like:

```
<bean id="sortTitle" class="org.dspace.discovery.configuration.DiscoverySortFieldConfiguration">
       <property name="metadataField" value="dc.title"/>
       <property name="type" value="text"/>
 </bean>
```

The `id` and `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 of six parts:

- The list of applicable sidebarFacets
- The list of applicable searchFilters
- The list of applicable searchSortFields
- Any default filter queries (optional)
- The configuration for the Recent submissions display
- The configuration of the tag cloud facet

Configuring lists of sidebarFacets and searchFilters

After modifying sidebarFacets and searchFilters, don't forget to reindex existing items by running `[dspace]/bin/dspace index-discovery -b`, otherwise the changes will not appear.

Below is an example of how one of these lists can be configured. It's important that each of the bean references corresponds to the exact name of the earlier defined facets, filters or sort options.

Each sidebar facet must also occur in the list of the search filters.

```
<property name="sidebarFacets">
    <list>
        <ref bean="sidebarFacetAuthor" />
        <ref bean="sidebarFacetSubject" />
        <ref bean="sidebarFacetDateIssued" />
    </list>
</property>
```

Configuring and customizing search sort fields

The search sort field configuration block contains the available sort fields and the possibility to configure a default sort field and sort order.
Below is an example of the sort configuration.

```
<property name="searchSortConfiguration">
    <bean class="org.dspace.discovery.configuration.DiscoverySortConfiguration">
        <!--<property name="defaultSort" ref="sortDateIssued"/>-->
        <!--DefaultSortOrder can either be desc or asc (desc is default)-->
        <property name="defaultSortOrder" value="desc"/>
        <property name="sortFields">
            <list>
                <ref bean="sortTitle" />
                <ref bean="sortDateIssued" />
            </list>
        </property>
    </bean>
</property>
```

The property name & the bean class are mandatory. The property field names are 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 and sidebar facet clicks. One useful application of default filter queries is ensuring that all returned results are items. As a result, subcommunities and collections that are returned as results of the search operation, are filtered out.
Similar to the lists above, the default filter queries are defined as a list. They are optional.

```
<property name="defaultFilterQueries">
    <list>
        <value>query1</value>
        <value>query2</value>
    </list>
</property>
```

This property contains a simple list which in turn contains the queries. Some examples of possible queries:

- search.resourcetype:2
- dc.subject:test
- dc.contributor.author: "Van de Velde, Kevin"
- ...

Access Rights Awareness

By default, when searching and browsing using Discovery, you will only see items that you have access to. So, your search/browse results may differ if you are logged into DSpace. This Access Rights Awareness feature ensures that anonymous users (and search engines) are not able to access information (both files and metadata) about embargoed or private items. It also provides you with more direct control over who can see individual items within your DSpace.

How does Access Rights Awareness work?

Access Rights Awareness checks the "READ" access on the Item.

If the "Anonymous" group has "READ" access on the Item, then anonymous/public users will be able to view that Item's metadata and locate that Item via DSpace's search/browse system. In addition, search engines will also be able to index that Item's metadata. However, even with Anonymous READ set at the Item-level, you may still choose to access-restrict the downloading/viewing of *files* within the Item. To do so, you would restrict "READ" access on individual Bitstream(s) attached to the Item.

If the "Anonymous" group does NOT have "READ" access on the Item, then anonymous users will never see that Item appear within their search/browse results (essentially the Item is "invisible" to them). In addition, that Item will be invisible to search engines, so it will never be indexed by them. However, any users who have been given READ access will be able to find/locate the item after logging into DSpace. For example, if a "Staff" group was provided "READ" access on the Item, then members of that "Staff" group would be able to locate the item via search/browse after logging into DSpace.

How can I disable Access Rights Awareness?

If you prefer to allow all access-restricted or embargoed Items to be findable within your DSpace, you can choose to turn off Access Rights Awareness. However, please be aware that this means that restricting "READ" access on an Item will not really do anything – the Item metadata will be available to the public no matter what group(s) were given READ access on that Item.

This feature can be switched off by going to the `[dspace.dir]/config/spring/api/discovery.xml` file & commenting out the bean & the alias shown below.

```
<bean class="org.dspace.discovery.SolrServiceResourceRestrictionPlugin" id="solrServiceResourceIndexPlugin"/>

<alias name="solrServiceResourceIndexPlugin" alias="org.dspace.discovery.SolrServiceResourceRestrictionPlugin"/>
```

The Browse Engine only supports the "Access Rights Awareness" if the Solr/Discovery backend is enabled (see Defining the Storage of the Browse Data). However, it is enabled by default for DSpace 3.x and above.

Access Rights Awareness - technical details

The *DSpaceObject* class has an *updateLastModified()* method which will be triggered each time an authorization policy changes. This method is only implemented in the item class where the last_modified timestamp will be updated and a modify event will be fired. By doing this we ensure that the discovery consumer is called and the item is reindexed. Since this feature can be switched off a separate plugin has been created: the *SolrServiceResourceRestrictionPlugin.* Whenever we reindex a DSpace object all the read rights will be stored in the read field. We make a distinction between groups and users by adding a *'g'* prefix for groups and the *'e'* prefix for epersons.

When searching in discovery all the groups the user belongs to will be added as a filter query as well as the users identifier. If the user is an admin all items will be returned since an admin has read rights on everything.

Customizing the Recent Submissions display

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 and 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.

Customizing hit highlighting & search snippets

The hit highlighting configuration element contains all settings necessary to display search snippets & enable hit highlighting.

Disabling hit highlighting / search snippets

You can disable hit highlighting / search snippets by commenting out the entire `<property name="hitHighlightingConfiguration">` Configuration in the `[dspace]/config/spring/api/discovery.xml` configuration file.

PLEASE BE AWARE there are two sections where this <property> definition exists. You should comment out both. One is under the `<bean id="defaultConfiguration">` and one is under the `<bean id="homepageConfiguration">`

Alternatively, you may also choose to tweak which fields are shown in hit highlighting, or modify the number of matching words shown (snippets) and/or number of characters shown around the matching word (maxSize).

For this change to take effect in the User Interface, you will need to restart Tomcat.

Changes made to the configuration will not automatically be displayed in the user interface. By default, only the following fields are displayed: dc.title, dc.contributor.author, dc.creator, dc.contributor, dc.date.issued, dc.publisher, dc.description.abstract and fulltext.

If additional fields are required, look for the "itemSummaryList" template.

Below is an example configuration of hit highlighting.

```
<property name="hitHighlightingConfiguration">
    <bean class="org.dspace.discovery.configuration.DiscoveryHitHighlightingConfiguration">
        <property name="metadataFields">
            <list>
                <bean class="org.dspace.discovery.configuration.DiscoveryHitHighlightFieldConfiguration">
                    <property name="field" value="dc.title"/>
                    <property name="snippets" value="5"/>
                </bean>
                <bean class="org.dspace.discovery.configuration.DiscoveryHitHighlightFieldConfiguration">
                    <property name="field" value="dc.contributor.author"/>
                    <property name="snippets" value="5"/>
                </bean>
                <bean class="org.dspace.discovery.configuration.DiscoveryHitHighlightFieldConfiguration">
                    <property name="field" value="dc.subject"/>
                    <property name="snippets" value="5"/>
                </bean>
                <bean class="org.dspace.discovery.configuration.DiscoveryHitHighlightFieldConfiguration">
                    <property name="field" value="dc.description.abstract"/>
                    <!-- Max number of characters to display around the matching word (Warning setting to 0
returns entire field) -->
                    <property name="maxSize" value="250"/>
                    <!-- Max number of snippets (matching words) to show -->
                    <property name="snippets" value="2"/>
                </bean>
                <bean class="org.dspace.discovery.configuration.DiscoveryHitHighlightFieldConfiguration">
                                    <!-- Displays snippets from indexed full text of document (for
supported formats) -->
                    <property name="field" value="fulltext"/>
                    <!-- Max number of characters to display around the matching word (Warning setting to 0
returns entire field) -->
                    <property name="maxSize" value="250"/>
                                    <!-- Max number of snippets (matching words) to show -->
                    <property name="snippets" value="2"/>
                </bean>
            </list>
        </property>
    </bean>
</property>
```

The property name and the bean class are mandatory. The property field names are:

- **field** (mandatory)**:** The metadata field to be highlighted (can also be * if all the metadata fields should be highlighted).
- **maxSize** (optional): Limit the number of characters displayed to only the relevant part (use metadata field as search snippet).
- **snippets** (optional)**:** The maximum number of snippets that can be found in one metadata field.

Hit highlighting technical details

The *org.dspace.discovery.DiscoveryQuery* object has a setter & getter for the hit highlighting configuration set in Discovery configuration. If this configuration is given the *resolveToSolrQuery* method located in the *org.dspace.discovery.SolrServiceImpl* class will use the standard Solr highlighting feature (http://wiki.apache.org/solr/HighlightingParameters). The *org.dspace.discovery.DiscoverResult* class has a method to set the highlighted fields for each object & field.

The rendering of search results is no longer handled by the METS format but uses a special type of list named "TYPE_DSO_LIST". Each metadata field (& fulltext if configured) is added in the DRI and IF the field contains hit higlighting the Java code will split up the string & add *DRI highlights* to the list. The XSL for the themes also contains special rendering XSL for the DRI; for Mirage, the changes are located in the *discovery.xsl* file. For themes using the old themes based on structural.xsl, look for the template matching "*dri:list[@type='dsolist']*".

"More like this" configuration

The "more like this"-configuration element contains all the settings for displaying related items on an item display page.
Below is an example of the "more like this" configuration.

```
<property name="moreLikeThisConfiguration">
    <bean class="org.dspace.discovery.configuration.DiscoveryMoreLikeThisConfiguration">
        <property name="similarityMetadataFields">
            <list>
                <value>dc.title</value>
                <value>dc.contributor.author</value>
                <value>dc.creator</value>
                <value>dc.subject</value>
            </list>
        </property>
        <!--The minimum number of matching terms across the metadata fields above before an item is found as
related -->
        <property name="minTermFrequency" value="5"/>
        <!--The maximum number of related items displayed-->
        <property name="max" value="3"/>
        <!--The minimum word length below which words will be ignored-->
        <property name="minWordLength" value="5"/>
    </bean>
</property>
```

The property name and the bean class are mandatory. The property field names are discussed below.

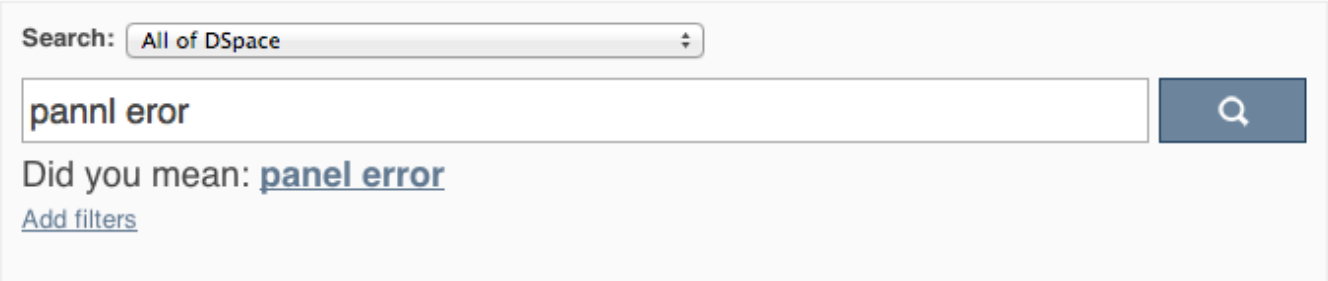
- similarityMetadataFields: the metadata fields checked for similarity
- minTermFrequency: The minimum number of matching terms accross the metadata fields above before an item is found as related
- max: The maximum number of related items displayed
- minWordLength: The minimum word length below which words will be ignored

"More like this" technical details

The *org.dspace.discovery.SearchService* object has received a *getRelatedItems()* method. This method requires an item & the more-like-this configuration bean from above. This method is implemented in the *org.dspace.discovery.SolrServiceImpl* which uses the item as a query & uses the default Solr parameters for more-like-this to pass the bean configuration to solr (https://cwiki.apache.org/confluence/display/solr/MoreLikeThis). The result will be a list of items or if none found an empty list.

"Did you mean" spellcheck aid for search configuration

DSpace 4 introduces the use of SOLR's SpellCheckComponent as an aid for search. When a user's search does not return any hits, the user is presented with a suggestion for an alternative search query.



The feature currently only one line of configuration to discovery.xml. Changing the value from true to false will disable the feature.

```
<property name="spellCheckEnabled" value="true" />
```

"Did you mean" spellcheck aid for search technical details

Similar to the More like this configuration, SOLR's spell check component is used with default configuration values. Any of these values can be overridden in the solrconfig.xml file located in dspace/solr/search/conf/. Following links provide more information about the SOLR SpellCheckComponent:

http://wiki.apache.org/solr/SpellCheckComponent

https://cwiki.apache.org/confluence/display/solr/Spell+Checking

Customizing the "Tag Cloud" facet

Not yet supported in DSpace 7.0

```
<!-- Set TagCloud configuration per discovery configuration -->
<property name="tagCloudFacetConfiguration" ref="defaultTagCloudFacetConfiguration"/>
```

*Declare the bean (of class: **TagCloudFacetConfiguration**) that holds the configuration for the tag cloud facet.*

```
<!--TagCloud configuration bean for homepage discovery configuration-->
<bean id="homepageTagCloudFacetConfiguration" class="org.dspace.discovery.configuration.
TagCloudFacetConfiguration">
        <!-- Actual configuration of the tagcloud (colors, sorting, etc.) -->
        <property name="tagCloudConfiguration" ref="tagCloudConfiguration"/>
        <!-- List of tagclouds to appear, one for every search filter, one after the other -->
        <property name="tagCloudFacets">
            <list>
                <ref bean="searchFilterSubject" />
            </list>
        </property>
</bean>
```

This bean has two properties:

- **tagCloudConfiguration**: is the bean which describes the actual appearance parameters
- **tagCloudFacets**: the search filter facets which will be used for the tag cloud. If you leave the list empty, no tag cloud will appear. If you declare more than one, such number of tag clouds will appear for each search filter, one after the other.

The appearance configuration can have the following properties, as shown in the following bean:

```
<bean id="tagCloudConfiguration" class="org.dspace.discovery.configuration.TagCloudConfiguration">
             <!-- Should display the score of each tag next to it? Default: false -->
             <property name="displayScore" value="true"/>
             <!-- Should display the tag as center aligned in the page or left aligned? Possible values: true
| false. Default: true  -->
             <property name="shouldCenter" value="true"/>
             <!-- How many tags will be shown. Value -1 means all of them. Default: -1 -->
             <property name="totalTags" value="-1"/>
             <!-- The letter case of the tags.
                     Possible values: Case.LOWER | Case.UPPER | Case.CAPITALIZATION | Case.PRESERVE_CASE |
Case.CASE_SENSITIVE
                     Default: Case.PRESERVE_CASE -->
             <property name="cloudCase" value="Case.PRESERVE_CASE"/>
             <!-- If the 3 CSS classes of the tag cloud should be independent of score (random=yes) or based
on the score. Possible values: true | false . Default: true-->
             <property name="randomColors" value="true"/>
             <!-- The font size (in em) for the tag with the lowest score. Possible values: any decimal.
Default: 1.1 -->
             <property name="fontFrom" value="1.1"/>
             <!-- The font size (in em) for the tag with the lowest score. Possible values: any decimal.
Default: 3.2 -->
             <property name="fontTo" value="3.2"/>
             <!-- The score that tags with lower than that will not appear in the rag cloud. Possible values:
any integer from 1 to infinity. Default: 0 -->
             <property name="cuttingLevel" value="0"/>
             <!-- The distance (in px) between the tags. Default: 5 -->
             <property name="marginRight" value="5"/>
             <!-- The ordering of the tags (based either on the name or the score of the tag)
                     Possible values: Tag.NameComparatorAsc | Tag.NameComparatorDesc | Tag.ScoreComparatorAsc
| Tag.ScoreComparatorDesc
                     Default: Tag.NameComparatorAsc  -->
             <property name="ordering" value="Tag.NameComparatorAsc"/>
    </bean>
```

When tagCloud is rendered there are some CSS classes that you can change in order to change the appearance of the tag cloud.

Class	Note
tagcloud	General class for the whole tagcloud
tagcloud_1	Specific tag class for tag of type 1 (based on score)
tagcloud_2	Specific tag class for tag of type 2 (based on score)
tagcloud_3	Specific tag class for tag of type 3 (based on score)

Disabling the "Has file(s)" facet

Since DSpace 6, a new "Has file(s)" facet has been enabled by default. This facet shows whether items have or do not have any bitstreams in the "ORIGINAL" bundle.
Should you want to turn this off, you can edit `[dspace]/config/spring/api/discovery.xml` to remove the following line from the `defaultConfiguration` and `homepageConfiguration` beans (in the `sidebarFacets` property):

```
<ref bean="searchFilterContentInOriginalBundle"/>
```

Then restart your servlet container.

Discovery Solr Index Maintenance

DSpace 8.0 version has an issue with `-i <object handle>` command (https://github.com/DSpace/DSpace/issues/9764). Handles aren't supported as parameters in this version. For the expected result, the user should use the `-i <object UUID>` instead.
This issue will be fixed in 9.0 / 8.1 and 7.6.3 versions.

Command used:	`[dspace]/bin/dspace index-discovery [-cbhf[r <item handle>]]`
Java class:	org.dspace.discovery.IndexClient

Arguments (short and long forms):	Description
	called without any options, will update/clean an existing index
`-b`	(re)build index, wiping out current one if it exists
`-c`	clean existing index removing any documents that no longer exist in the db
`-f`	if updating existing index, force each handle to be reindexed even if uptodate
`-h`	print this help message
`-i <object handle>`	Reindex an individual object (and any child objects). When run on an Item, it just reindexes that single Item. When run on a Collection, it reindexes the Collection itself and all Items in that Collection. When run on a Community, it reindexes the Community itself and all sub-Communities, contained Collections and contained Items.
`-r <object handle>`	remove an Item, Collection or Community from index based on its handle
-s	Rebuild the spellchecker, can be combined with -b and -f.
`-t <object type>`	Only index objects of a specific type (e.g. Collection, Community, Item, ClaimedTask, PoolTask, XmlWorkflowItem, WorkspaceItem). May be combined with other options. For example when combined with "-f" you can force reindex only Items ("-f -t Item").

It is recommended to run maintenance on the Discovery Solr index occasionally (from crontab or your system's scheduler), to prevent your servlet container from running out of memory:

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

Advanced Solr Configuration

Discovery is built as an application layer on top of the Solr open source enterprise search server. Therefore, Solr configuration can be applied to the Solr cores that are shipped with DSpace.
The DSpace Solr instance currently runs several cores (which means indexes in Solr parlance). The "statistics" core is for collection of DSpace usage events for statistical purposes (if you have been collecting statistics for multiple years, you may have chosen to use sharding and you will see one core per each year collected). The "search" core is used by Discovery for for search and faceting, for displaying the collection/community hierarchy and item counts. The "authority" core is used by SolrAuthority to store information about authors, including their data imported from the ORCID registry.

```
solr
 search
    conf
        protwords.txt
        schema.xml
        solrconfig.xml
        stopwords.txt
        synonyms.txt
|
 ...
 statistics
     conf
         protwords.txt
         schema.xml
         solrconfig.xml
         stopwords.txt
         synonyms.txt
```

IIIF Configuration

- Overview
  - Format Support
- Enable IIIF Support on Backend
  - Install a IIIF Image Server
    - Installing and Configuring Cantaloupe
  - Required IIIF Configuration
  - Additional Configuration Options
  - CORS Configuration
  - IIIF Search API
- Enable/Install the Mirador Viewer on Frontend
  - Configuring Mirador
- Configure IIIF viewer via Metadata Fields

Overview

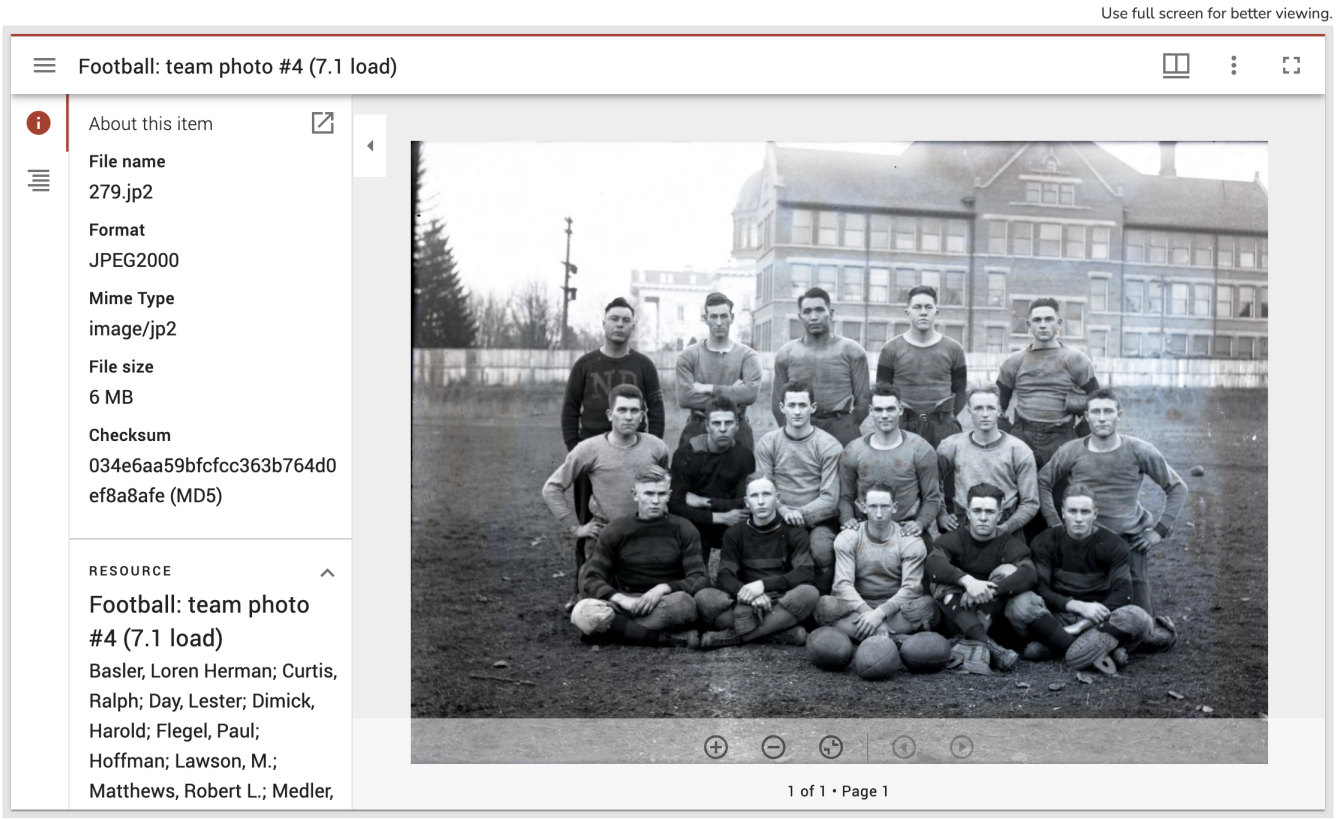
DSpace supports the International Image Interoperability Framework (IIIF). The DSpace REST API implements the IIIF Presentation API version 2.1.1, IIIF Image API version 2.1.1, and the IIIF Search API version 1.0 (experimental). The DSpace Angular frontend uses the Mirador 3.0 viewer.

Administrators can configure IIIF behavior at the Collection, Item, Bundle and Bitstream levels using metadata. To support additional sharing, viewing, comparing, and annotating, DSpace can be configured to share IIIF metadata with external IIIF clients (see CORS Configuration). IIIF REST endpoints implement the same security protocol as the primary REST API so that DSpace authorization policies are enforced for IIIF access as well.

IIIF Image Server

Running IIIF in production requires an IIIF-compatible image server. You are free to use any compatible image server you choose. However, instructions for configuring the Cantaloupe Image Server are included below. A preconfigured Cantaloupe image server can be started via docker-compose to simplify evaluation and testing.

Use full screen for better viewing.



Format Support

Currently, DSpace only supports IIIF viewing of Image formats (any format whose MIME type starts with "image/*"). For example, PDF viewing is not currently supported.

Enable IIIF Support on Backend

DSpace IIIF support is not enabled by default. To enable IIIF, you first need to install a IIIF Image Server, and then update your DSpace configuration as described below.

Install a IIIF Image Server

The Cantaloupe Image Server is currently recommended for use with DSpace, but you are free to use the image server of your choice. A list of IIIF-compliant image servers is maintained by the IIIF community.

Here is a brief overview of how the IIIF image server works with DSpace.

First, the base path to the image server is defined in config/modules/iiif.cfg.

```
iiif.image.server = https://imageserver.mycampus.edu/image-server/cantaloupe/iiif/2/
```

Given this configuration, the IIIF `manifest` returned by the DSpace backend will include an image resource annotation like the following:

IIIF Image Resource Annotation

```
resource: {
        @id: "https://imageserver.mycampus.edu/image-server/cantaloupe/iiif/2/4b415036-57a8-42f4-a971-
c5e982f55f92/full/full/0/default.jpg",
        @type: "dctypes:Image",
        service: {
                @context: "http://iiif.io/api/image/2/context.json",
                @id: "https://imageserver.mycampus.edu/image-server/cantaloupe/iiif/2/4b415036-57a8-42f4-a971-
c5e982f55f92",
                profile: "http://iiif.io/api/image/2/level1.json",
                protocol: "http://iiif.io/api/image"
        },
        format: "image/jp2"
}
```

The Mirador viewer (see below) uses this annotation to communicate with the image server using the IIIF Image API.

Finally, notice that the image server needs to retrieve the requested bitstream from DSpace. There are a number of ways to do this and the details vary with the image server chosen. The easiest approach is for the image server to request the bitstream via HTTP and the DSpace API, e.g.:

```
 http:/dspace.mycampus.edu:8080/server/api/core/bitstreams/4b415036-57a8-42f4-a971-c5e982f55f92/content
```

Installing and Configuring Cantaloupe

The Cantaloupe getting started page provides installation instructions. The basic installation process is simple.

The simplest way to configure Cantaloupe to retrieve images from DSpace is to use HTTPSource with the following configuration.

```
HttpSource.BasicLookupStrategy.url_prefix = <dspace-url>/server/api/core/bitstreams/
HttpSource.BasicLookupStrategy.url_suffix = /content
```

Required IIIF Configuration

To enable IIIF, edit `config/modules/iiif.cfg` or your `local.cfg` file and set **`iiif.enabled`** to be "true".

```
iiif.enabled = true
```

In addition, you need to provide the URL for your newly installed IIIF image server. e.g.:

```
iiif.image.server = http://localhost:8182/iiif/2/
```

Finally, update `dspace.cfg` or your `local.cfg` file by adding "iiif" to the default event dispatcher, as shown below:

```
event.dispatcher.default.consumers = versioning, discovery, eperson, iiif
```

With these changes in place, DSpace will be ready to respond to IIIF requests. Restart your DSpace backend (i.e. Tomcat) for these changes to all take effect.

Additional Configuration Options

The full set if IIIF configuration options can be found in config/modules/iiif.cfg.

Property	Description
iiif.enabled	Enables the DSpace IIIF service.
iiif.image.server	Base URL path for the IIIF image server. e.g. http://localhost:8182/iiif/2/
iiif.document.viewing.hint	Default viewing hint. Can be overridden with the metadata setting described below.
iiif.logo.image	Optional URL for a small image. This will be included in all IIIF manifests.
iiif.cors.allowed-origins	Comma separated list of allowed CORS origins. The list must include the default value: ${dspace.ui.url}.
iiif.metadata.item	Sets the Dublin Core metadata that will be added to the IIIF resource manifest. This property can be repeated.
iiif.metadata.bitstream	Sets the Bitstream metadata that will be added to the IIIF canvas metadata for individual images. This property can be repeated.
iiif.license.uri	Sets the metadata used for information about the resource usage rights.
iiif.attribution	The text to use as attribution in the iiif manifests. Defaults to: ${dspace.name}
iiif.document.viewing.hint	Either "individuals", "paged" or "continuous". Can be overridden with the metadata setting described below.
iiif.canvas.default-width	Default value for the canvas size. Can be overridden at the item, bundle or bitstream level.
iiif.canvas.default-height	Default value for the canvas size. Can be overridden at the item, bundle or bitstream level.

Canvas Dimensions

As of 7.2, the canvas dimension options (*iiif.canvas.default-width* and *iiif.canvas.default-height*) are updated with additional behaviors.

- If you do not provide your own default dimensions in `iiif.cfg`, DSpace will attempt to optimize canvas dimensions when dimension metadata is missing from the first bitstream in the item. This will often produce more accurate viewer layouts, but note that it is not sufficient to assure accurate layouts in all cases.
- If you decide to add your own default dimensions in `iiif.cfg` file your dimensions are used for every bitstream that lacks dimension metadata.
- You may also set both default dimensions in `iiif.cfg` to the value `-1`. In this case, DSpace creates accurate default dimensions for every bitstream that lacks dimension metadata. Note that this impacts performance.

It is recommended that *iiif.image.width* and *iiif.image.height* metadata be added to Item, Bundle, or Bitstream metadata to assure accurate layout and top performance. Default dimension configurations are intended to improve the user experience when dimension metadata has not yet been added.

CORS Configuration

The wildcard "*" configuration is the default CORS setting for IIIF. With this setting, all remote viewers and applications can retrieve manifests, assuring maximum interoperability. You can restrict CORS origins using the `iiif.cors.allowed-origins` property defined in `iiif.cfg`. Remove the wildcard and add a comma-separated list of origins instead.

IIIF Search API

DSpace includes a plugin to support the IIIF Search API. This plugin is designed to work specifically with the Solr OCR Highlighting Plugin and METS /ALTO data. You are welcome to experiment with the plugin. To do so,uncomment the following settings in `config/modules/iiif.cfg`:

```
iiif.search.url = ${solr.server}/word_highlighting
iiif.search.plugin = org.dspace.app.rest.iiif.service.WordHighlightSolrSearch
```

Once you have successfully indexed ALTO files using the Solr plugin, you can enable search within a DSpace Item by adding the `iiif.search.enabled metadata` field.

Indexing Support

Support for indexing OCR files using the the Solr OCR Highlighting Plugin or other services is not currently provided by DSpace. Institutions will need to develop their own approach to indexing their data.

Enable/Install the Mirador Viewer on Frontend

The Mirador 3.0 viewer is included in the dspace-angular (UI) source code. Before enabling Mirador, be sure to review the instructions for installing the Angular frontend if you haven't already.

To add the Mirador viewer to your DSpace frontend installation, run the following command:

```
# This builds and runs the DSpace UI with the Mirador Viewer in a single step
npm run start:mirador:prod
```

This will build and copy Mirador to your `dist/` directory and start the frontend server.

The actual steps for deploying the Angular UI with Mirador into Production will likely vary with your setup. However, one possible command-line scenario is the following:

```
# Build Mirador viewer
npm run build:mirador
# Build DSpace UI for production
npm run build:prod
# Run the DSpace UI with Mirador viewer
npm run serve:ssr
```

Running in Development

In the Dspace 7.1 release, the Mirador viewer cannot be used when running in development mode. For now, you need to use a production build.

Configuring Mirador

The Mirador viewer is highly configurable. The Mirador configuration file for DSpace includes a number of settings that you can override manually, including CSS values for styling. Note that some of the Mirador behavior (like the inclusion of thumbnail navigation on the right) is set by the Angular component at runtime. You can choose to override these runtime settings if you like.

Configure IIIF viewer via Metadata Fields

IIIF configuration at the Item-level is quite flexible and is managed using metadata. Canvas sizes, image labels, ranges and and other settings are controlled by using the following fields.

Required Field

Note that the `dspace.iiif.enabled` metadata field **MUST** be added to the Item and set to "true". Otherwise, the Item display will use the default DSpace view.

Schema	Element	Qualifier	Scope	Description
dspace	iiif	enabled	Item	Stores a boolean text value (true or false) to indicate if the iiif feature is enabled or not for the dspace object. If absent the value is derived from the parent dspace object.
iiif	label		Bitstream	Metadata field used to set the IIIF label associated with the canvas resource otherwise the system will derive one according to the configuration setting or the canvas.naming metadata field.
iiif	description		Item	Metadata field used to set the IIIF description associated with the resource.
iiif	canvas	naming	Item	Metadata field used to set the base label used to name all the canvas in the Item. The canvas label will be generated using the value of this metadata as prefix and the canvas position. e.g. Page 1, Page 2, etc.
iiif	viewing	hint	Item	Metadata field used to set the viewing hint overriding the configuration value if any. Possible values are "individuals" and "paged". Default value: individuals.
iiif	image	width	Item, Bundle, or Bitstream	Metadata field used to store the width of an image in pixels. Determines the canvas size.
iiif	image	height	Item, Bundle, or Bitstream	Metadata field used to store the height of an image in pixels. Determines the canvas size.
iiif	toc		Bitstream	Metadata field used to set the position of the iiif resource in the "table of contents" structure.
iiif	search	enabled	Item	Metadata field used to enable the IIIF Search service at the item level. This feature is experimental and requires additional setup.

Multilingual Support

DSpace supports a number of languages & you can even add your own translation. This may also be referred to as Localization (l10n) or Internationalization (i18n).

- Multilingual Support on the Backend (REST API)
  - Where to find the message catalog
  - Where to edit the message catalog
  - Localization of email messages
  - Metadata localization
  - Localization of submission-forms.xml
  - Localization of license.default
- Multilingual Support on the Frontend (UI)

Multilingual Support on the Backend (REST API)

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

Where to find the message catalog

The latest **English** message catalog is part of the main DSpace distribution and can be found at: `[dspace-source]/dspace-api/src/main/resources/Messages.properties`

The **different translations** for this message catalog are being managed separately from the DSpace core project, in order to release updates for these files more frequently than the DSpace software itself. Visit the dspace-api-lang project on Github.

Where to edit the message catalog

In some cases you may want to add additional keys to the message catalog or changing the particular wording of DSpace concepts. For example, you may want to change "Communities" into "Departments". These kind of changes may get automatically overwritten again when you upgrade to the newest version of DSpace. It is therefore advised to keep such changes isolated in the following location: `[dspace-source]/dspace/modules/server/src/main/resources/Messages.properties`

After rebuilding DSpace, any messages files placed in this directory will be automatically included in the Server web application. Files of the same name will override any default files. By default, this full directory path may not exist or may be empty. If it does not exist, you can simply create it. You can place any number of translation catalogues in this directory. To add additional translations, just add another copy of the *Messages.properties* file translated into the specific language and country variant you need.

For more information about the `[dspace-source]/dspace/modules/` directory, and how it may be used to "overlay" (or customize) the default Server Webapp, classes and files, please see: Advanced Customisation

Localization of email messages

All email templates used by DSpace can be found in `[dspace]/config/emails/`

The contents of the emails can be edited and translated.

Metadata localization

DSpace associates each metadata field value with a language code (though it may be left empty, e.g. for numeric values).

Localization of submission-forms.xml

The display labels for submission-forms.xml are currently not managed in the messages catalogs. To localize this file, you can create versions of this file in the same folders, appending _COUNTRY at the end of the filename, before the extension. For example, `submission-forms_de.xml` can be used to translate the submission form labels in German.

There is a known bug that any translated submission forms (e.g. submission-forms_de.xml) must include all the form-definitions available in the system. When they are not all included, DSpace will fall back to the default submission forms / locale. See https://github.com/DSpace/DSpace/issues/2827

Localization of license.default

The text in the default submission license (license.default) is currently not managed in the messages catalogs. It is translatable by appending _COUNTRY at the end of the filename, before the extension like for the localization of the input-forms.xml.

Multilingual Support on the Frontend (UI)

By default, DSpace will look at the user's browser language. If it has a language file in the user's language, it will render the interface in that language. If not, it will default to English or another default that you have configured.

The User Interface translations can be found in the `/src/assets/i18n/` folder of your UI's codebase. You can add additional translations & contribute them back to the project. For details see DSpace 7 Translation - Internationalization (i18n) - Localization (l10n)

All translations of the UI are provided in JSON5 format, which includes support for inline comments.

You can choose which languages you wish to enable/support in your UI by modifying the language section of your config.prod.yml file, (view sample list at config.example.yml) which in turn, will generate a section like this in your `environment.prod.ts` configuration file:

```
// Default Language in which the UI will be rendered if the user's browser language is not an active language
defaultLanguage: 'en',
// Languages. DSpace Angular holds a message catalog for each of the following languages.
// When set to active, users will be able to switch to the use of this language in the user interface.
languages: [{
  code: 'en',
  label: 'English',
  active: true,
}, {
  code: 'de',
  label: 'Deutsch',
  active: true,
}, {
  code: 'cs',
  label: 'eština',
  active: true,
}, {
  code: 'nl',
  label: 'Nederlands',
  active: true,
}],
```

As shown above, the "defaultLanguage" is the language that your UI will use *by default* , if the user's browser has not specified a preferred language

The array of "languages" are all of the additional languages you wish to support.

- The "code" must match the prefix of a `*.json5` language file located in your `/src/assets/i18n/` folder
- The "label" is the text you want to display in the UI language selector (the globe in the header)
- The "active" setting allows you to decide whether that language appears in the UI language selector or not.

Any changes to the language settings require rebuilding & redeploying your UI.

User Interface Debugging

This page provides some basic guidelines for debugging issues that may occur in the User Interface when following the User Interface Customization guide or similar.

- How to find what the User Interface is doing when you click something
- Finding which component is generating the content on a page

How to find what the User Interface is doing when you click something

HINT: This is very similar to our Troubleshoot an error guide.

If you want to determine what action the User Interface is taking when you click a button/link, you can find that information in your browser's Developer Tools.

- Open a new browser tab and visit your homepage.
- With that browser tab open, open "Developer Tools" in your browser. This "Developer Tools" tab/window will now follow all the actions you perform in your other browser tab.
  - In Chrome, go to "More Tools Developer Tools".
  - In Firefox, go to "Web Developer Web Developer Tools".
  - In Microsoft Edge, go to "More Tools Developer Tools".
- Now, go back to your open browser tab. Browse to the page you are interested in and click that button/link.
- Back in Developer Tools, you'll see what happened when you clicked that link
  - The "Network" tab will provide information on what requests the User Interface sent to the REST API. Clicking any will give you the details of the request and the response. If any appear in "red", that means the REST API returned an error status. Clicking the red request will show you the exact Headers sent/returned and the error Response.
  - If any errors occurred, you'll also see details about them in your "Console" tab.

Finding which component is generating the content on a page

Every DSpace Angular component has a corresponding `<ds-*>` HTML-like tag. The HTML tag is called the "selector" (or CSS selector), and it is defined in the "*.component.ts" tag using the "@Component" decorator (see Angular's Component Overview for the basics). The key point to remember is that *if you can find the "<ds-* >" tag, then it is easy to determine which component generated that tag*!

So, supposing you are trying to determine which component is generating part of a DSpace page.

1. View the HTML source of the page in your browser. Search for that section of the page. (Or, right click on that part of the page and select "Inspect")
   a. For example, on the homepage view the source of the "Communities in DSpace" heading
2. Look for a parent HTML tag that begins with "ds-". This is the component selector!
   a. Continuing the example, if you view the source of the "Communities in DSpace" heading, you'll see something like this (all HTML attributes have been removed to make the example more simple):

```
<ds-top-level-community-list>
  <div>
    <h2> Communities in DSpace </h2>
    <p>Select a community to browse its collections.</p>
  </div>
</ds-top-level-community-list>
```

   b. Based on the above HTML source, you can see that the "Communities in DSpace" header/content is coming from a component who's selector is "ds-top-level-community-list"
3. Now, search the source code (./src/app/) directories for a ".component.ts" file which includes that "ds-" tag name. This can most easily be done in an IDE, but also could be done using command line tools (e.g. grep like this).
   a. Continuing the example, if you search the `./src/app/` directories for "ds-top-level-community-list" you'll find a match in the "`src/app/home-page/top-level-community-list/top-level-community-list.component.ts`" file:

```
@Component({
  selector: 'ds-top-level-community-list',
  ...
})
```

   b. This lets you know that to modify the display of that section of the page, you may need to edit either the "`top-level-community-list.component.ts`" file **or** it's corresponding HTML file at "`top-level-community-list.component.html`"
4. Once you've located the component, you can edit that component's HTML file (ending in "component.html") to change that section of the page.
   a. Keep in mind, the component's HTML file may reference other "ds-" tags! Those are other components in DSpace which you can find again by searching the "./src/app" directories for that tag.

Learning DSpace

The DSpace Community Advisory Team (DCAT) is developing this user-facing guide to DSpace 8. All are welcome to participate.

Pages

- Community and Collection management
  - 'Assigning Roles' in Collections: How to Set Up Workflows and Grant Privileges
  - Collection Management
    - Create Collection
    - Delete Collection
    - Edit Collection
  - Community Management
    - Create a Community
    - Delete Community
    - Edit Community
  - Export Metadata by Collection or Community
- Content (Item) management
  - Add item
  - Delete item
  - Edit Item
    - Authorizations (Manage access to an item)
    - Collection Mapper
    - Edit Bitstream
    - Edit Metadata
    - Edit Relationship
    - Make an item discoverable
    - Make an item non-discoverable
    - Move an Item
    - Versioned Item
    - Withdraw an item
  - Embargo an item
  - Lease an item
- DSpace Demo Quick Start
- Management sidebar
  - Administrator Reports (Beta feature)
  - COAR Notify
    - COAR Notify - Dashboard
    - COAR Notify - LDN Services
    - Use Cases - Getting two DSpace application to communicate with each other
  - Notifications
    - Publication Claim
    - Quality Assurance
      - COAR Notify Integration
      - OpenAIRE Integration
- Menus
- Registry management
  - Metadata Registry Management
- Request-a-copy
  - Approve an access request
  - Create a request
- Search - Advanced
- User management
  - Add or Manage an E-Person
  - Create or manage a user group

Videos

https://new.d2t.co/knowledge-center

For repository/DSpace administrators

Community and Collection management

Content (Item) management

User management

Registry management

Draft page not yet transferred -

- Administrative search

Pages not yet created; please feel free to add, and start writing, these pages for topics requested by the DCAT DSpace Community Advisory Team:

- Perform curation tasks
- Manage active workflow tasks

For submitters

Add item

Embargo an item

Lease an item

For DSpace users

Request-a-copy

Editing Guidelines

- Crop images before loading.
- Add a border around each image.
- Add an image as "large" if text is small.
- Avoid tables, if possible.

Transfer images from Google Doc to the Wiki

1. Download the Google Doc as an HTML file. This will include an images folder.
2. Crop the images, if needed. (GIMP, free open source; cropping instructions)
3. Upload those images to the Wiki storage area.

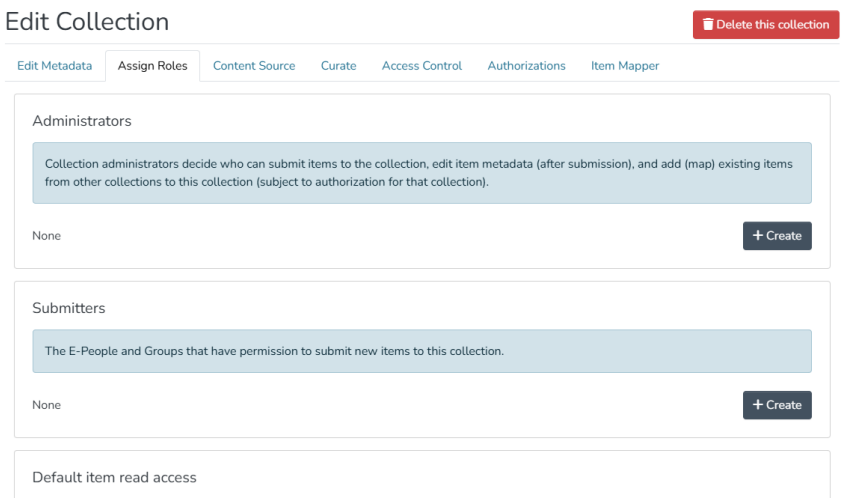
Community and Collection management

Documentation for repository managers.

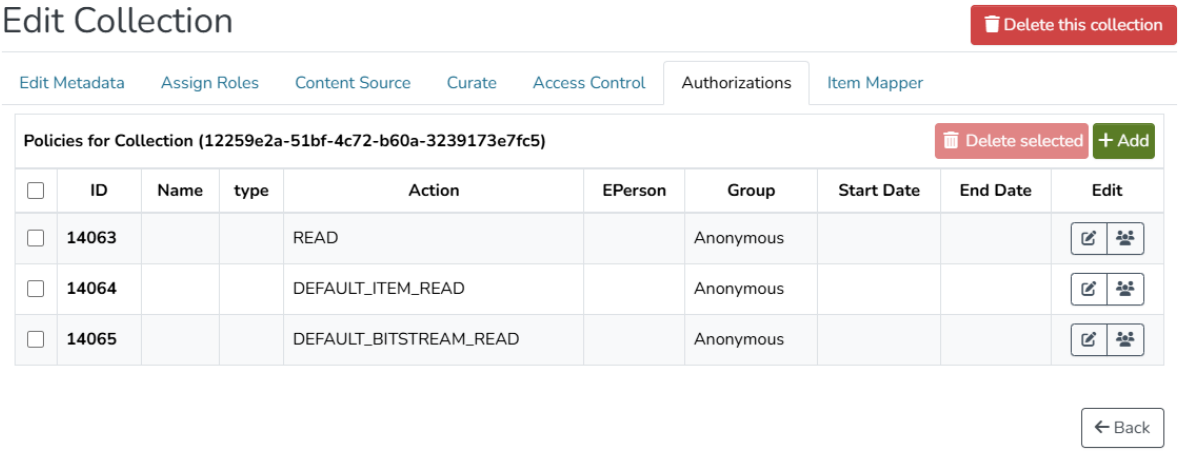
- 'Assigning Roles' in Collections: How to Set Up Workflows and Grant Privileges
- Collection Management
  - Create Collection
  - Delete Collection
  - Edit Collection
- Community Management
  - Create a Community
  - Delete Community
  - Edit Community
- Export Metadata by Collection or Community

'Assigning Roles' in Collections: How to Set Up Workflows and Grant Privileges

After creating a collection, if you would like individuals other than the repository administrator to be able to submit items or edit materials after submission, or if you want materials to be reviewed or edited by your team before they're added to the collection, you need to 'assign roles':



Assigning roles also creates a policy related to that role and the user/group to which is assigned. These policies can be viewed in the 'Authorizations' tab. No use cases have been identified by the DSpace community for why an administrator would need to add or modify policies through this section of the interface; it is generally simpler to add policies by assigning roles. Below are the default authorization policies for a collection:



There are two types of roles: (1) those which grant privileges within the collection, and (2) those which set up the workflow for the collection. These are reviewed below.

(1) Roles granting privileges within a collection:

1.Administrator

The DSpace interface states 'Collection administrators decide who can submit items to the collection, edit item metadata (after submission), and add (map) existing items from other collections to this collection (subject to authorization for that collection).'

Implications:

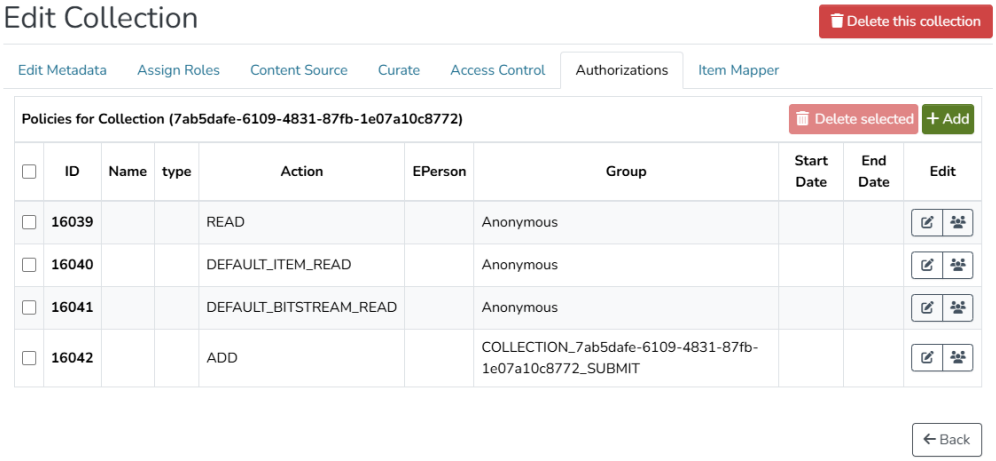
- If you give an individual administrator privileges on a collection, they can submit materials to the collection, delete the collection, withdraw materials, change embargo dates, change the workflow on their own, and assign other users to be collection administrators
- It is redundant to give a top-level administrator a role as a collection admin; they already have these privileges via their current role
- Administrators, however, will not have materials submitted to the collection appear in their 'Workflow Tasks' in 'My DSpace'; this only occurs for individuals assigned to a role of the second type.
- This role may also be assigned for communities

2. Submitters

The DSpace interface states that Submitters are 'the E-People and Groups that have permission to submit new items to this collection.' By default, in a typical DSpace installation, only top-level Administrators and collection administrators should be able to submit to a collection.

Implications:

- If you would like to allow for anonymous submissions to a collection, you must 'Create' a group under submitters, and then add the 'Anonymous' subgroup to that group.
- Creating a submitters group adds an 'ADD' policy under 'Authorizations'; this is however not information you will likely ever need to use:



3. Default item read access

The DSpace interface states that those with this role are the 'E-People and Groups that can read new items submitted to this collection. Changes to this role are not retroactive. Existing items in the system will still be viewable by those who had read access at the time of their addition.' By default, this role is set to anonymous.

4. Default item bitstream read access

E-People and Groups that can read new bitstreams submitted to this collection. Changes to this role are not retroactive. Existing bitstreams in the system will still be viewable by those who had read access at the time of their addition.

(2) Roles setting up workflow for a collection

- The number of roles configured here determine the number of steps in your collection workflow. For example, assigning a populated group to Reviewers, Editors, and Final Editors creates a three-step approval process.
- You cannot review or edit materials you submit to a collection, either by yourself or while impersonating another user. They will not appear in your workflow.
- Neither editors nor final editors can edit the metadata of a submission AFTER it has been accepted.

1.Reviewers

The DSpace interface states that 'Reviewers are able to accept or reject incoming submissions. However, they are not able to edit the submission's metadata.'

2. Editors

'Editors are able to edit the metadata of incoming submissions, and then accept or reject them.'

3. Final Editors

'Final editors are able to edit the metadata of incoming submissions, but will not be able to reject them.'

Collection Management

The collection is a level within a community or sub-community that holds items. This document provides an overview of creating, editing, and deleting a collection.

The documentation below assumes that the user has the relevant authorizations. For example, the admin menu and edit buttons would appear to a user having collection administration permission.

If you’re unsure about collection administration permissions assigned to your account for the target community, contact your system administrator.

- Create Collection
- Delete Collection
- Edit Collection

Create Collection

- Scope and Use Case
- Audience
- Create Collection

Scope and Use Case

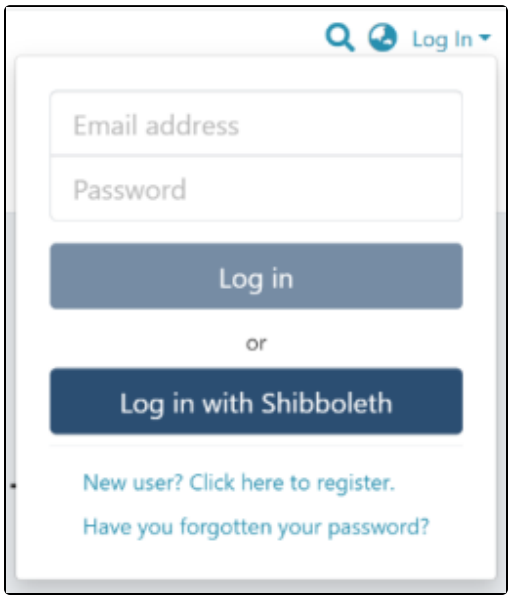
DSpace uses a hierarchy of Communities, Collections, and Items. Collections are created within Communities and contain Items. Use these instructions to create a new Collection within an existing Community. Once the Collection exists, Items can be added to it.

Audience

- Repository Administrator
- Community Administrator

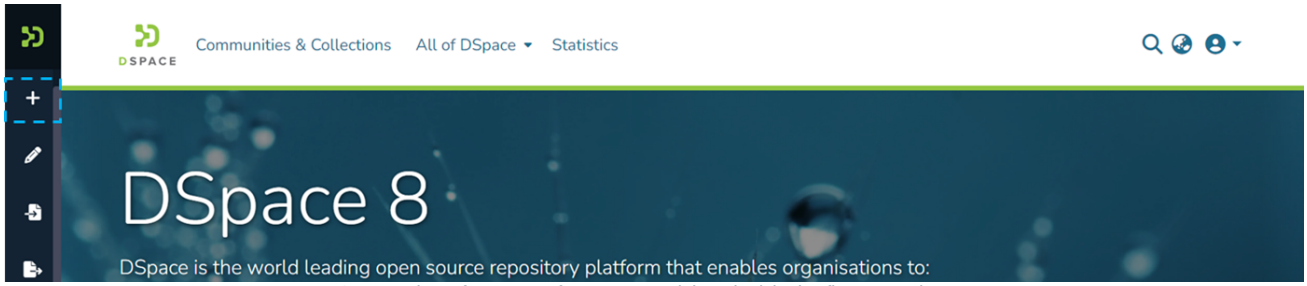
Create Collection

Step 1: Login using your credentials



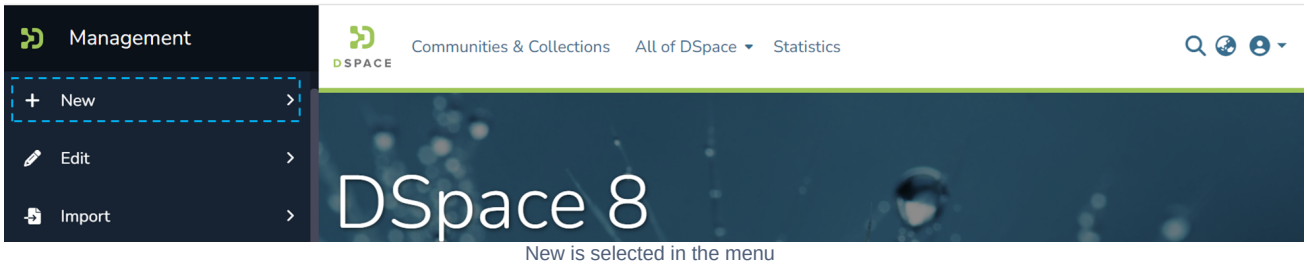
Login window

Step 2: Move your cursor over the “+” sign

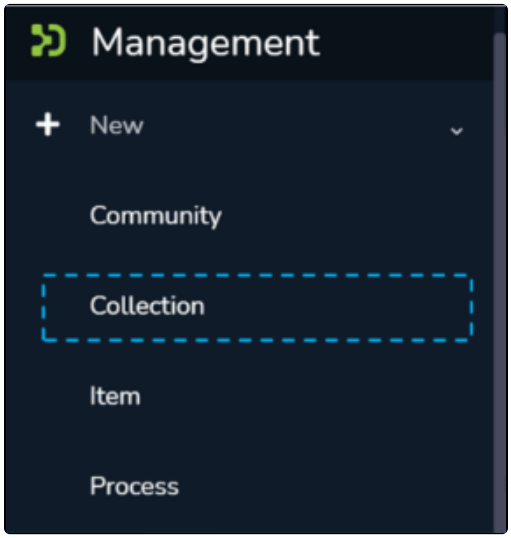


Screen shot of top part of menu. New (plus sign) is the first menu item.

Step 3: Click on “New” and click on “Collection” to proceed with the collection creation process.

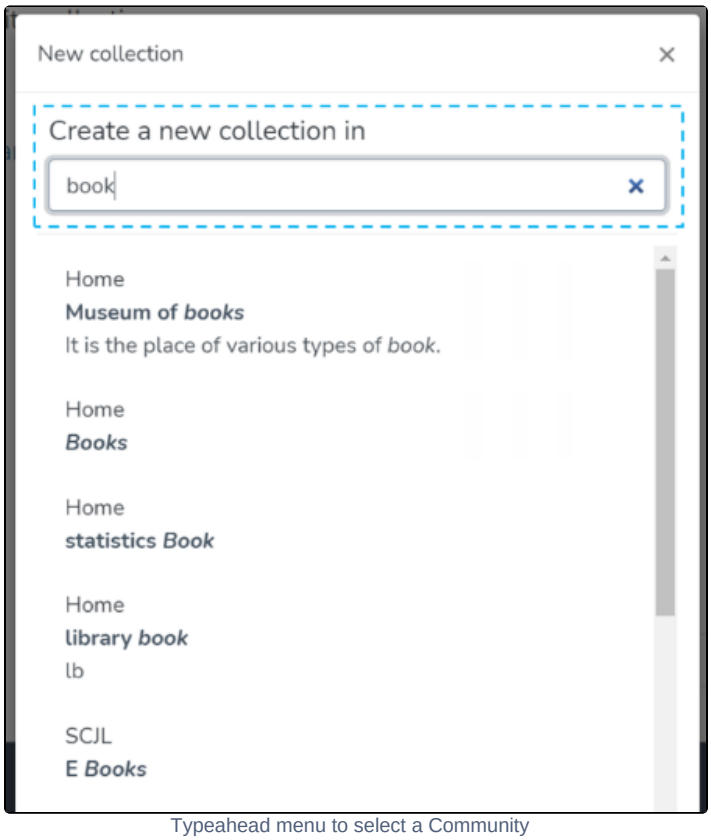


New is selected in the menu



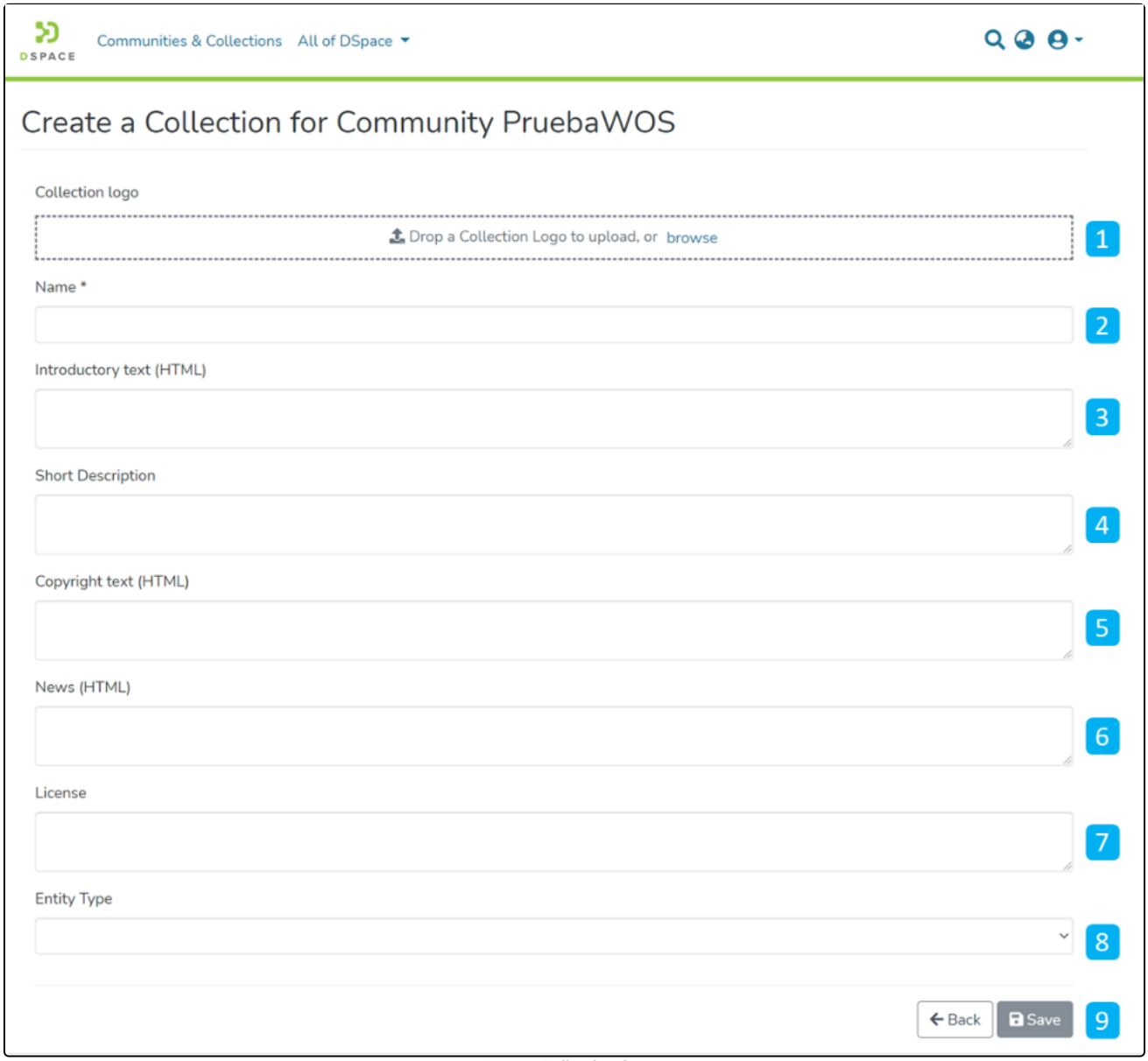
Collection is selected in the dropdown menu

Step 4: A pop-up window showing a list of communities will appear. Type the community's name in the search field where you want to add this collection. Upon typing a few letters of the community's name, a list of communities having those letters or words will appear. Click on the community name to initiate collection creation.



Typeahead menu to select a Community

Step 5: The application will take the user to the create collection form to populate information regarding the collection. Please find below the description of this form.

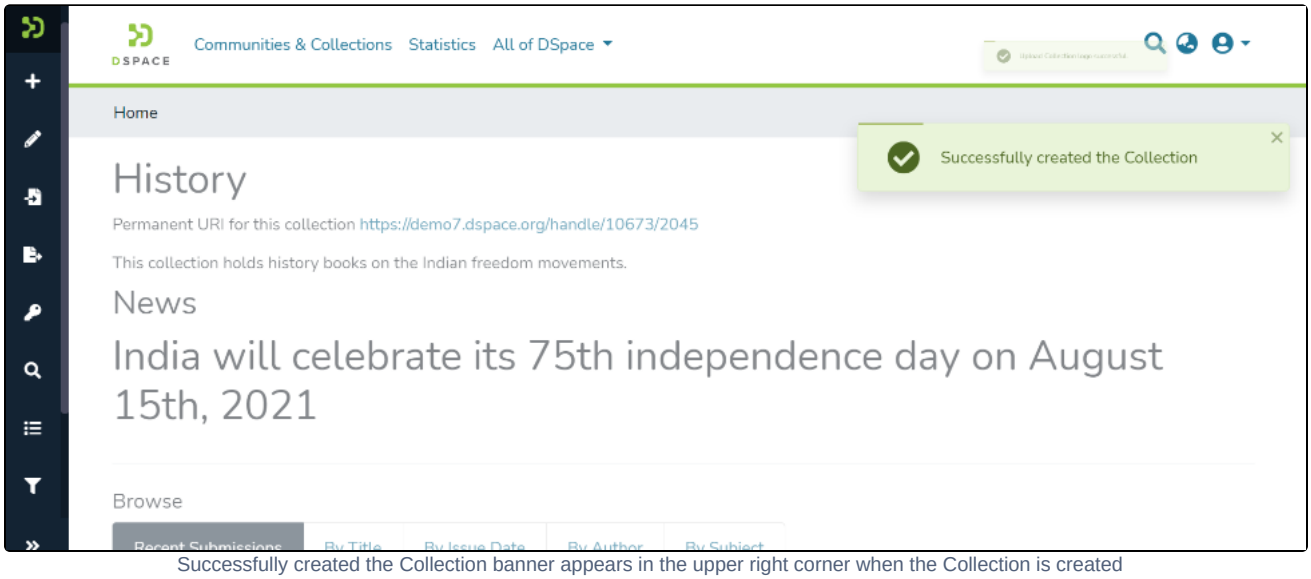


Create Collection form

Fields marked with (HTML) support HTML tags-based formatting.

1. *Collection logo – Click on the 'browse' link to select an image file user wants to add as the collection's logo. It is advisable to maintain uniform dimensions of the logo across the repository.*
2. *Name – Enter the collection's name here. It is the only mandatory field, as indicated by the '*.'*
3. *Introductory text (HTML) – Users can enter introductory text providing an overview of the contents stored in this collection. Users can utilize HTML tags to format the text or continue entering the plain text content.*
4. *Short Description – This field can have a one-liner description of the collection that appears with the collection name on the community homepage.*
5. *Copyright text (HTML) – Users can enter copyrights related information here. Users can utilize HTML tags to format the text or continue entering the plain text content.*
6. *News (HTML) – Enter news about this collection. Users can update this by going to this section via edit collection regularly. Users can utilize HTML tags to format the text or continue entering the plain text content.*
7. *License – Add license-related information here.*
8. *Entity Type – Select Entity from the drop-down that must be uploaded in the collection.*
9. *Action Buttons – Users can click on the appropriate button as desired. Clicking on the Save button will add the collection to the repository.*

Step 6: Click on the ‘Save’ button to complete the Collection creation process. A success prompt will pop up upon collection creation, and the application will automatically open the collection homepage.



Successfully created the Collection banner appears in the upper right corner when the Collection is created

Success prompt upon collection creation



Example collection homepage

Collection homepage

Edited and approved by DCAT Documentation Working Group, November 2024.

Delete Collection

- Scope and Use Case
- Audience
- Delete Collection

Scope and Use Case

DSpace uses a hierarchy of Communities, Collections, and Items. Collections are created within Communities and contain Items.

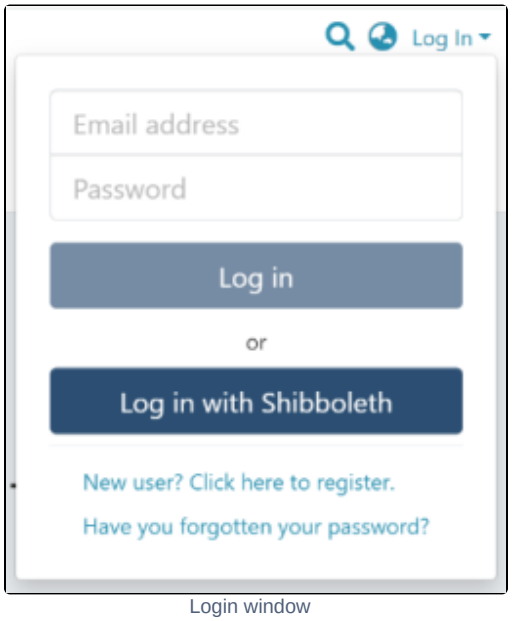
Use these instructions to delete an entire Collection. **This will also delete any Items that are contained within the Collection,** even if they are mapped to other Collections.

Audience

Repository Administrator
Community Administrator
Collection Administrator

Delete Collection

Step 1: Login using your credentials



Login window

Step 2: There are two ways to initiate the delete collection process. Follow Option 1 or Option 2, then proceed to Step 3.

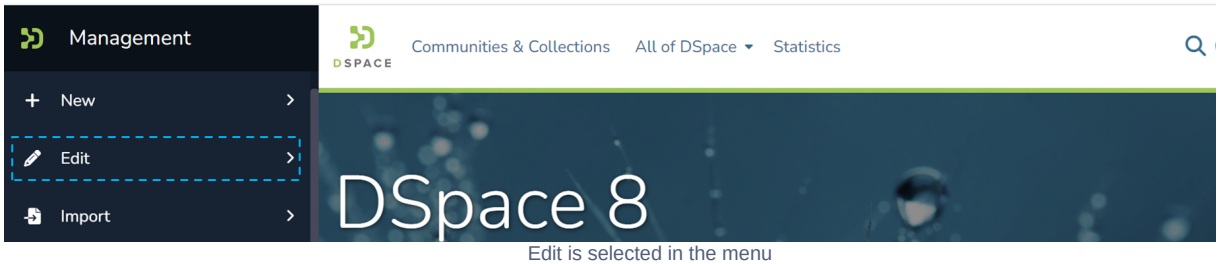
Step 2, Option 1: Go to the target collection using Admin options.

Step A: Rollover your cursor on the ”Edit“ sign.

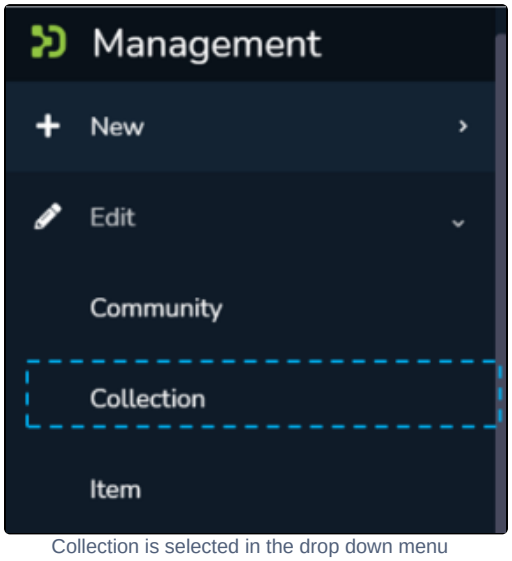


Screen shot of top part of menu. Edit (pencil icon) is the second menu item.

Step B: Click on “Edit” and click on “Collection” to proceed with the edit collection process.

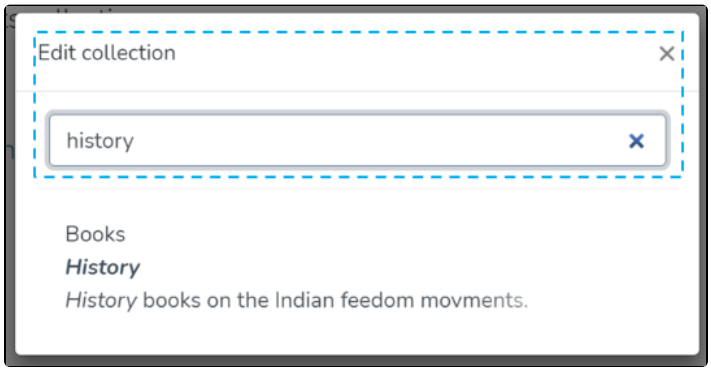


Edit is selected in the menu



Collection is selected in the drop down menu

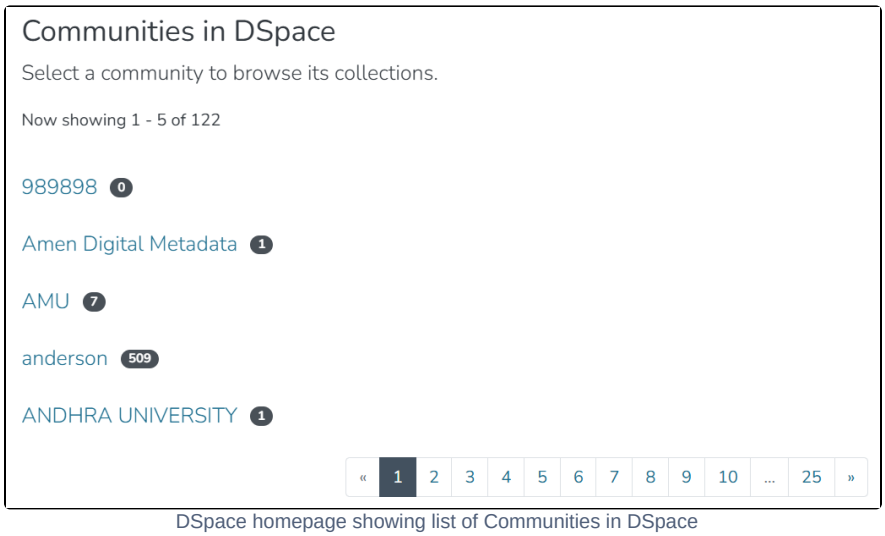
Step C: A pop-up with the list of collections will appear. Type the Collection’s name in the search field you want to delete. A list of collections having typed values will appear upon typing a few letters of the Collection’s title. Click on the Collection to continue with the deletion.



Typeahead menu to select a Collection

Step 2, Option 2: Navigate to the collection and open the edit collection process.

Step A: Navigate to the collection either by using the list of communities on the homepage, or by clicking "Communities & Collections" on the homepage and browsing the list of communities that appears.

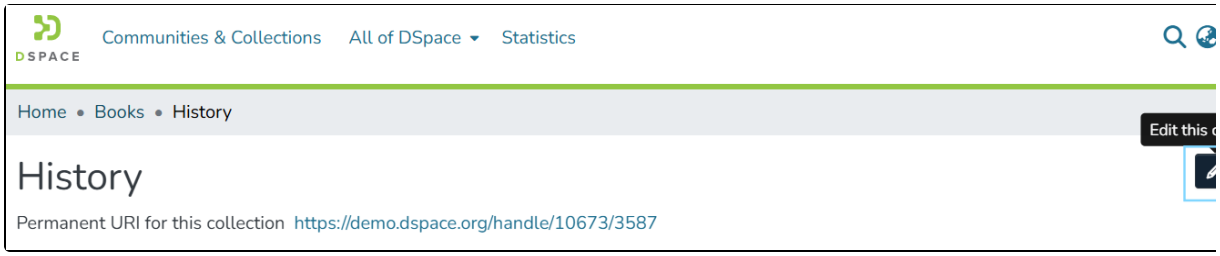


DSpace homepage showing list of Communities in DSpace



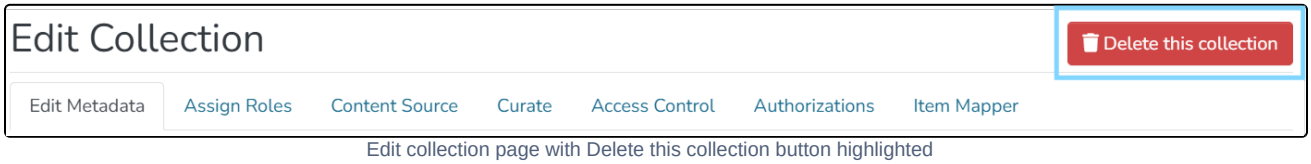
DSpace homepage with Communities & Collections button highlighted

Step B: Once on the collection page, click the "Edit this collection" icon.



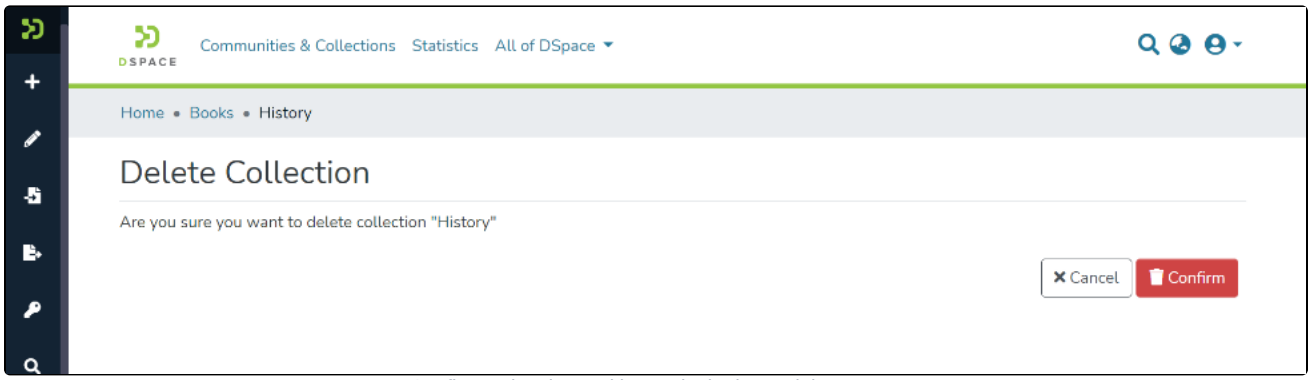
Collection with Edit this collection icon selected

Step 3: The application will take the user to the edit collection form. To initiate the collection deletion, click the ‘Delete this collection’ button.



Edit collection page with Delete this collection button highlighted

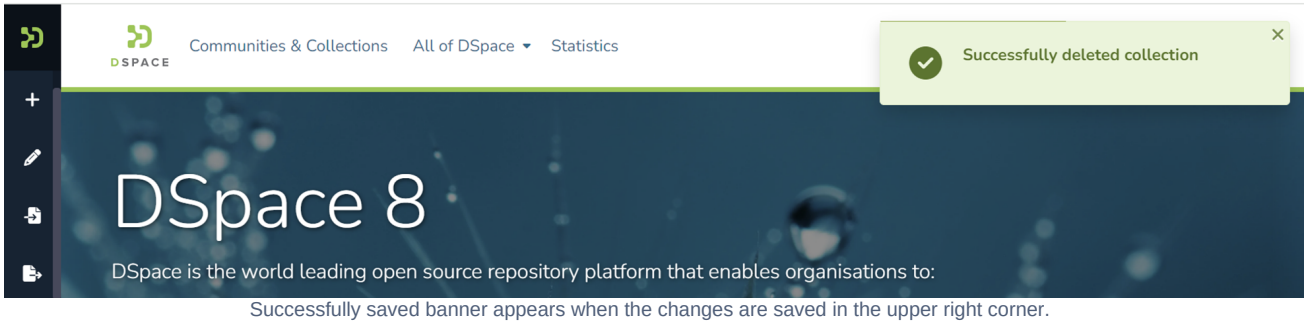
Step 6: Click the 'Confirm' button to continue with the collection deletion or click the 'Cancel' button to return to the previous page.



Confirm Delete is a red button in the lower right corner

Deleting the collection will also delete any items contained within the collection, even if they are mapped to other collections.

A success prompt confirming the deletion will appear, and the DSpace homepage will open.



Successfully saved banner appears when the changes are saved in the upper right corner.

Edited and approved by DCAT Documentation Working Group, November 2024.

Edit Collection

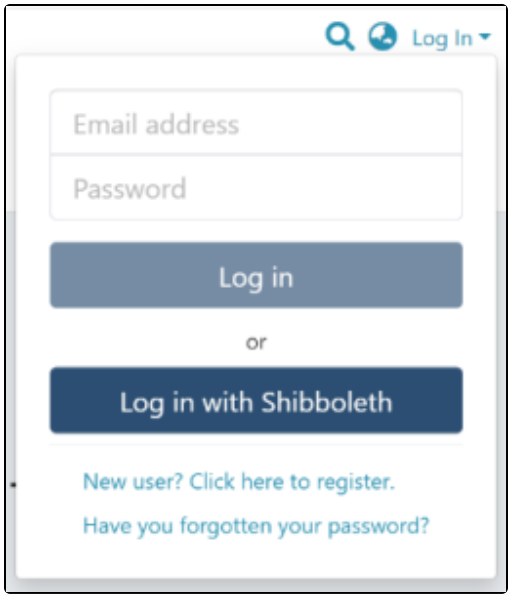
- Audience
- Edit Collection
- Edit Metadata
    - Template Item
- Assign Roles
- Content Source
- Curate
- Access Controls
- Authorizations
- Item Mapper
    - Manage mapped items
    - Map new items

Audience

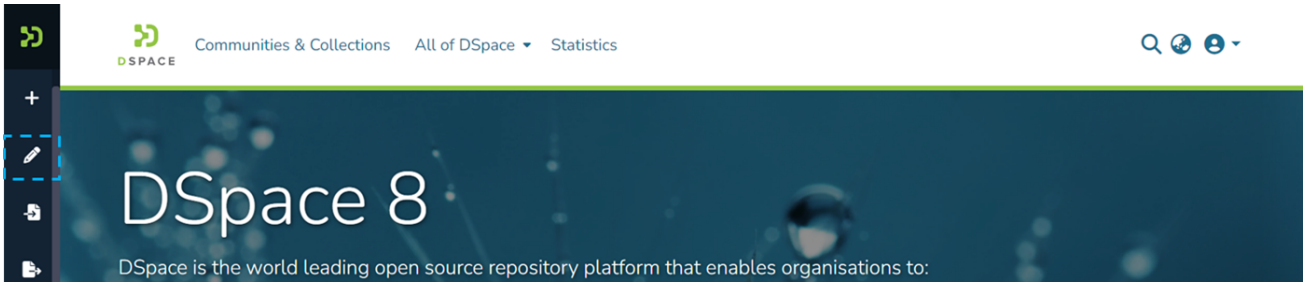
Repository Administrator
Community Administrator
Collection Administrator

Edit Collection

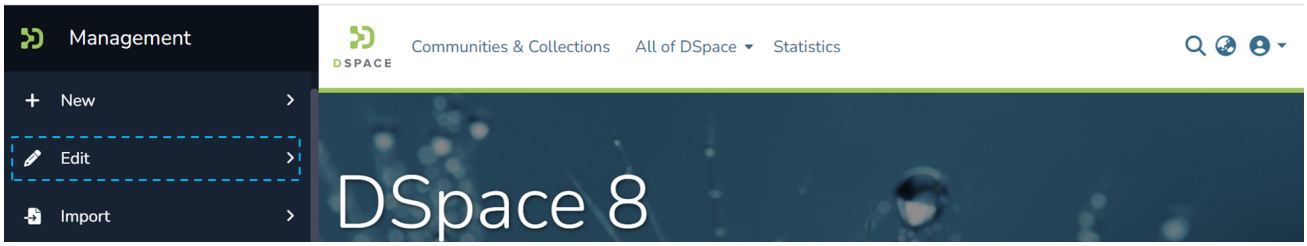
Step 1: Login using your credentials

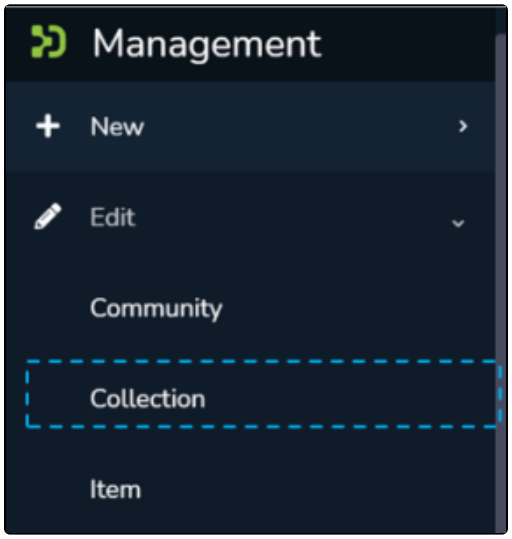


Step 2: There are multiple ways to initiate the Edit collection process. One of them is by going to the target collection using Admin options. Rollover your cursor on the "Edit" sign.

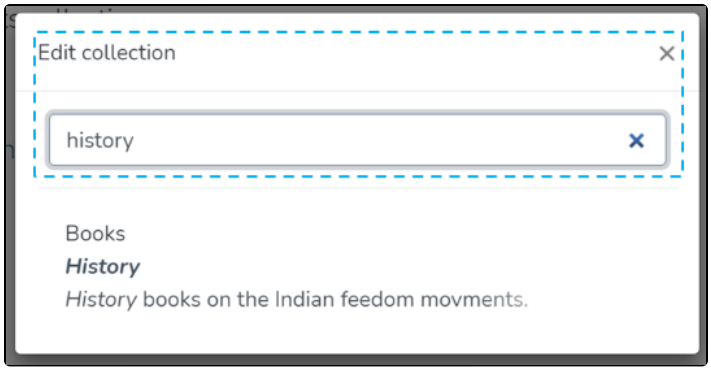


Step 3: Click on “Edit” and click on “Collection” to proceed with the edit collection process.

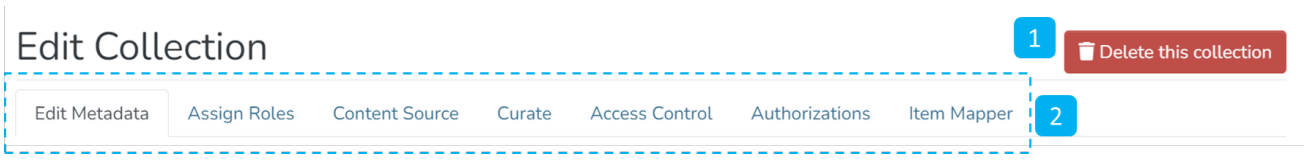




Step 4: A pop-up window showing list of collections will appear. Type the collection's name in the search field you want to edit. Upon typing a few letters of the collection's name, a list of collections with those letters or words will appear. Click on collection for initiating editing.



Step 5: The application will take the user to the edit collection form. The user can perform a range of actions to edit the collection. Each tab is explained in a separate process in this document.



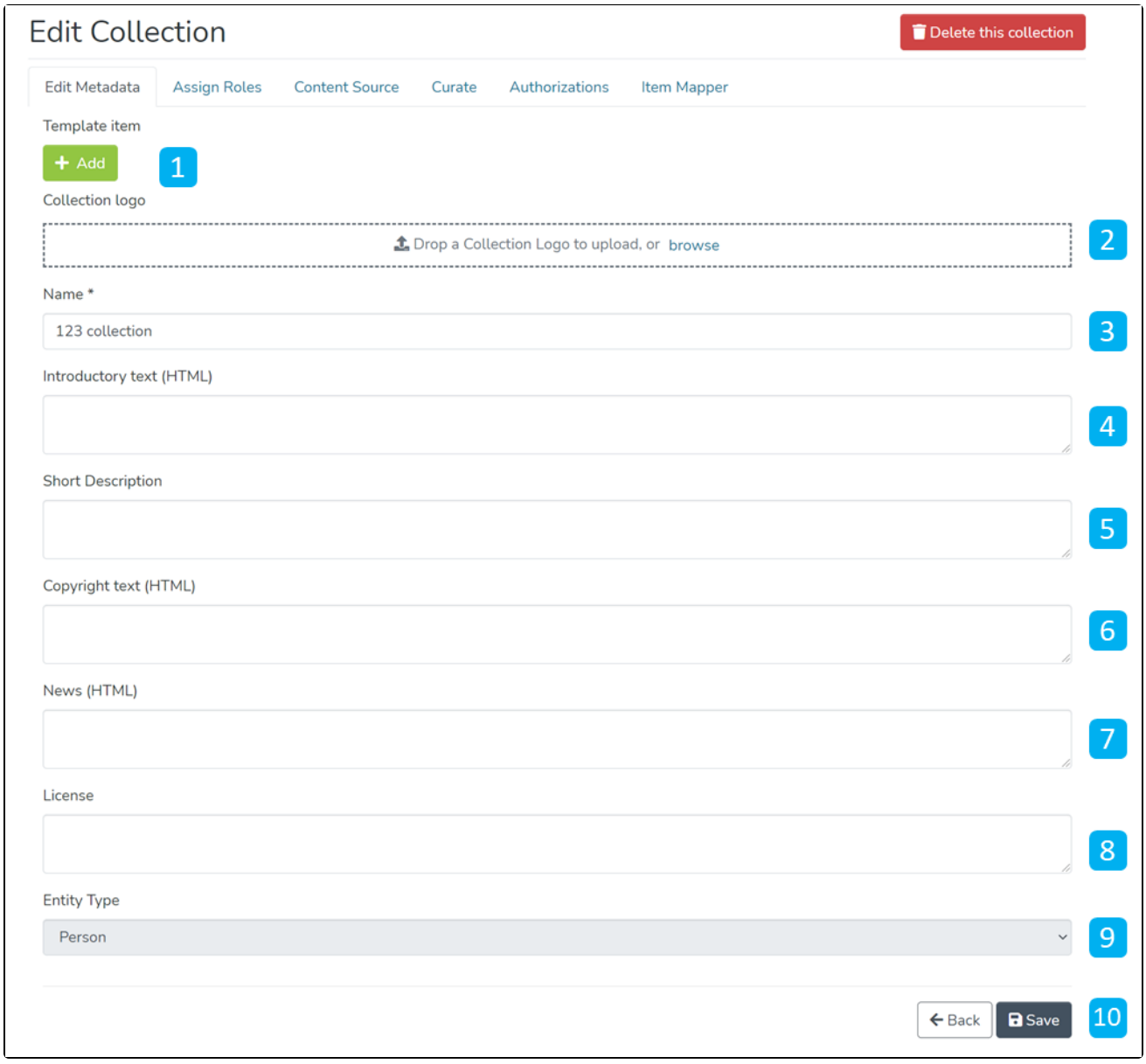
1. *Delete this collection – The button provided for deleting the collection. Detailed steps are explained in the latter part of this document.*
2. *Tabs – Edit collection has a variety of activities involved, which are grouped in a logical manner across various tabs. Below is the summary of these tabs*
    a. *Edit Metadata – Tab covers activities related to editing Collection's profile information*
    b. *Assign Roles – This tab allows users to create specific roles for the collection*
    c. *Content Source – This tab enables harvesting the contents from various sources using OAI standards*

d. *Curate – Users can set up various workflows related to content curation in this tab*
e. *Access Control - The tab allow users to perform changes to the access conditions of all the items owned by the collection. Changes may be performed to either all Item metadata or all content (bitstreams).*
f. *Authorizations – Under this tab, users can manage various groups created for managing different access rights and workflows specific to the collection*
g. *Item Mapper - The item mapper tool allows collection administrators to map items from other collections. Collection administrators can search items from other collections and map them, or browse the list of currently mapped items.*

Edit Metadata

Step 6: The Edit Metadata tab allows users to update the collection's profile-related information, a.k.a. collection Metadata.

Various actions on this tab are explained immediately after the Edit Metadata illustration is added below.

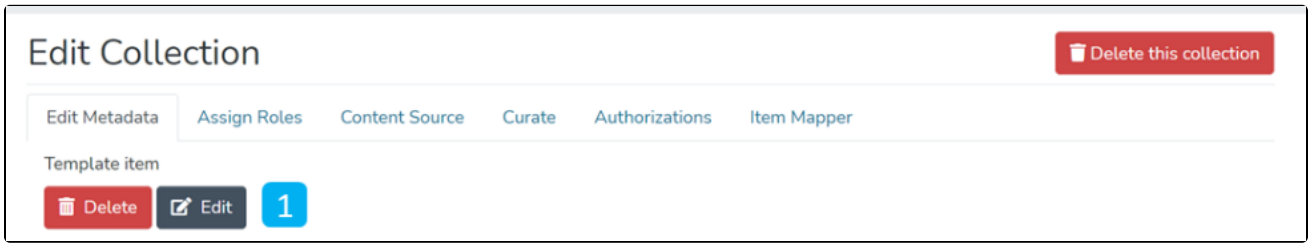


1. Template Item – Users can add metadata elements and values during item submission in this collection. Item submitters can update or delete these pre-populated values during the submission process.
2. Collection logo – Click on the delete button to remove the existing logo or add it if no logo exists.
3. Name – Update the existing collection name in this field.
4. Introduction text (HTML) – Update introductory text if already added or add new text. One can utilize HTML tags to format the text or continue entering the plain text content.
5. Short Description – Update the collection description or add a fresh short description for the collection.
6. Copyright text (HTML) – Update copyright-related information in this field. Fields marked with (HTML) support HTML tags-based formatting.

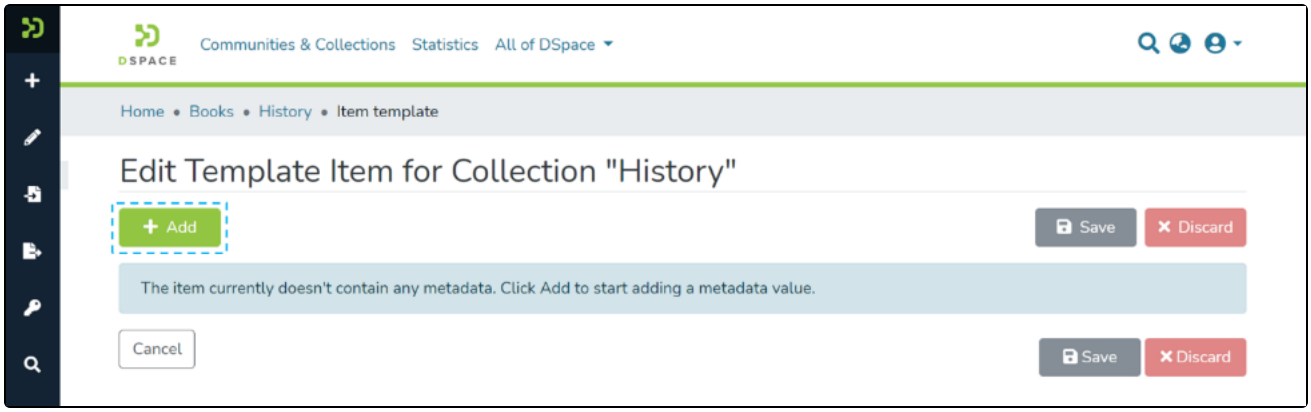
7. New (HTML) – Add/Update news specific to this collection in the field.
8. License – Add/Update license-related information here.
9. Entity Type – After adding an entity once to the collection, the value remains constant and uneditable.
10. Action Button – Clicking on the Save button will update the metadata information for the collection.

Template Item

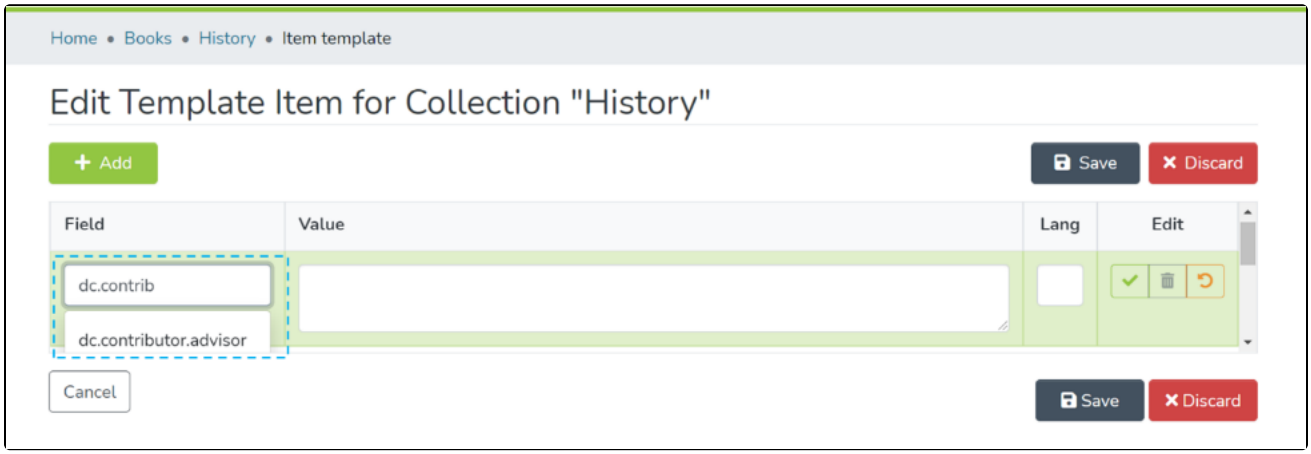
Step 7: Click the 'Edit' button under the Template Item section to add metadata elements with pre-populated values for the item submission process.



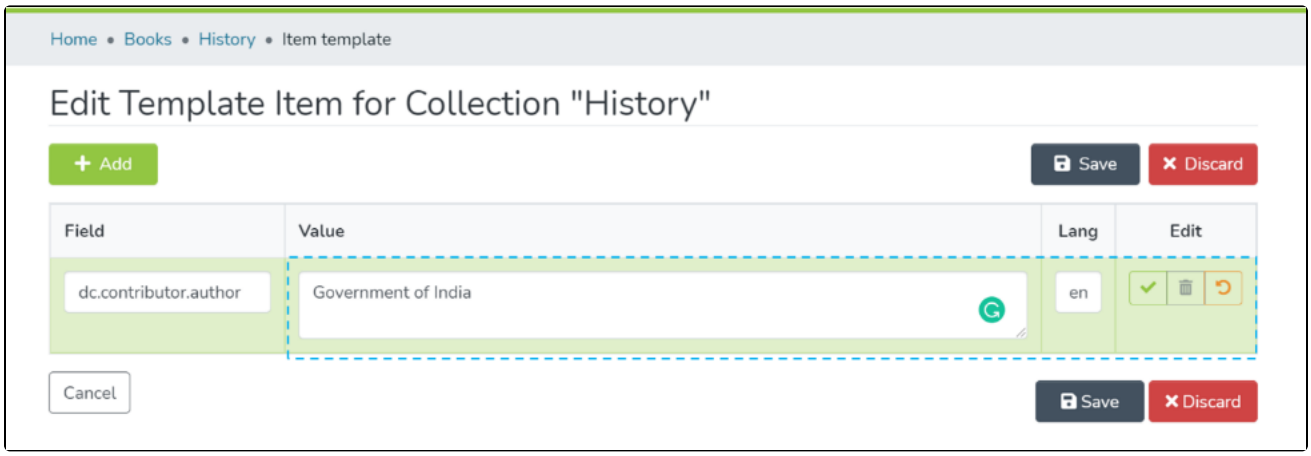
Step 8: Click on the Add button to add the metadata element.



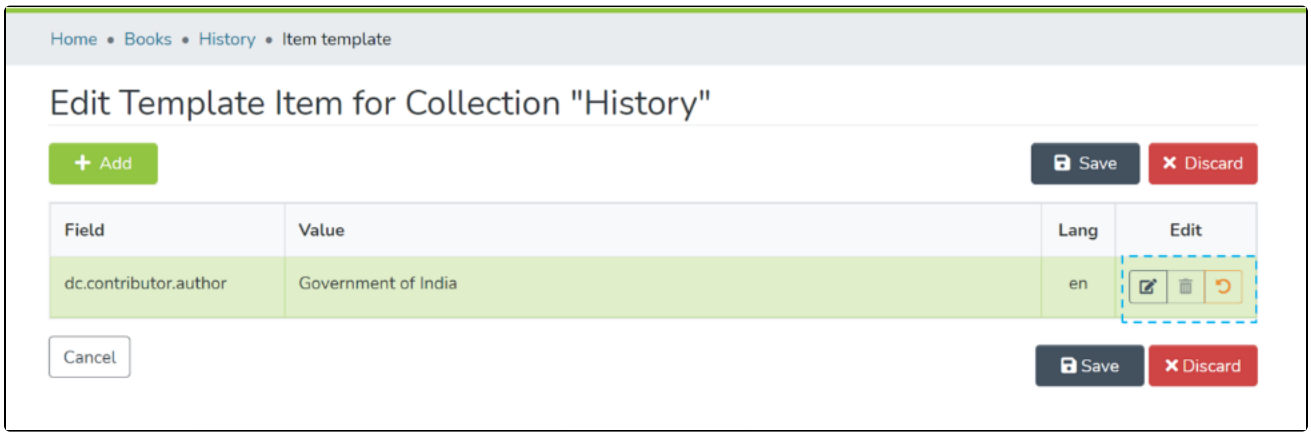
Step 9: Users can start typing metadata elements as demonstrated below and select the appropriate component from the drop-down list.



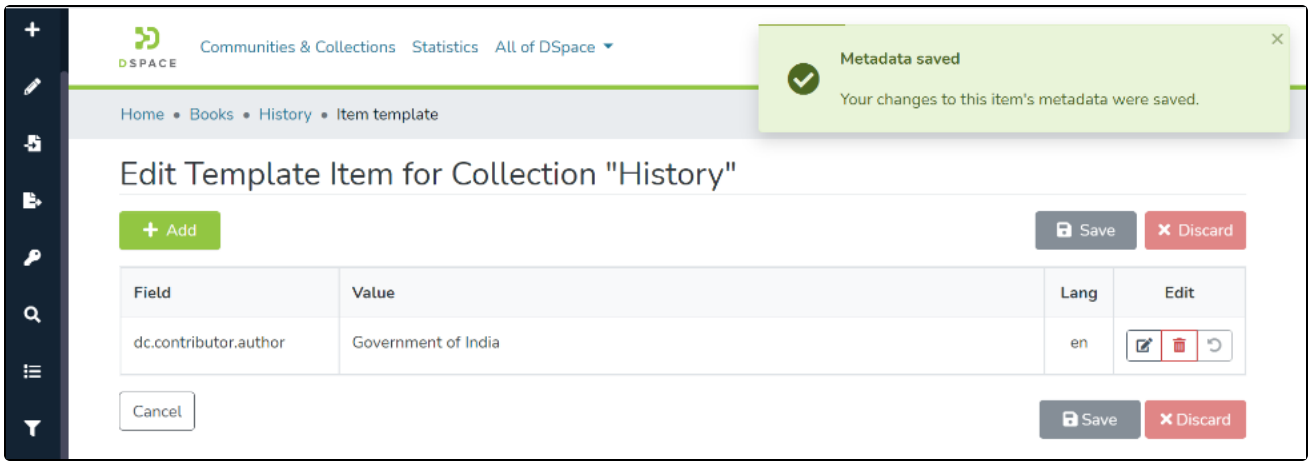
Step 10: Enter the desired value in the Value field, and enter the ISO code of the language used. Then, click on add button for adding the template metadata element.



Step 11: Users can click on the action buttons appearing to the right of the element added for updating or deleting the value-added in the element.



Step 12: Click on the Save button to finish the template edit process. A pop-up notification confirming successful updates of the metadata element will appear, as demonstrated below.



Step 13: Click on the Save button appearing at the bottom of the Edit Metadata tab to save all updates. A success prompt will appear, and the collection homepage will open.



Assign Roles

Step 14: This tab helps assign users to roles designed for the collection. These roles include administrative, maker-checker, and content consumption activities. The description for each role is provided below the screenshot.

Edit Metadata | Assign Roles | Content Source | Curate | Authorizations | Item Mapper

Administrators

Collection administrators decide who can submit items to the collection, edit item metadata (after submission), and add (map) existing items from other collections to this collection (subject to authorization for that collection).

None + Create

Submitters

The E-People and Groups that have permission to submit new items to this collection.

None + Create

Default item read access

E-People and Groups that can read new items submitted to this collection. Changes to this role are not retroactive. Existing items in the system will still be viewable by those who had read access at the time of their addition.

Default read for incoming items is currently set to Anonymous. Restrict

Default bitstream read access

Community administrators can create sub-communities or collections, and manage or assign management for those sub-communities or collections. In addition, they decide who can submit items to any sub-collections, edit item metadata (after submission), and add (map) existing items from other collections (subject to authorization).

Default read for incoming bitstreams is currently set to Anonymous. Restrict

Editors

Editors are able to edit the metadata of incoming submissions, and then accept or reject them.

None + Create

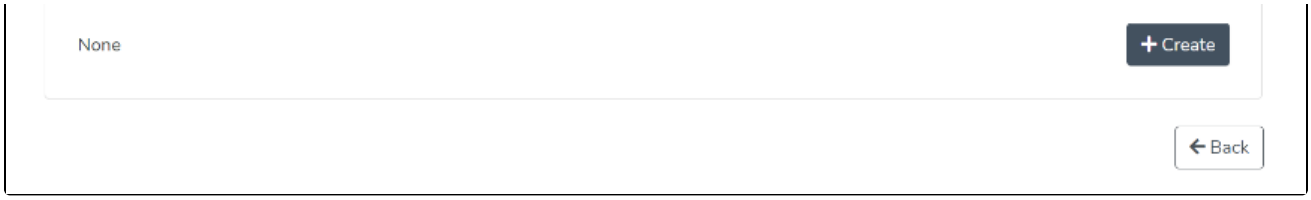
Final editors

Final editors are able to edit the metadata of incoming submissions, but will not be able to reject them.

None + Create

Reviewers

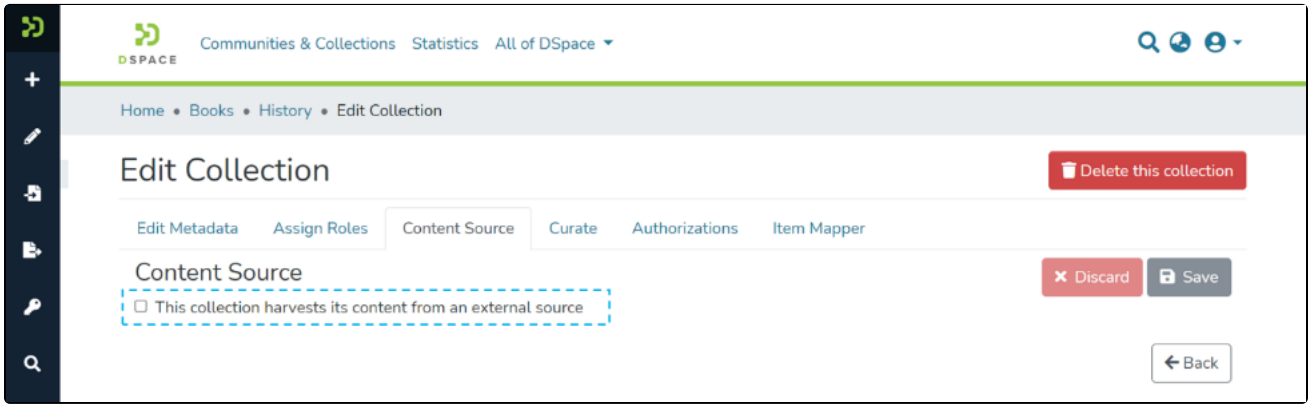
Reviewers are able to accept or reject incoming submissions. However, they are not able to edit the submission's metadata.



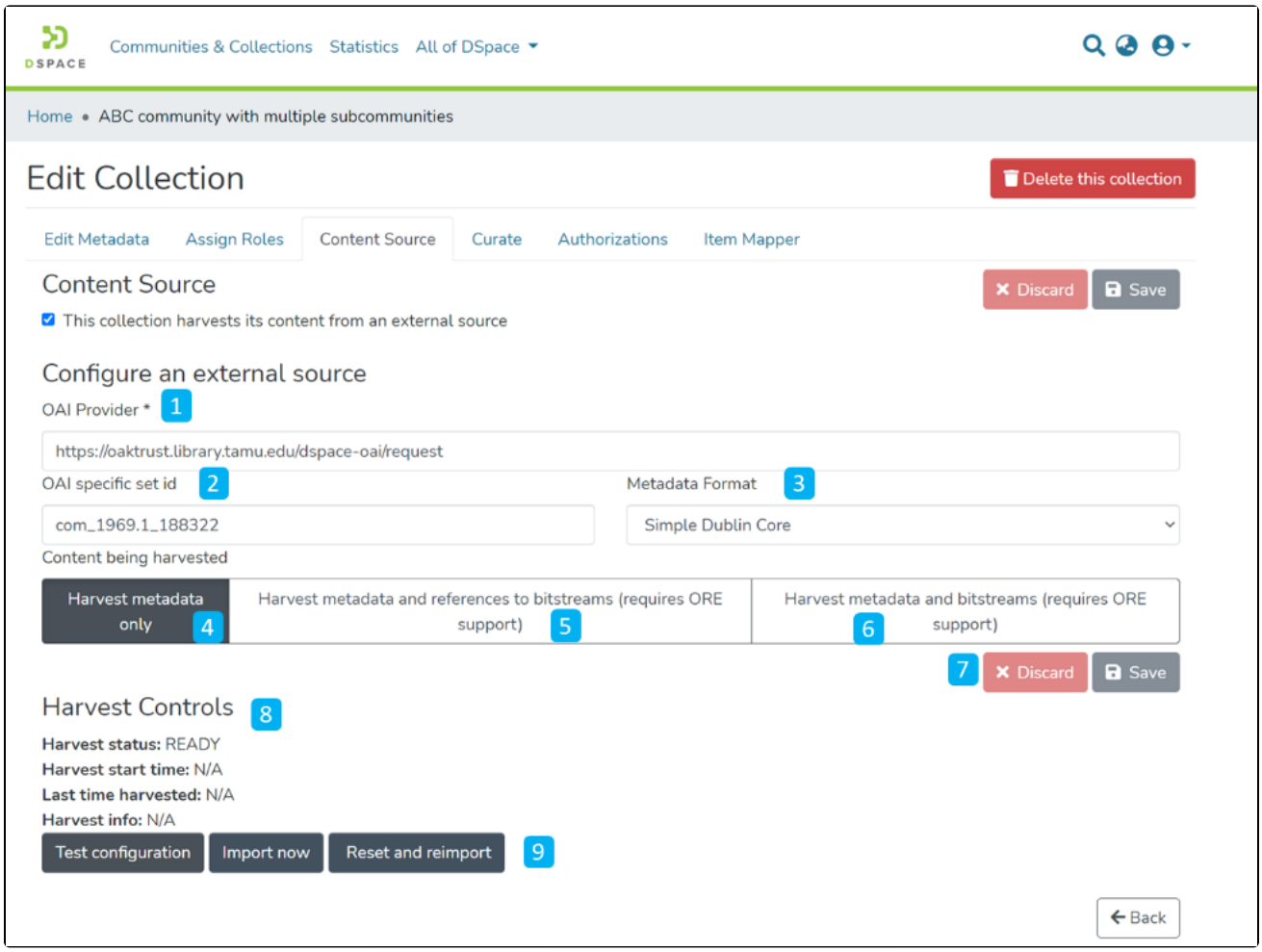
1. Administrators – The collection administrator can assign rights like item submission, edit item metadata, and map existing items from other collections to this collection. Click the create button to create a dedicated Administrator group for the collection.
2. Submitters – Users or User groups part of this group can submit items to the collection. Click on the create button to add specific users and user groups to perform item submission.
3. Default item read access - E-people and Groups can read new items submitted to this collection. Changes to this role are not retroactive. Existing items in the system will still be viewable by those who had read access at the time of their addition. Click the restrict button to restrict default item read access rights to a specific user group.
4. Default bitstream read access – E-People and Groups added in this section can read bitstreams (attachments) in items by default. Click the restrict button to restrict default bitstream read access rights to a specific user group.
5. Editors - Editors can edit the metadata of submissions and then accept or reject them. Click on the create button to add the workflow step of editing metadata and assigning roles to specific users or user groups.
6. Final editors - Final editors can edit the metadata of incoming submissions but can not reject them. Click the Create button to add this workflow step to the collection and assign a role to specific users or user groups.
7. Reviewers - Reviewers can accept or reject incoming submissions. However, they can not edit the metadata. Click the Create button to add this workflow step to the collection and assign a role to specific users or user groups.

Content Source

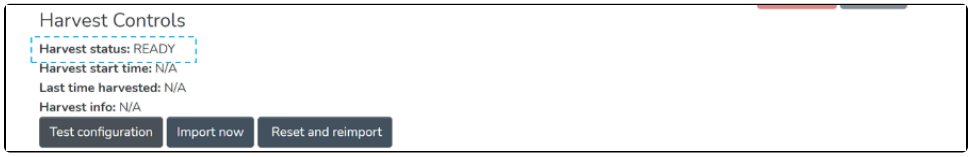
Step 16: This tab enables harvesting the content from external sources using OAI standards. Users can start harvesting by clicking the checkbox, “This collection harvests its content from an external source.”



Step 17: Users will see various parameters related to OAI-based content harvesting upon clicking the checkbox as explained in the previous step. Below is the explanation of elements appearing under Configure an external source header.

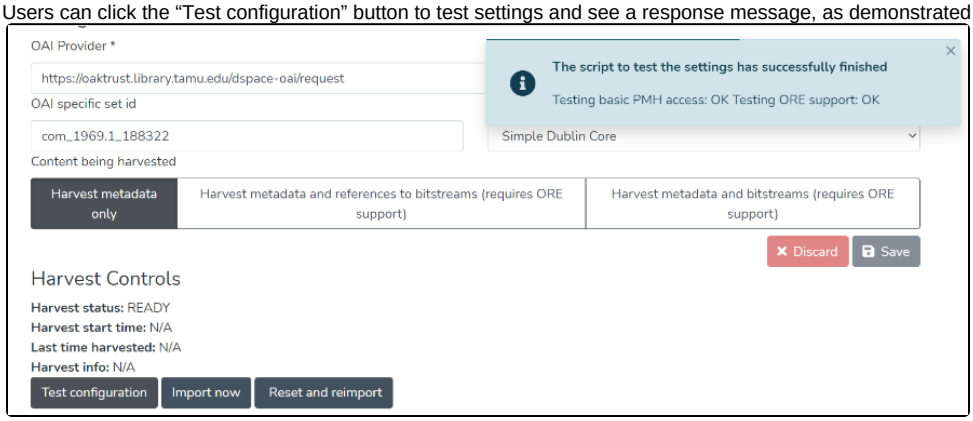


1. OAI Provider – Enter the source OAI provider's URL.
2. OAI-specific set ID – Enter the set ID to source content.
3. Metadata Format – Select a suitable metadata format using the dropdown list, e.g., Simple Dublin Core, Qualified Dublin Core, and DSpace Intermediate metadata.
4. Harvest metadata only – Select this option to harvest only metadata from the source.
5. Harvest metadata and references to bitstreams (requires ORE support) – Click on this option to harvest metadata and reference links to corresponding bitstreams.
6. Harvest metadata and bitstream (requires ORE support) – Use this option to harvest both metadata and corresponding bitstreams into the target repository.
7. Click on the 'Save' button to update harvesting settings.
8. Upon clicking the save button and subject to successful validation of values entered, "Harvest Status" will turn to "Ready," as demonstrated in the screenshot below.



9. After successfully configuring an OAI profile, these buttons will become active, and the user can start harvesting data immediately.

10. Users can click the “Test configuration” button to test settings and see a response message, as demonstrated on the screen below.



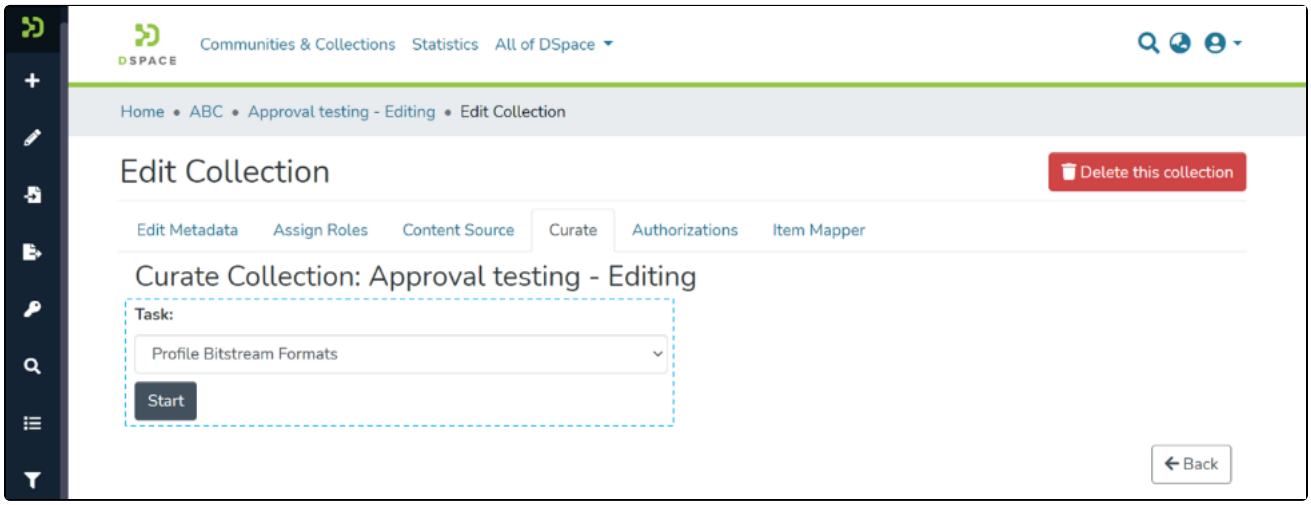
11. Upon successfully testing settings, click on the “Import now” button to harvest metadata immediately.

Curate

Step 18: The Curate tab provides various workflows for curating items stored in the collection. Below are standard flows, and there can be customized curation workflows as well

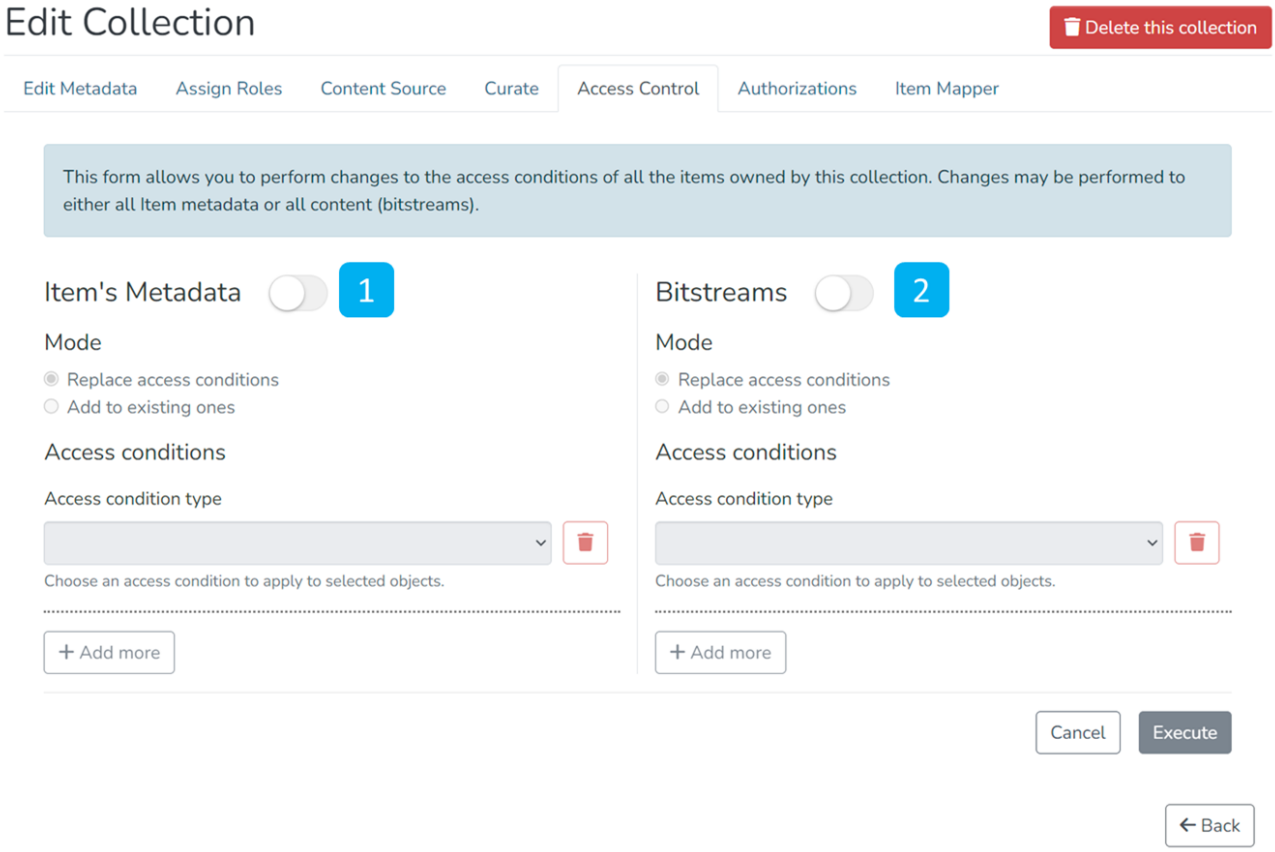
1. Profile bitstream formats
2. Check for Required Metadata
3. Check Links in the Metadata

Users must select a workflow from the dropdown list and click the “Start” button to initiate the curation process.



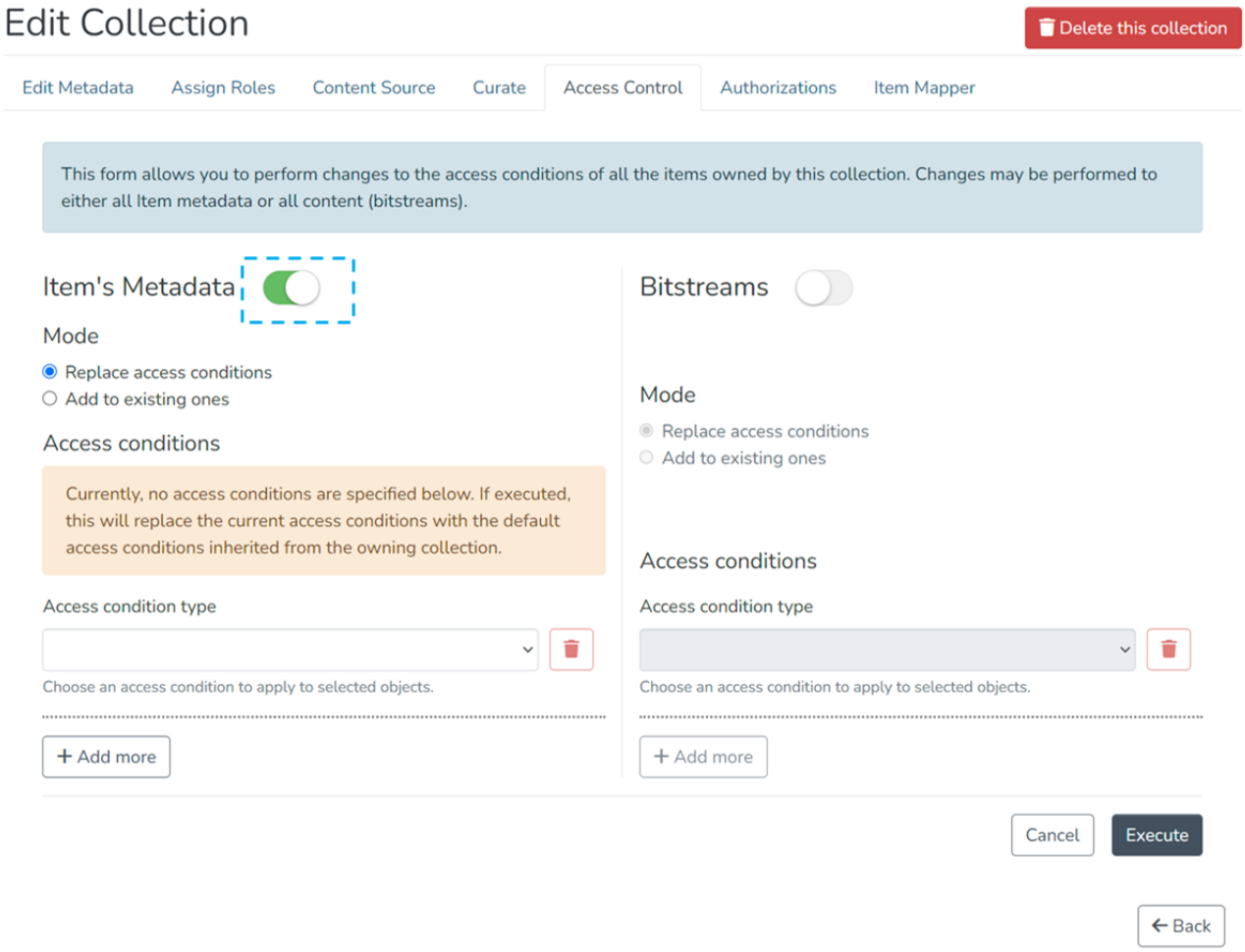
Access Controls

Step 19: The section allows the user to change access conditions of all the items owned by the collection. Changes may be performed to either all Item metadata or all content (bitstreams).

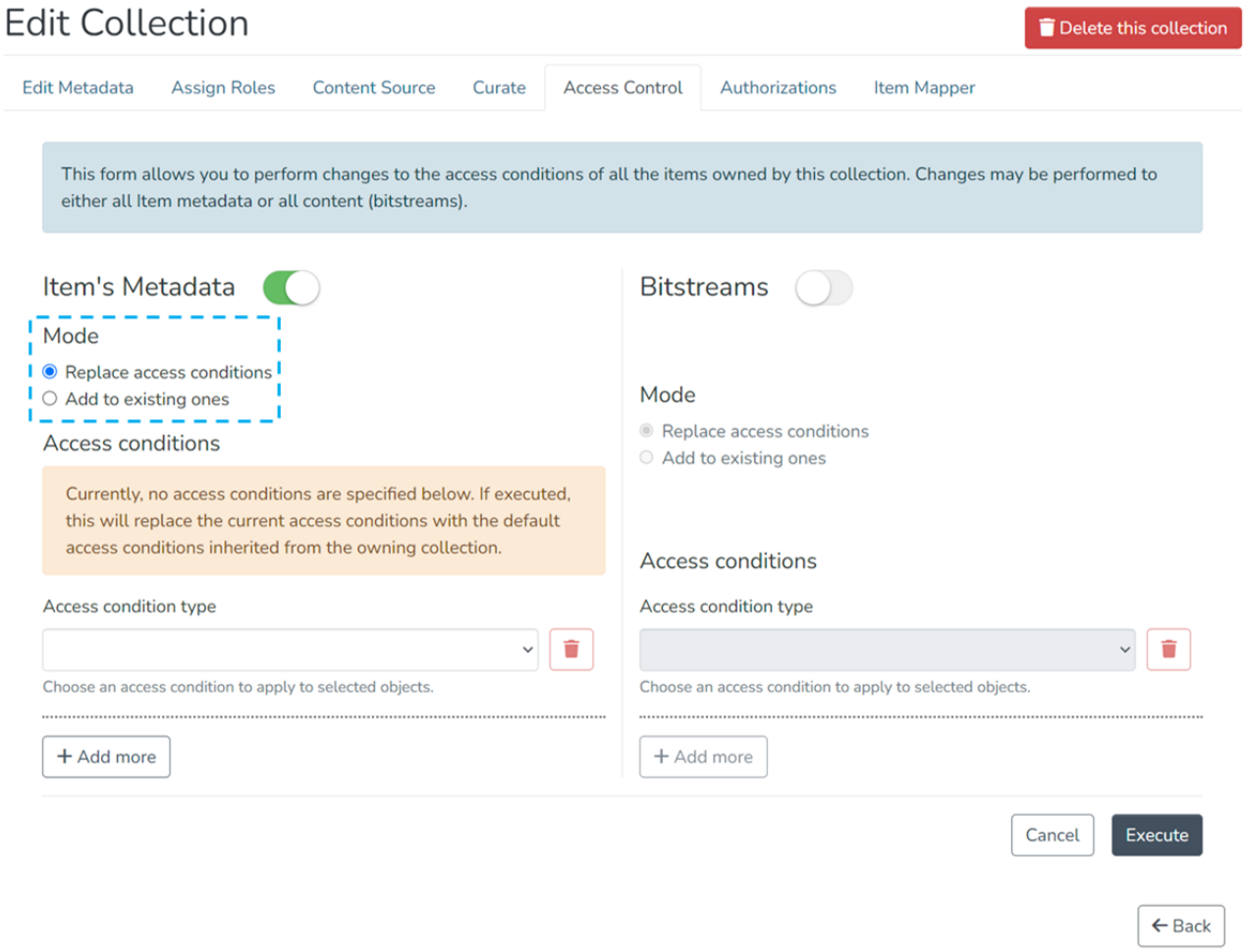


1. Item's Metadata – Click this option to manage access rights on metadata of items stored in the collection.
2. Bitstreams – Click this option to manage access rights on bitstreams (attachments) of items stored in the collection.

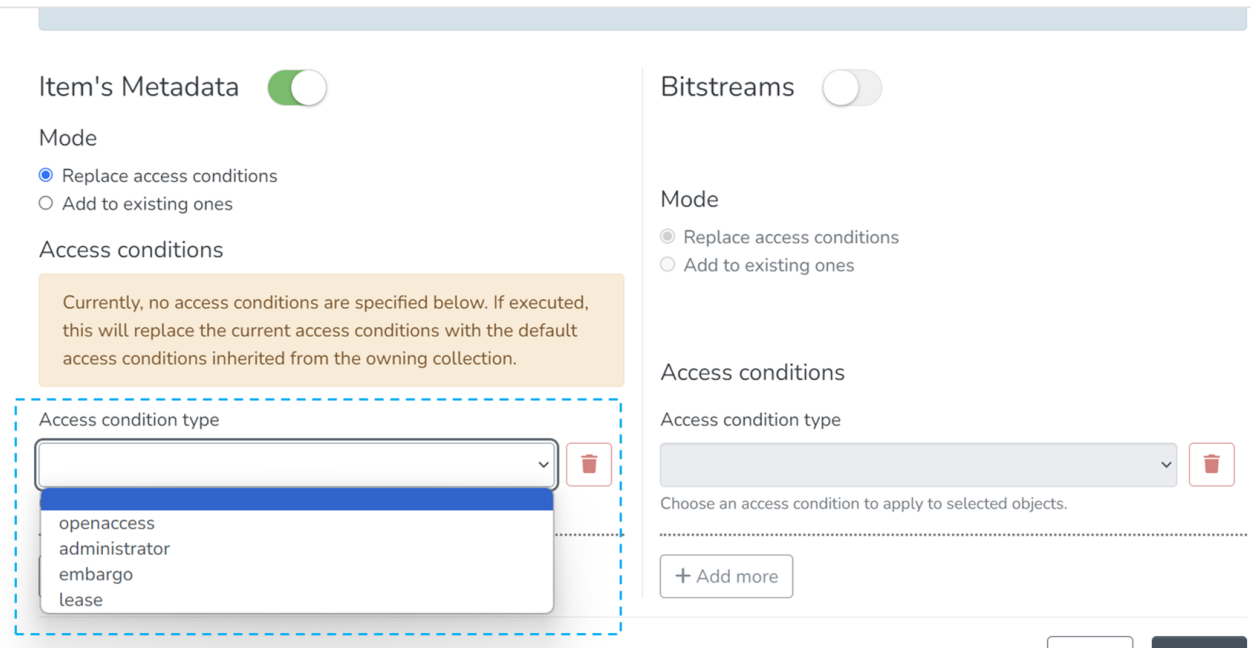
Step 20: Click the switch next to the Item's Metadata header to initiate changes to the access rights on the metadata of items.



Step 21: Select a Mode from the following options as per update requirements.

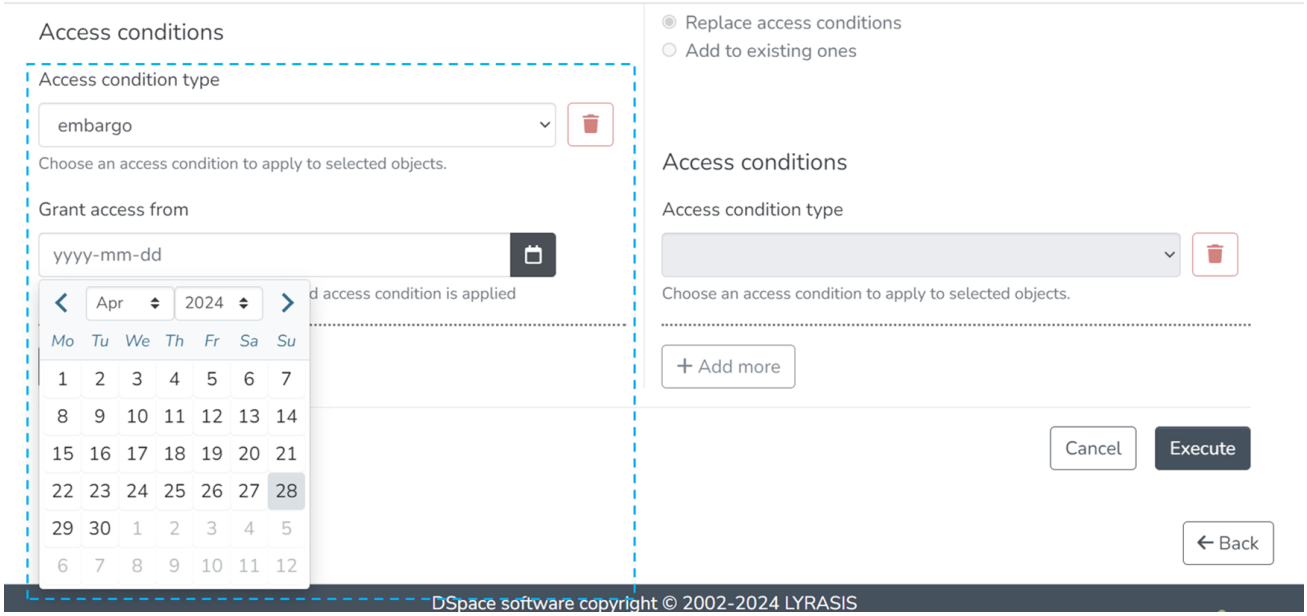


Step 22: Select the access condition type from the drop-down list as required.

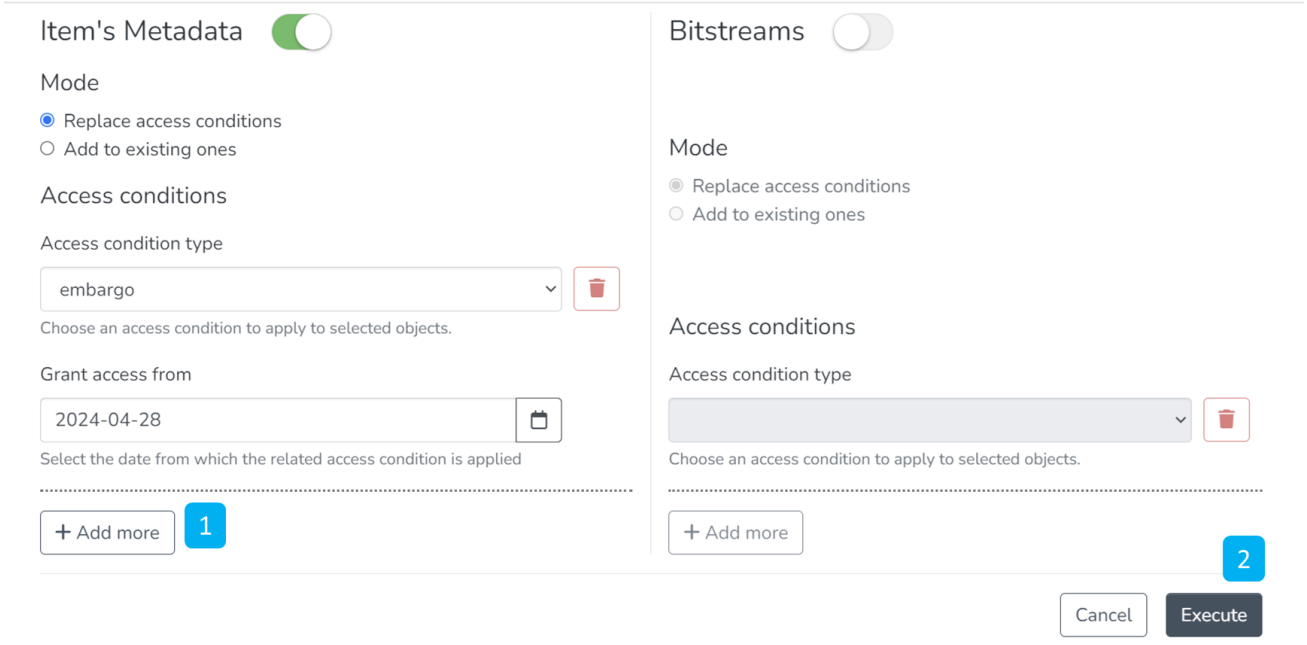


1. openaccess – Select the option to make the Item's metadata available to everyone.
2. administrator – This option will limit items' access to the administrator user group.
3. embargo - Embargo will restrict access to items until the selected future date, as explained in the following steps.
4. lease - The lease will restrict access to items after the selected future date, as explained in the following steps.

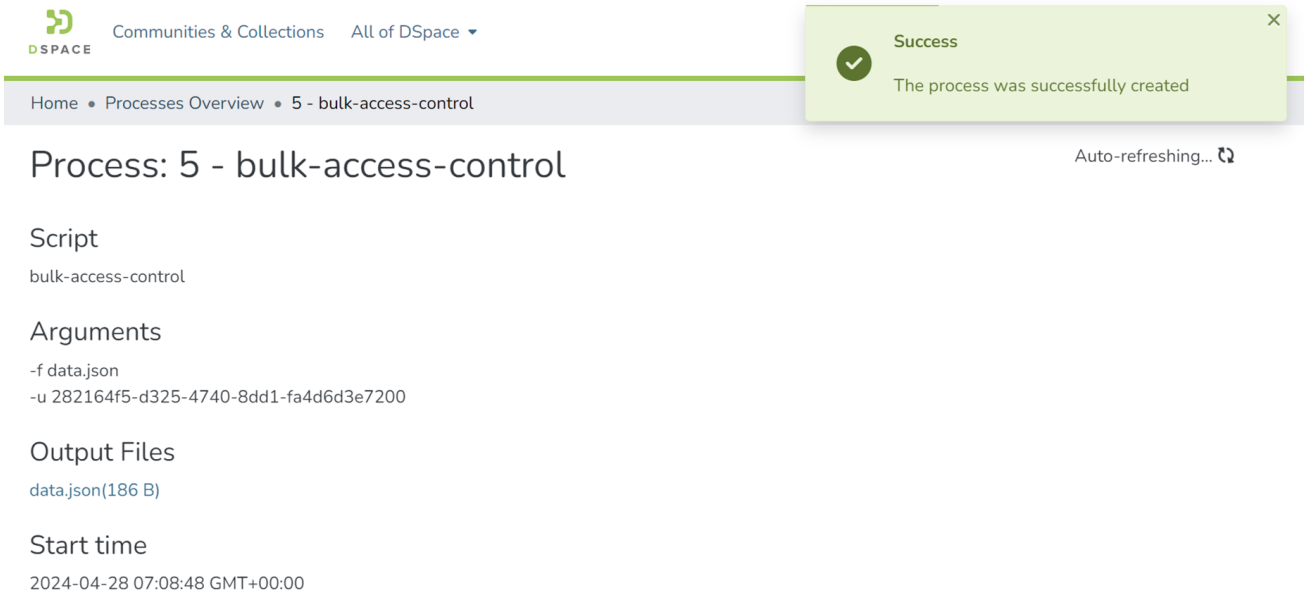
Step 23: Select the 'Embargo' from the drop-down, and in the following field, select an embargo date to limit access to items until the selected date.



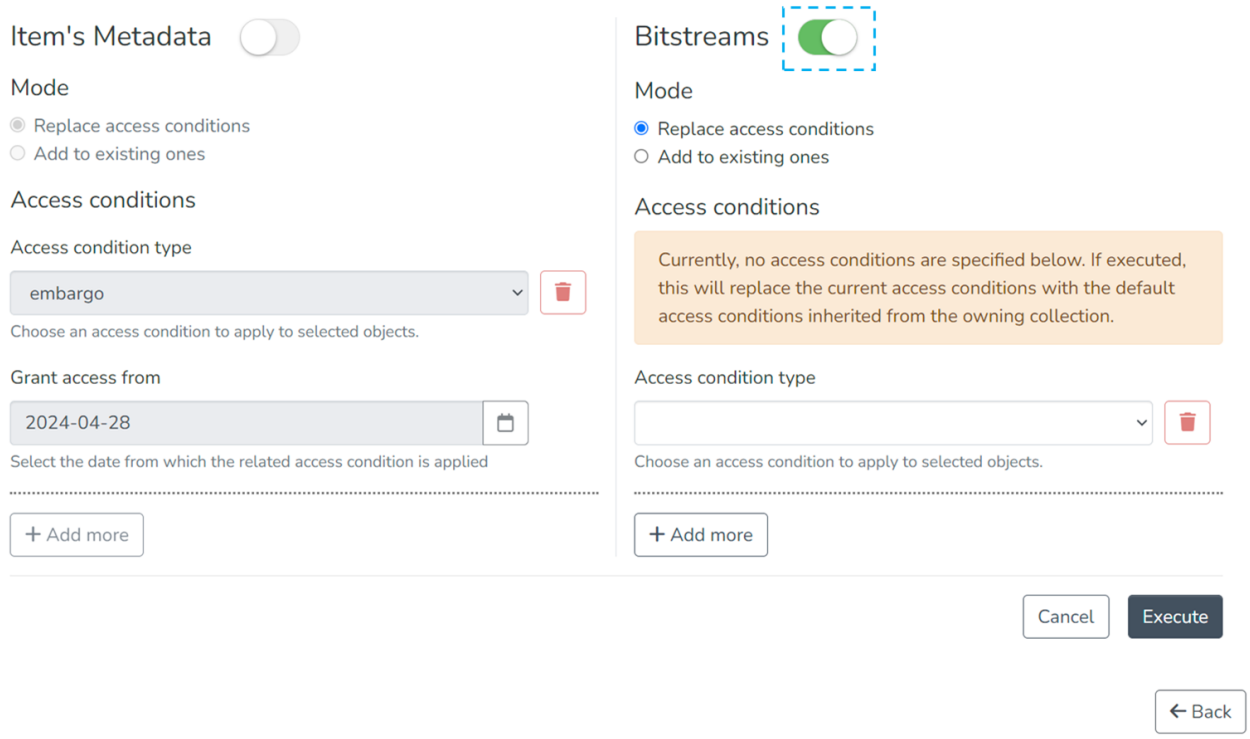
Step 24: Click the 'Add more' button to add another policy by repeating the above steps, and click the 'Execute' button to apply policies to the collection.



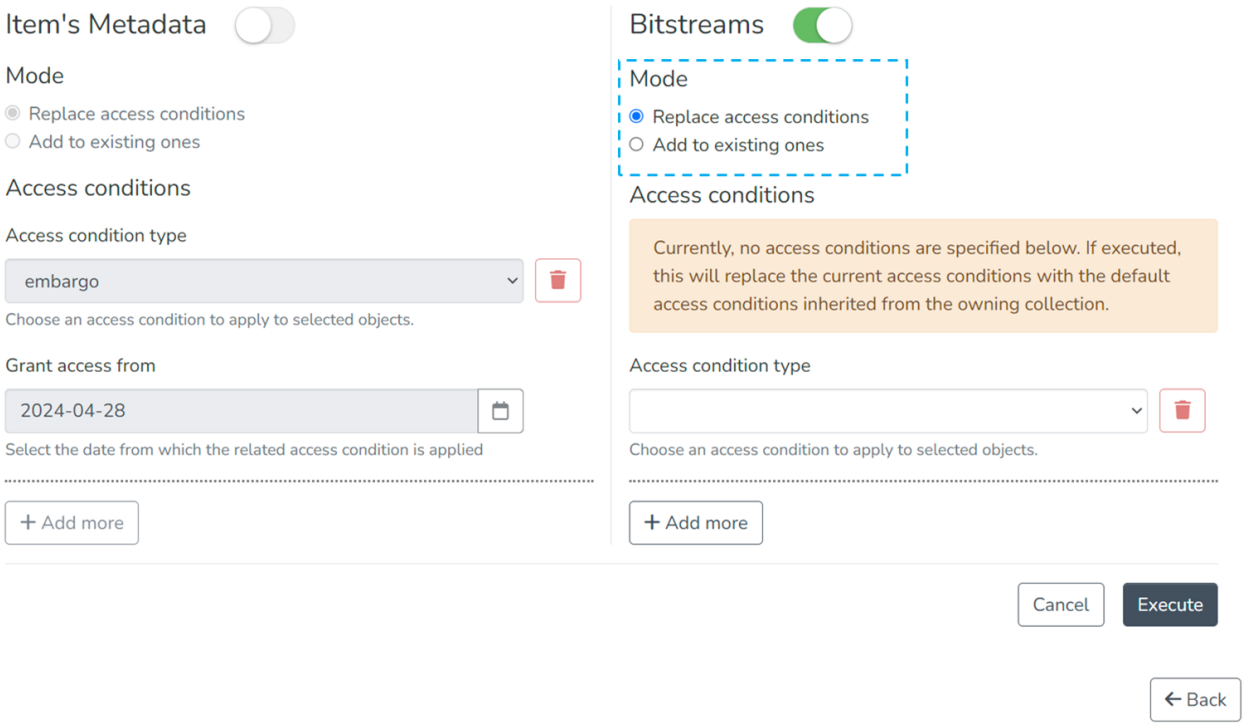
Step 25: Users shall see the following screen upon the successful execution of policies.



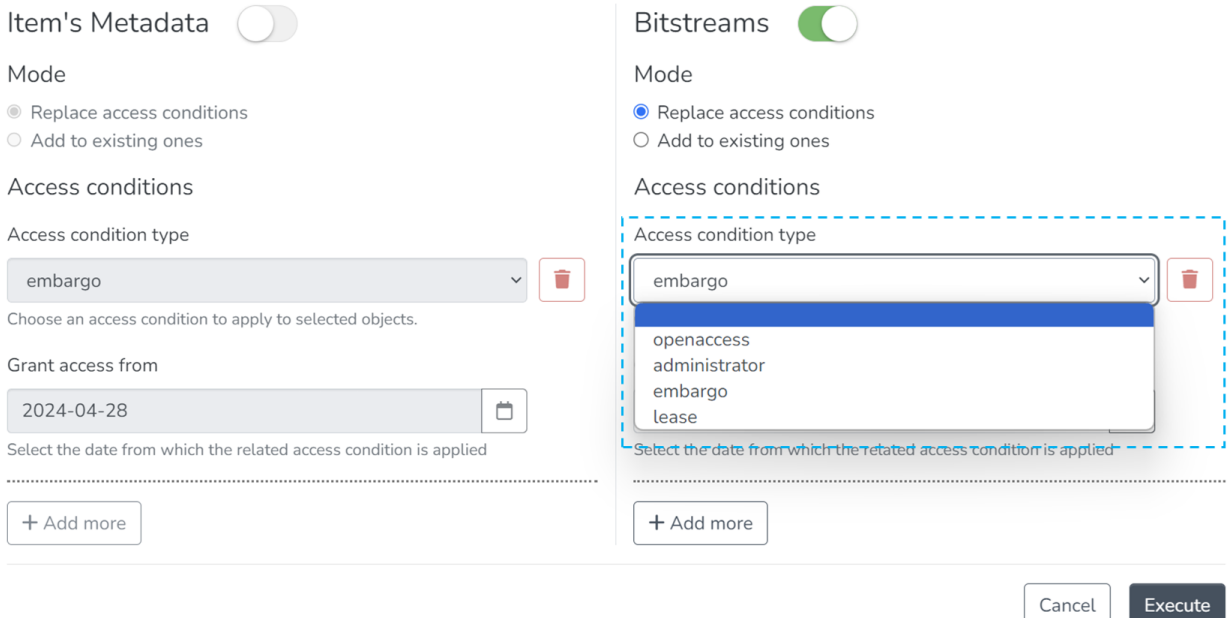
Step 26: Click the switch next to the Bitstream header to initiate changes to the access rights on the bitstreams of items.



Step 27: Select a Mode from the following options as required.

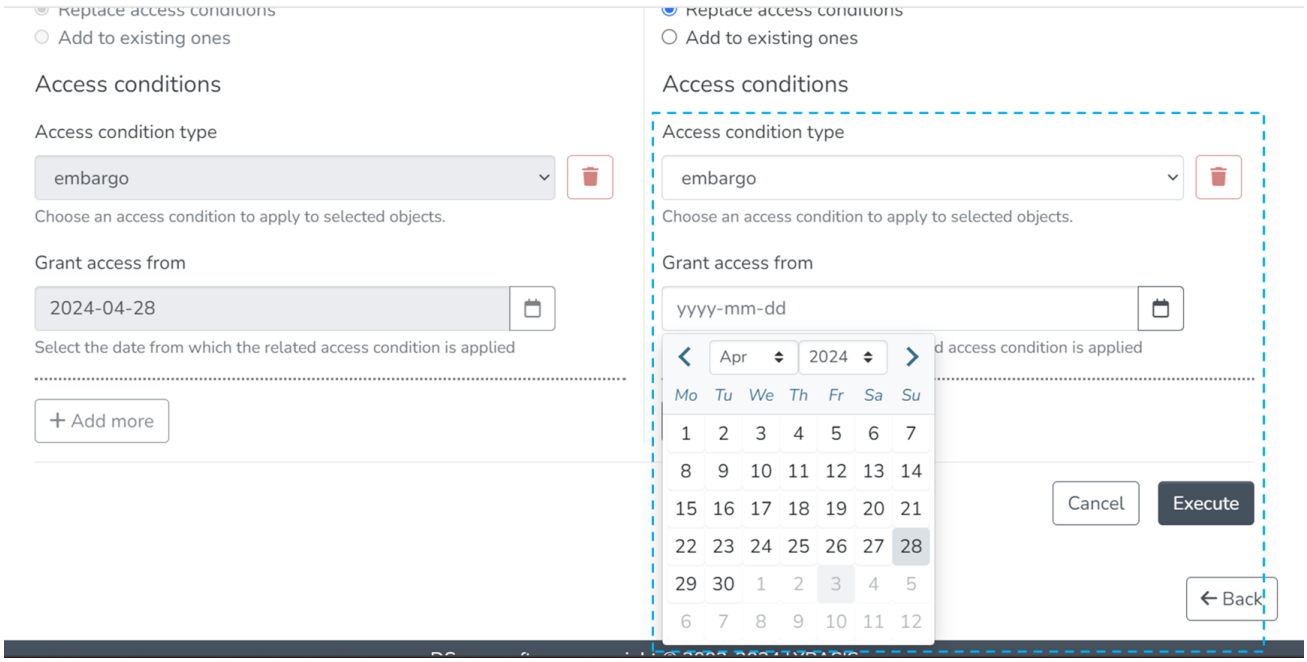


Step 28: Select the access condition type from the drop-down list as required.

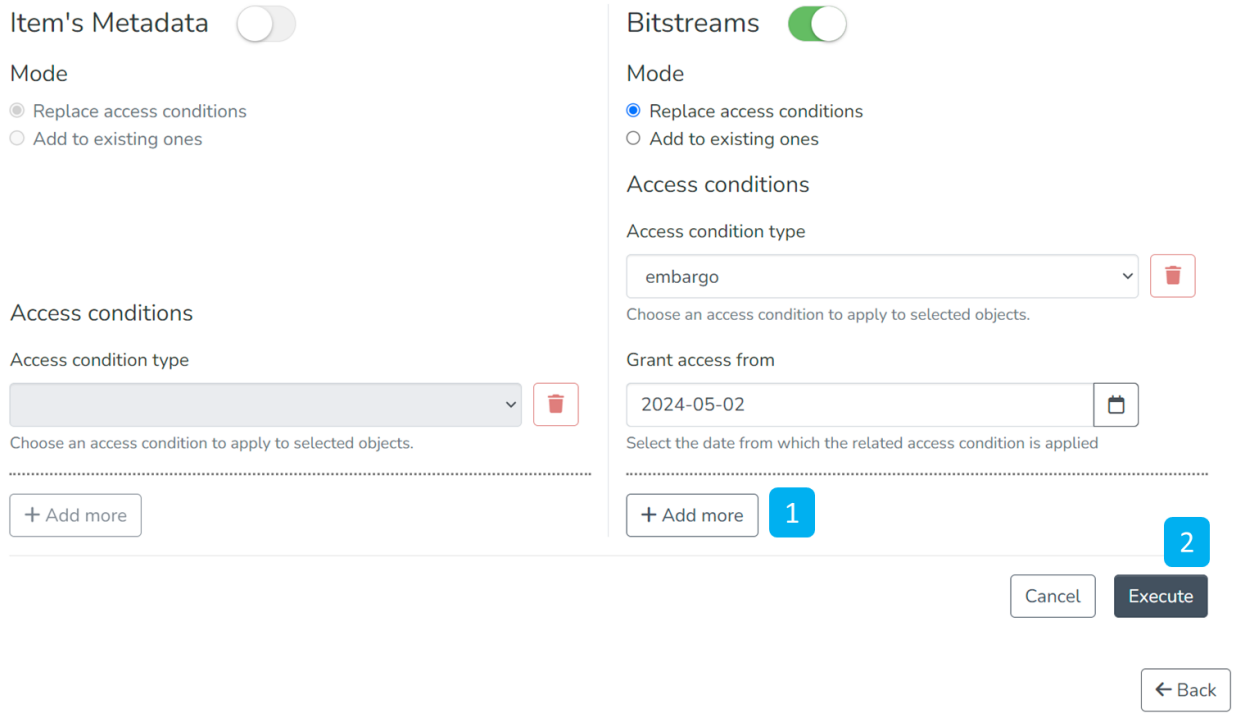


1. openaccess – Select the option to make the bitstream's metadata available to everyone.
2. administrator – This option will limit bitstream access to the administrator user group.
3. embargo - Embargo will restrict access to bitstreams until the selected future date, as explained in the following steps.
4. lease - The lease will restrict access to bitstreams after the selected future date, as explained in the following steps.

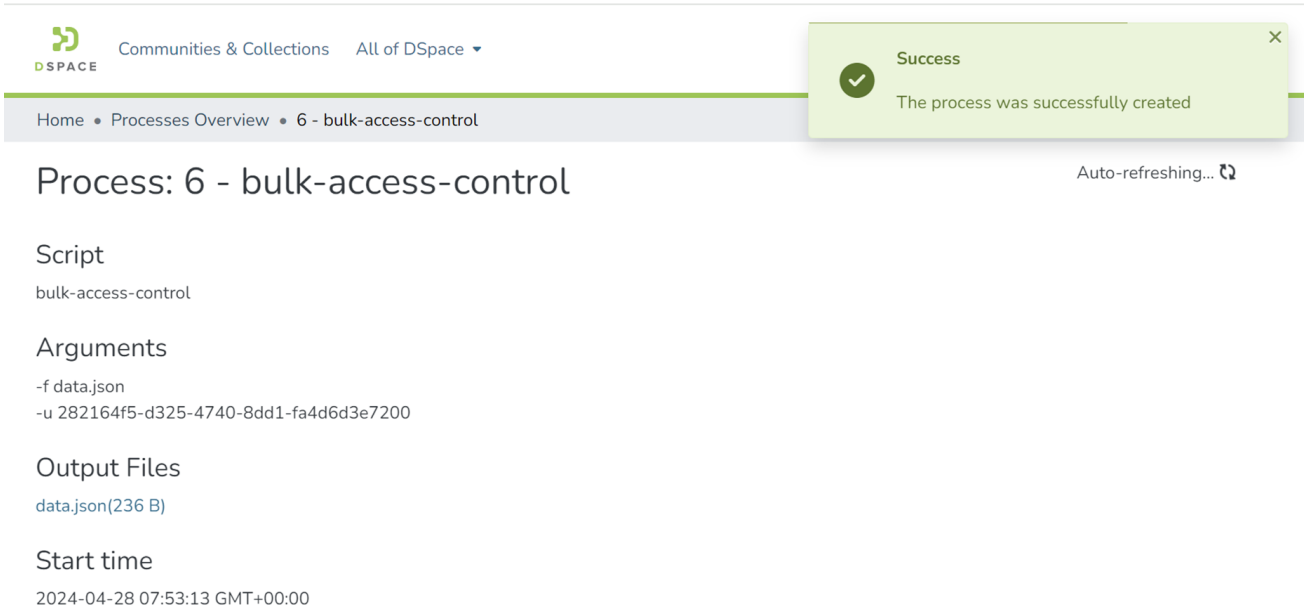
Step 29: Select the 'Embargo' from the drop-down, and in the following field, select an embargo date to limit access to bitstream until the selected date.



Step 30: Click the 'Add more' button to add another policy by repeating the above steps, and click the 'Execute' button to apply policies to the collection.

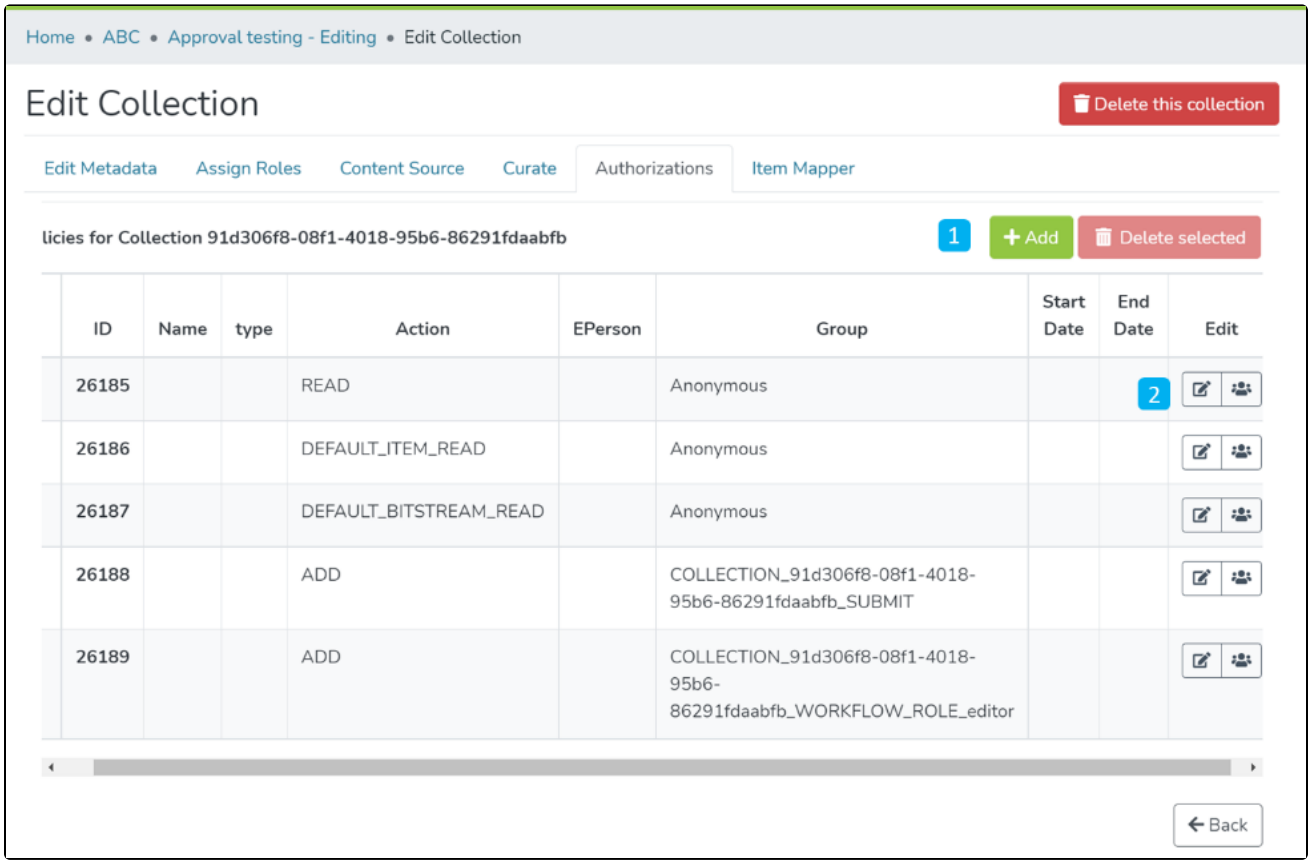


Step 31: Users shall see the following screen upon the successful execution of policies.



Authorizations

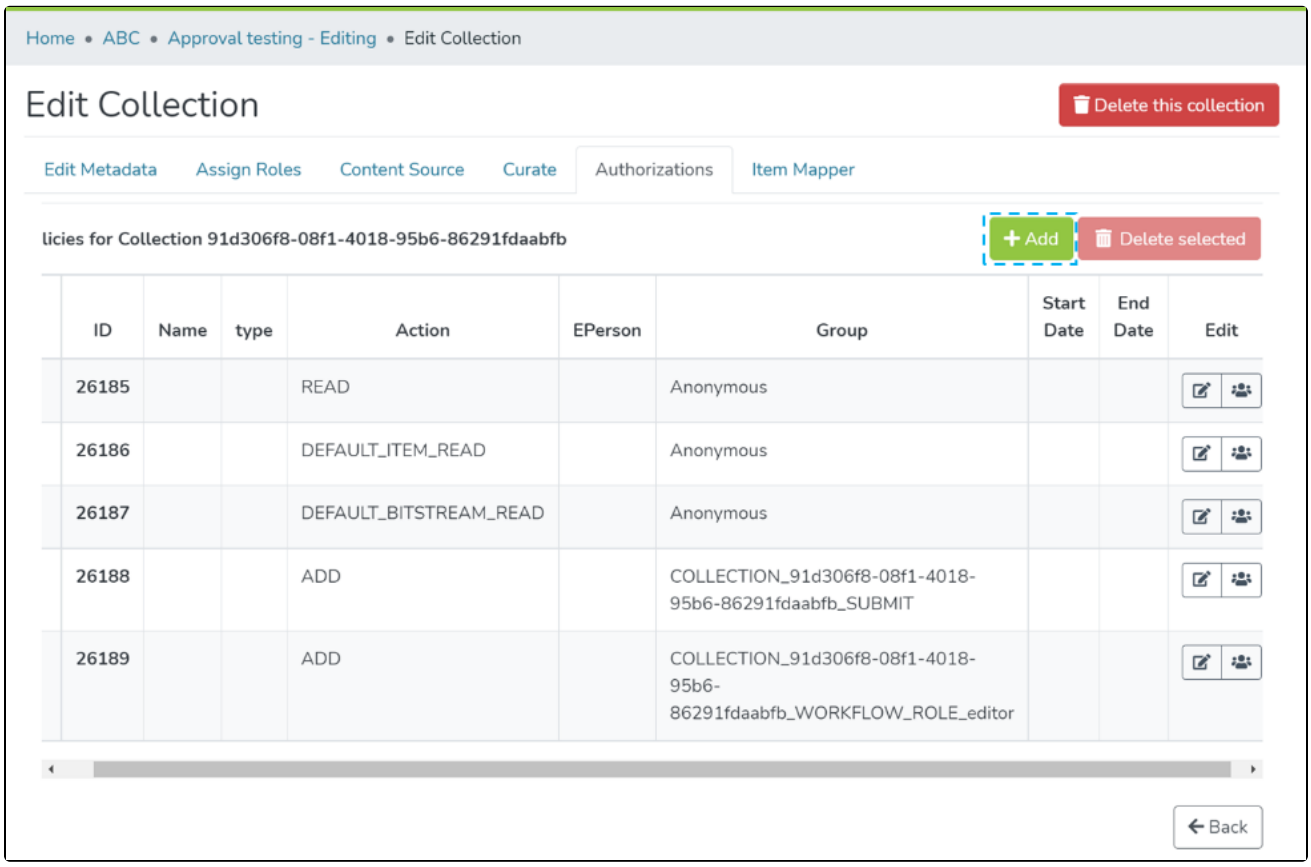
Step 32: The Authorizations tab has all the policies defined for the collection. These are in addition to policies created from the “Assign Roles” tab. Key actions available in this tab are explained below.



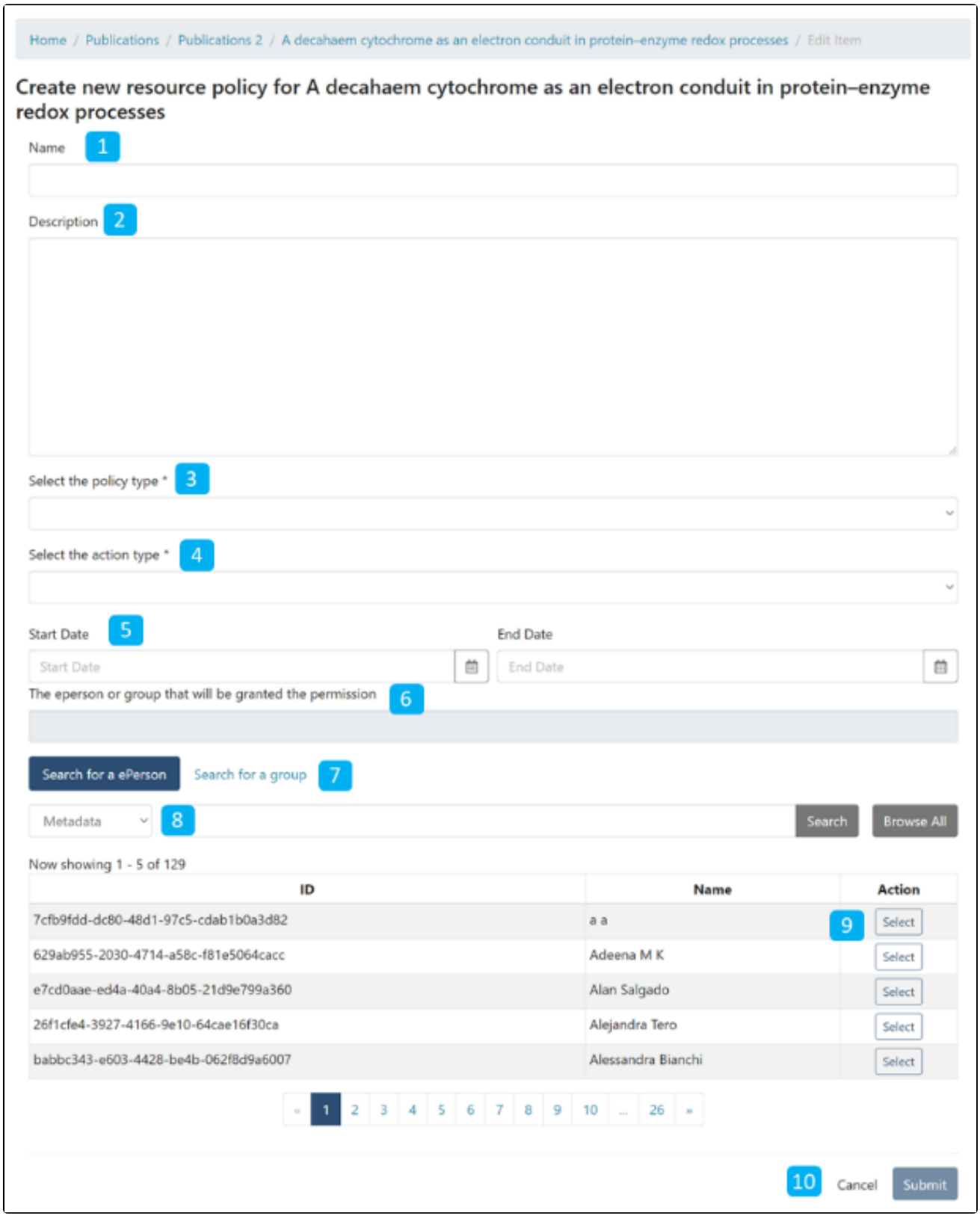
1. Manage Policies – Click the Add button or select policies from the table to create a new policy. Next, click the 'Delete selected' button for a batch deletion of the policies.

2. Edit policy and members in a policy – Click the edit button to edit an individual policy or click on the group icon to edit the user group.

Step 33: Click on the 'Add' button to create a new Authorization policy.



Step 34: Users can add information in the fields available in this form to the policy and save it by clicking the submit button. Please see the description of each field followed by the below screenshot.



1. **Name:** Enter the Policy name in this field.
2. **Description:** Enter the Policy description here for future reference and understanding of other users.
3. **Select the policy type:** The user can select one of the following policy classification types from the list
    a. TYPE_SUBMISSION: a policy in place during the submission
    b. TYPE_WORKFLOW: a policy in place during the approval workflow
    c. TYPE_INHERITED: a policy that has been inherited from a container (the collection)
    d. TYPE_CUSTOM: a policy defined by the user during the submission or workflow phase
4. **Select the action type:** The user can select one of the following actions from the dropdown list:

a. READ - Ability to view the specific object
b. WRITE - Ability to edit the specific object
c. ADD - Ability to add an object to a parent object (e.g. add a Bitstream to a Bundle, add a Bundle to an Item, add an Item to a Collection , add a Collection to a Community, etc.)
d. REMOVE - Ability to remove an object from a parent object (i.e. the opposite of "ADD")
e. ADMIN - Full Admin permissions on the specific object
f. DELETE - Ability to delete the specific object
g. WITHDRAWN_READ - Ability to view a withdrawn Item (only valid for Items)
h. DEFAULT_BITSTREAM_READ - Default "READ" permissions added to any Bitstreams of a newly submitted Item (only valid for Collections)
i. DEFAULT_ITEM_READ - Default "READ" permissions added to any newly submitted Item (only valid for Collections)

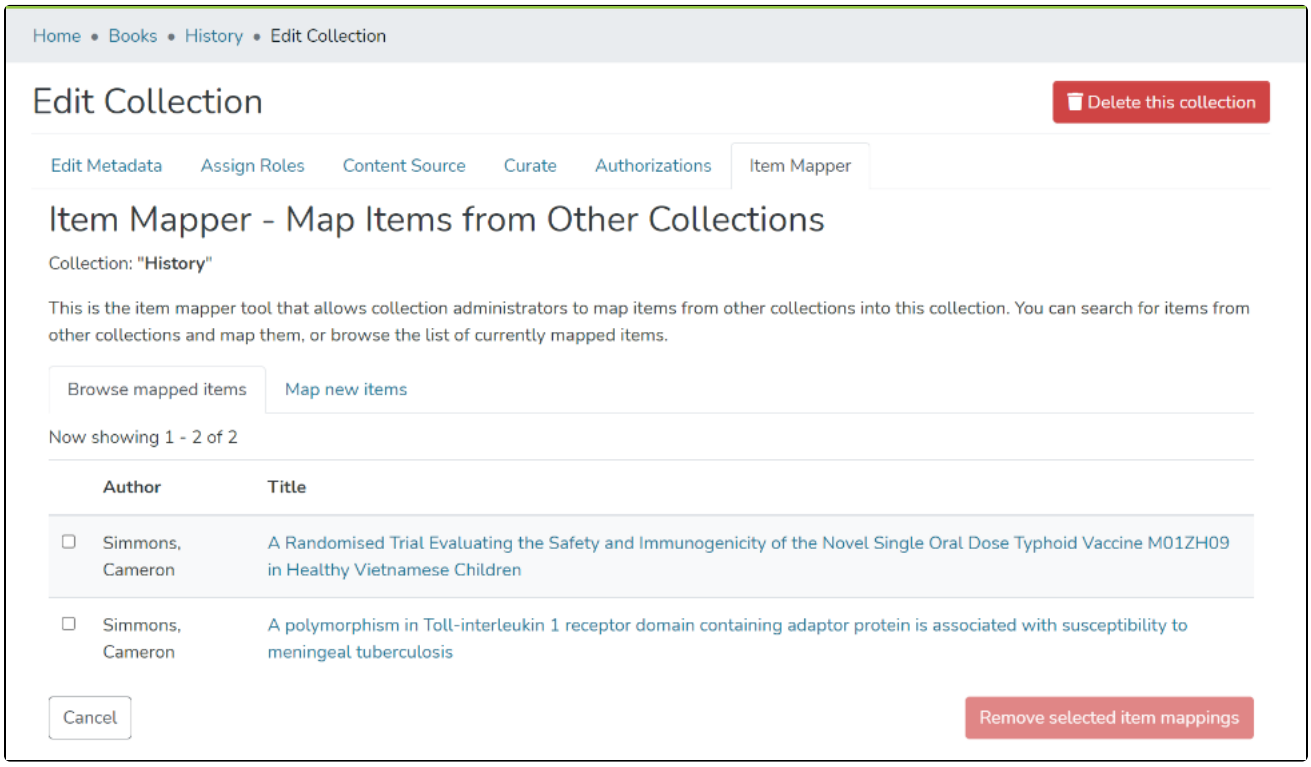
5. **Start date – end date:** The user can select the start date and end date for using the policy, should they want to apply it for a fixed period.
6. **The eperson or group that will be granted the permission:** List of users/groups selected for granting permission under the policy
7. **Search for an ePerson / Search for a group:** Select ePerson or group for searching the entity
8. **Search field:** Enter keywords for searching the ePerson/Group
9. **ePerson/Group list:** Click on the select button against the user/group you want to add to the policy
10. **Submit/Cancel button:** Click on the Submit button to complete policy creation or click on the Cancel button to cancel the entire process.

You'll see a confirmation prompt upon successfully creating the policy, as shown below. After that, the user will be back on the Authorizations screen.

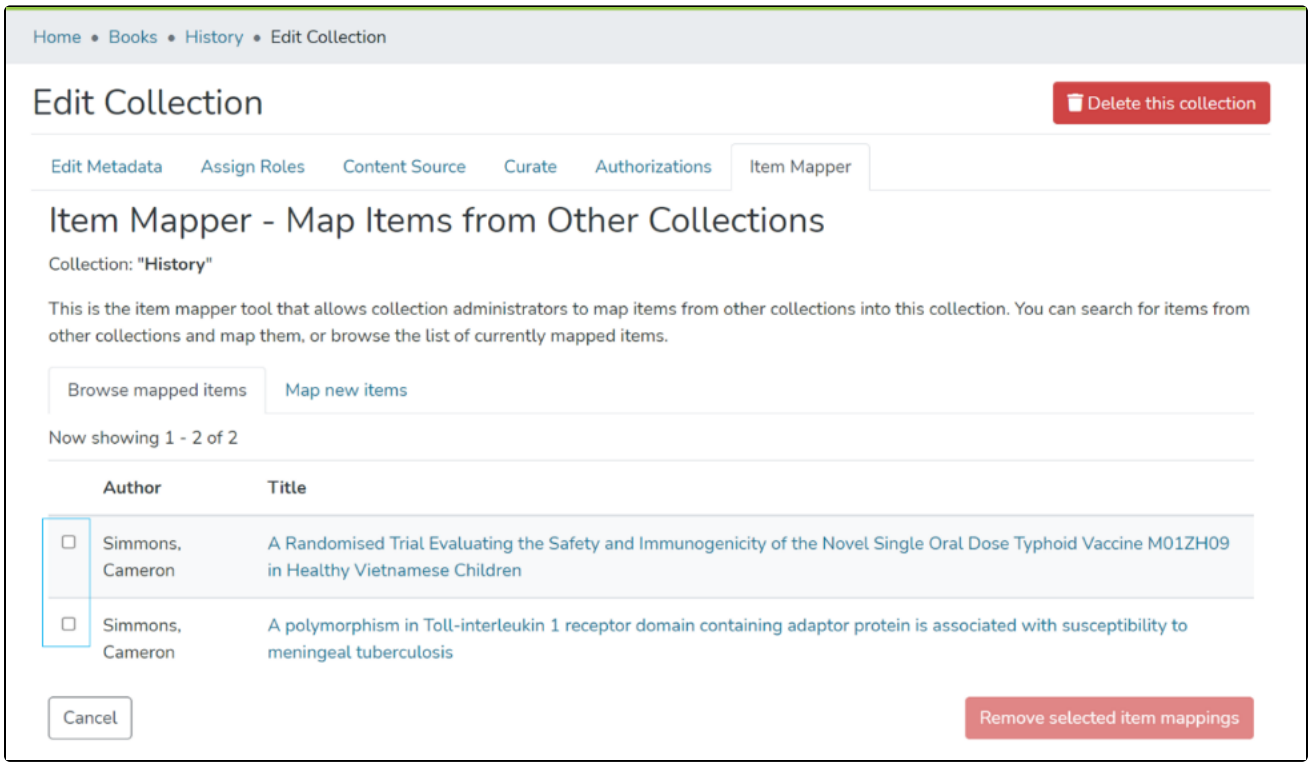
Item Mapper

Manage mapped items

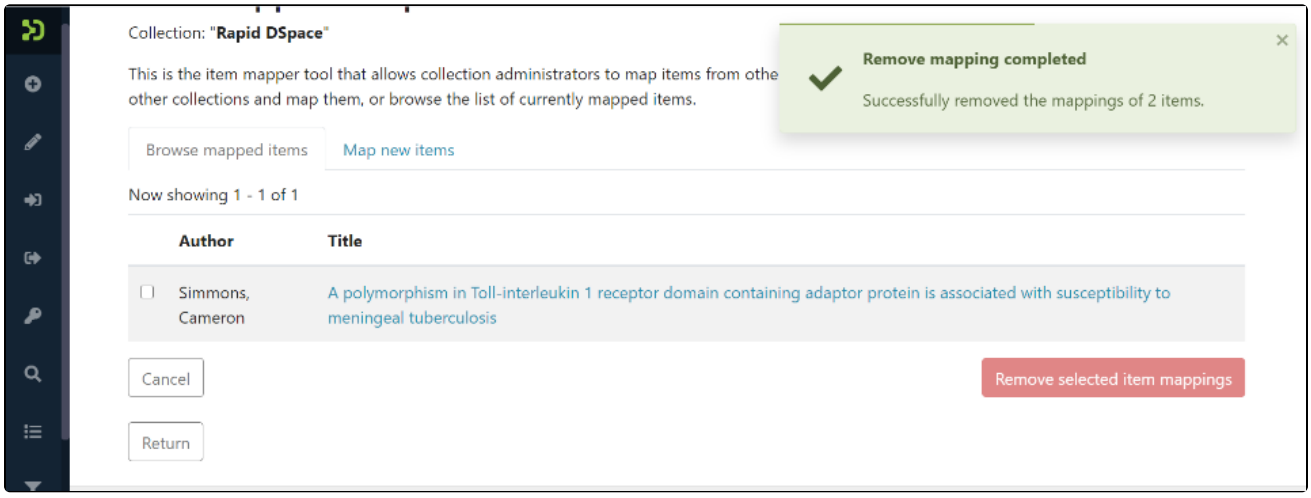
Step 35: The item mapper tab allows users to map items from other collections and manage mapped items.



Step 36: You'll see items mapped with collections under the "Browse mapped items" tab. Click on the checkbox appearing with each item to select the item (s) required to be unmapped.

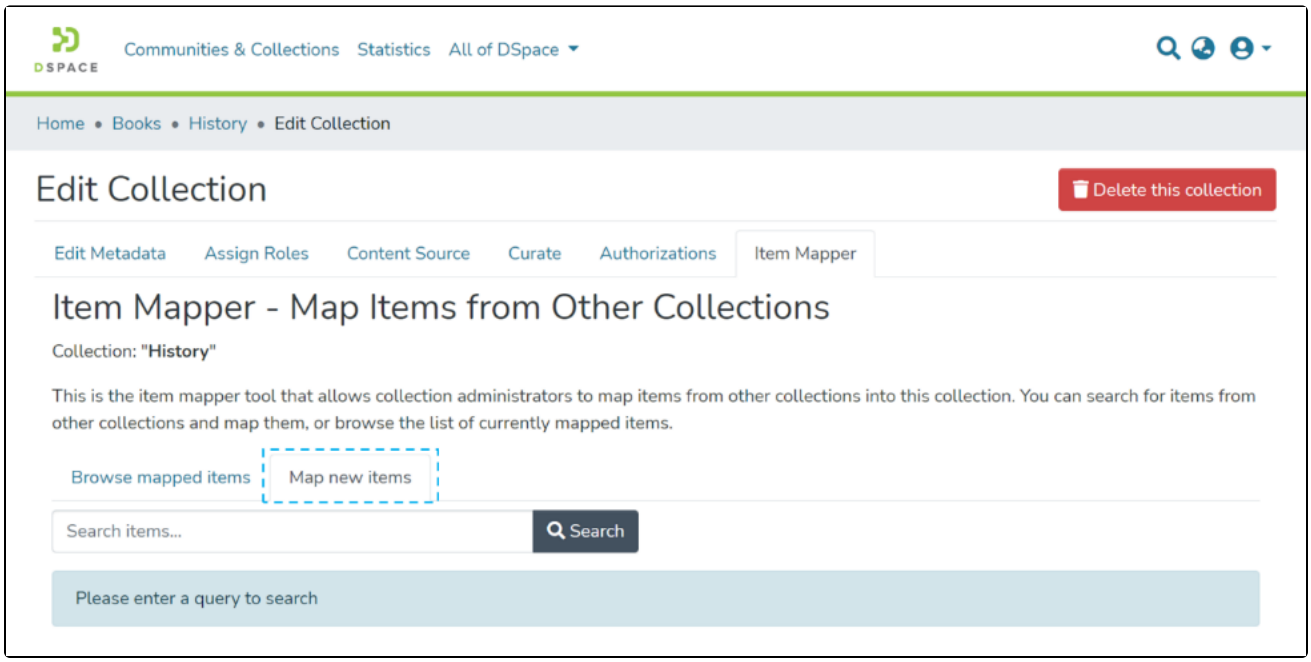


Step 37: After selecting items required to be unmapped, please click on “Remove selected item mappings” to complete the operation. Click on the “Cancel” button on the left of “Remove selected item mappings” to cancel the process.



Map new items

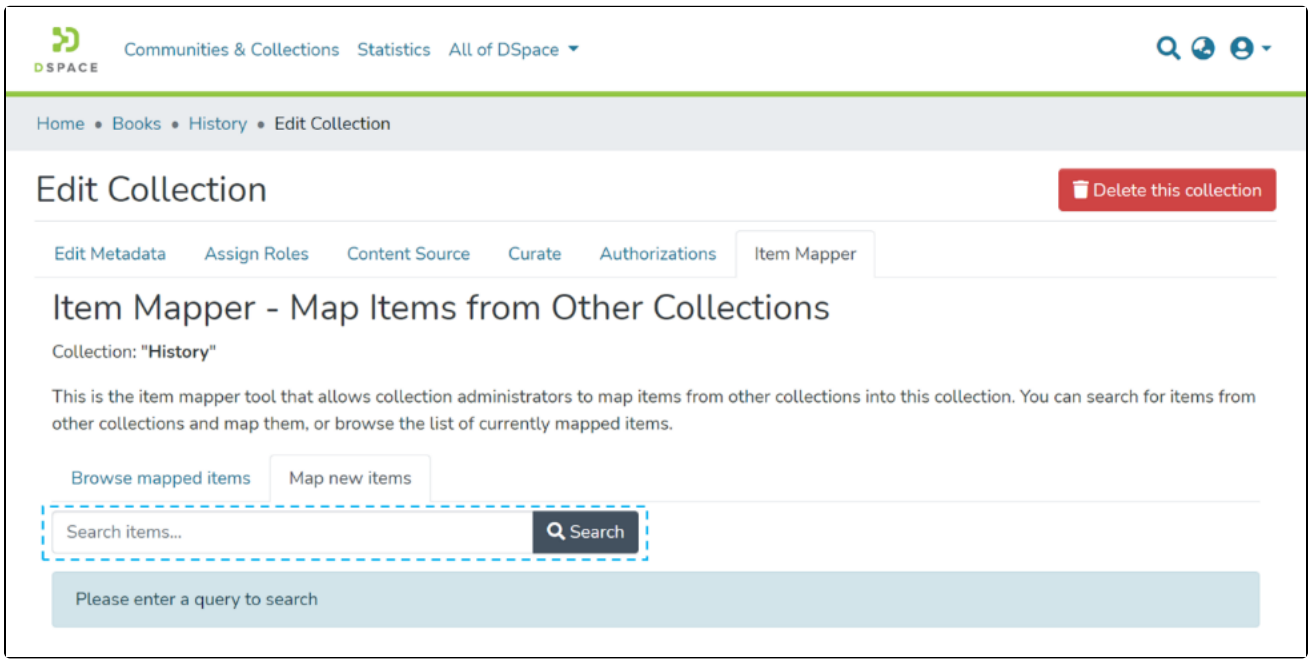
Step 38: Click on “Map new items” to search for items for mapping.



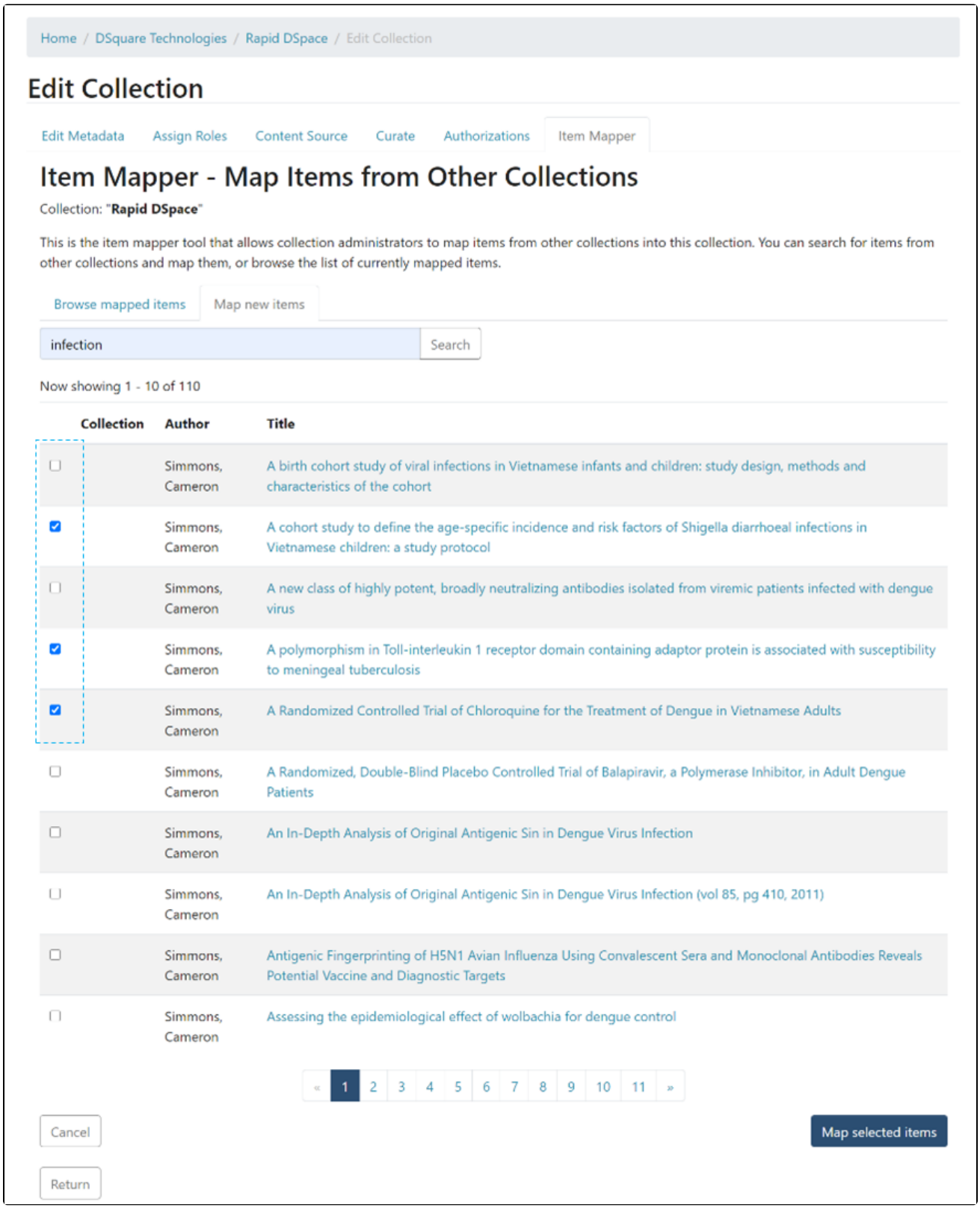
Step 39: After confirming the non-existence of the target item in the existing mapped items list, please click on “Map new items.” Then, enter keywords /keyphrases in the search field to search for target items.

Click on the “Search” button as highlighted on the below screen.

You must know that you can enter keywords or keyphrases from any metadata field. The search field under “Map new items” works exactly like the basic search field of DSpace.

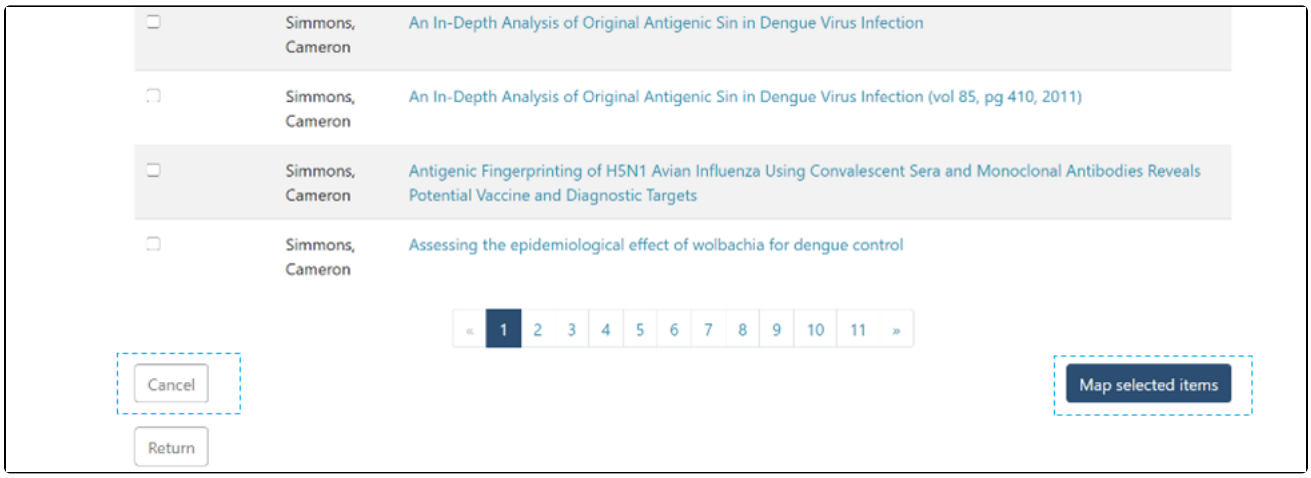


Step 40: Users can select target items from the search results by clicking the checkbox appearing with items.

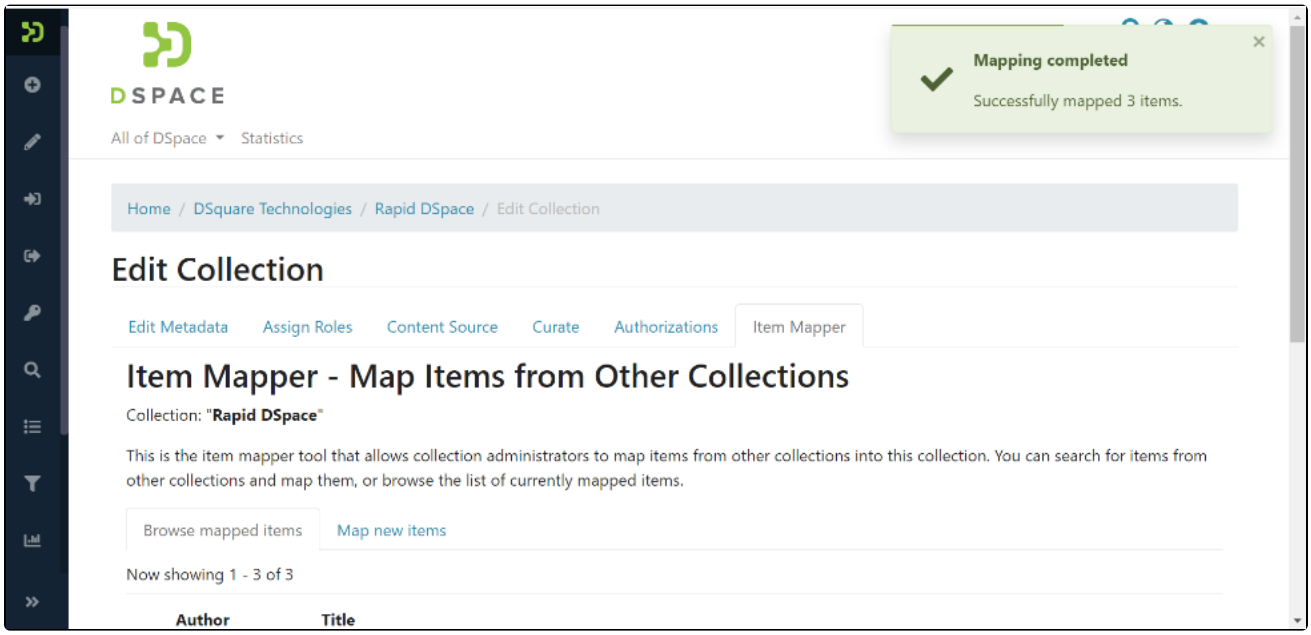


Step 41: After selecting target items, please click on the “Map selected items” button at the bottom of the page to complete the item mapping process.

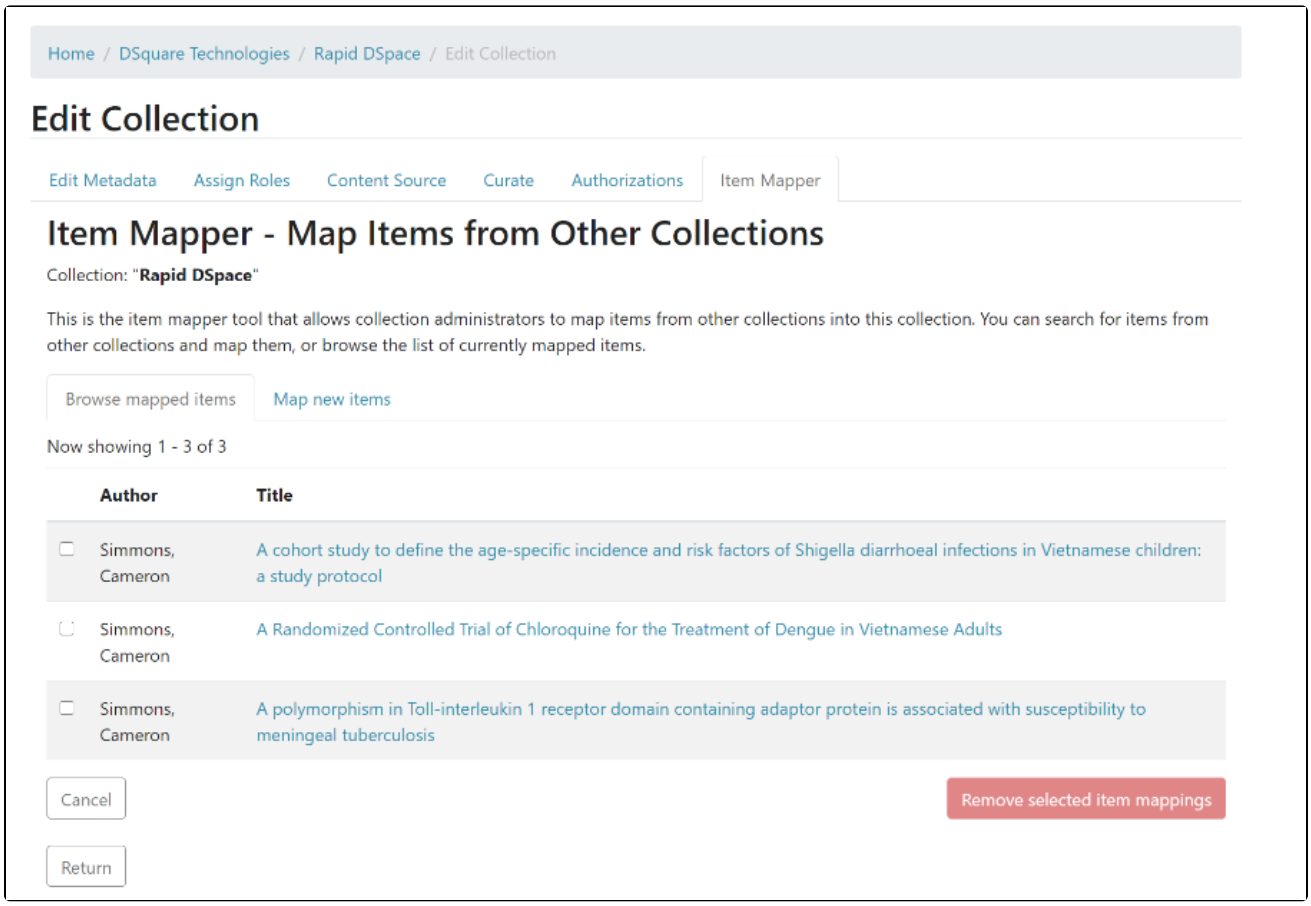
Click the “Cancel” button to cancel the activity and return to the collection edit page.



Step 42: A prompt confirming the successful mapping of items will appear upon completing the task, as demonstrated below.



Mapped items will appear in the collection and under the “Browse mapped items” tab, as demonstrated below.


Home / DSquare Technologies / Rapid DSpace / Edit Collection
Edit Collection
Edit Metadata
Assign Roles
Content Source
Curate
Authorizations
Item Mapper
Item Mapper - Map Items from Other Collections
Collection: "Rapid DSpace"
This is the item mapper tool that allows collection administrators to map items from other collections into this collection. You can search for items from other collections and map them, or browse the list of currently mapped items.
Browse mapped items
Map new items
Now showing 1 - 3 of 3
Author
Title
Simmons, Cameron
A cohort study to define the age-specific incidence and risk factors of Shigella diarrhoeal infections in Vietnamese children: a study protocol
Simmons, Cameron
A Randomized Controlled Trial of Chloroquine for the Treatment of Dengue in Vietnamese Adults
Simmons, Cameron
A polymorphism in Toll-interleukin 1 receptor domain containing adaptor protein is associated with susceptibility to meningeal tuberculosis
Cancel
Remove selected item mappings
Return

Community Management

The community is the primary storage level in the DSpace's storage hierarchy that holds sub-community and collections. This document provides an overview of creating, editing, and deleting a community. The documentation below assumes that the user has the relevant authorizations. For example, the admin menu and edit buttons would appear to a user having community administration permission.

If you're unsure about community administration permissions assigned to your account for the target community, contact your system administrator.

- Create a Community
- Delete Community
- Edit Community

Create a Community

- Scope and Use Case
- Audience
- Create Community

Scope and Use Case

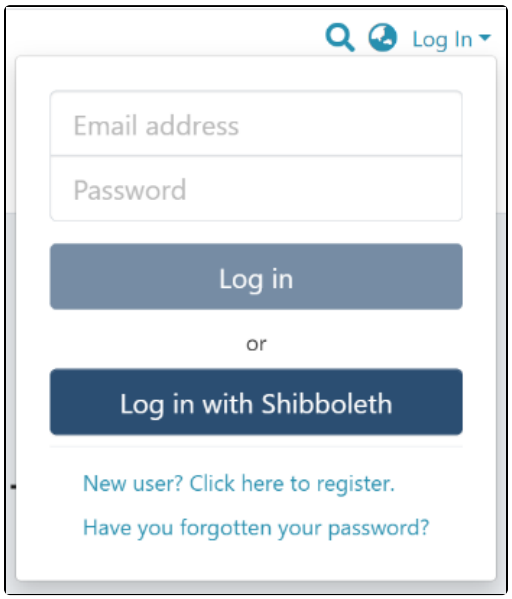
DSpace uses a hierarchy of Communities, Collections, and Items. Communities hold sub-communities (which hold collections) and collections (which hold items). Use these instructions to create a new community or sub-community.

Audience

- Repository Administrator
- Community Administrator

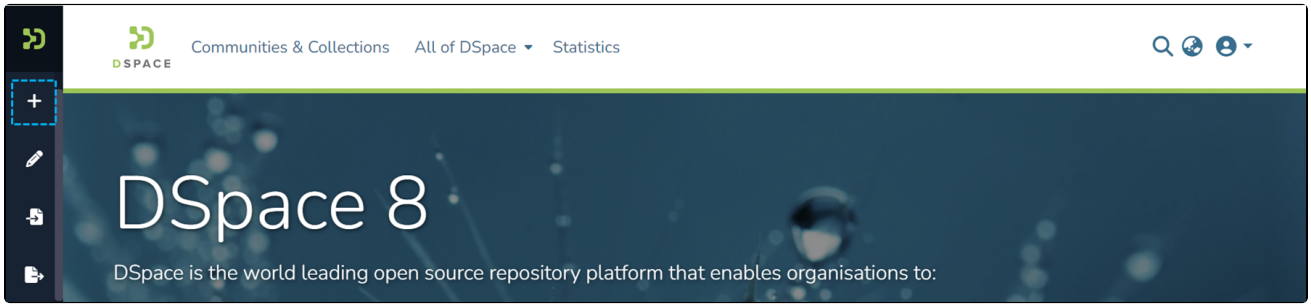
Create Community

Step 1: Login using your credentials



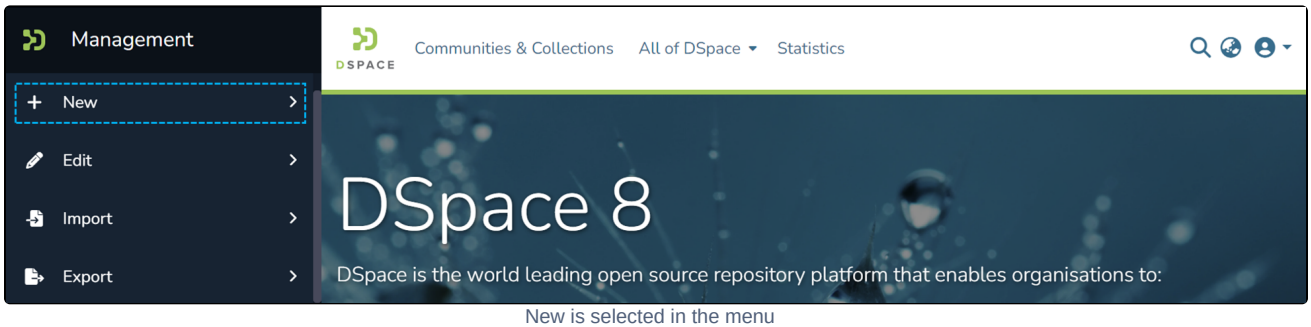
Login window

Step 2: Move your cursor over the “+” sign

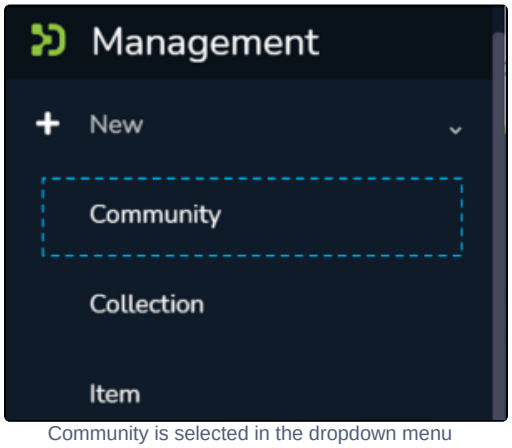


Screen shot of top part of menu. New (plus sign) is the first menu item.

Step 3: Click on the “New” link and click on “Community” to proceed with the community creation process.



New is selected in the menu



Community is selected in the dropdown menu

Step 4: A popup providing the option to either create a Parent community or a sub-community will appear, with a list showing existing communities. Create your new community by either:

1. Clicking on “Create a new top-level community” to create a top-level community.
   … or…
2. Typing the name of the existing parent community in the search field to add a sub-community within. Upon typing a few letters of the community’s name, a list of communities having those letters or words will appear. Click on the community name to create a sub-community.

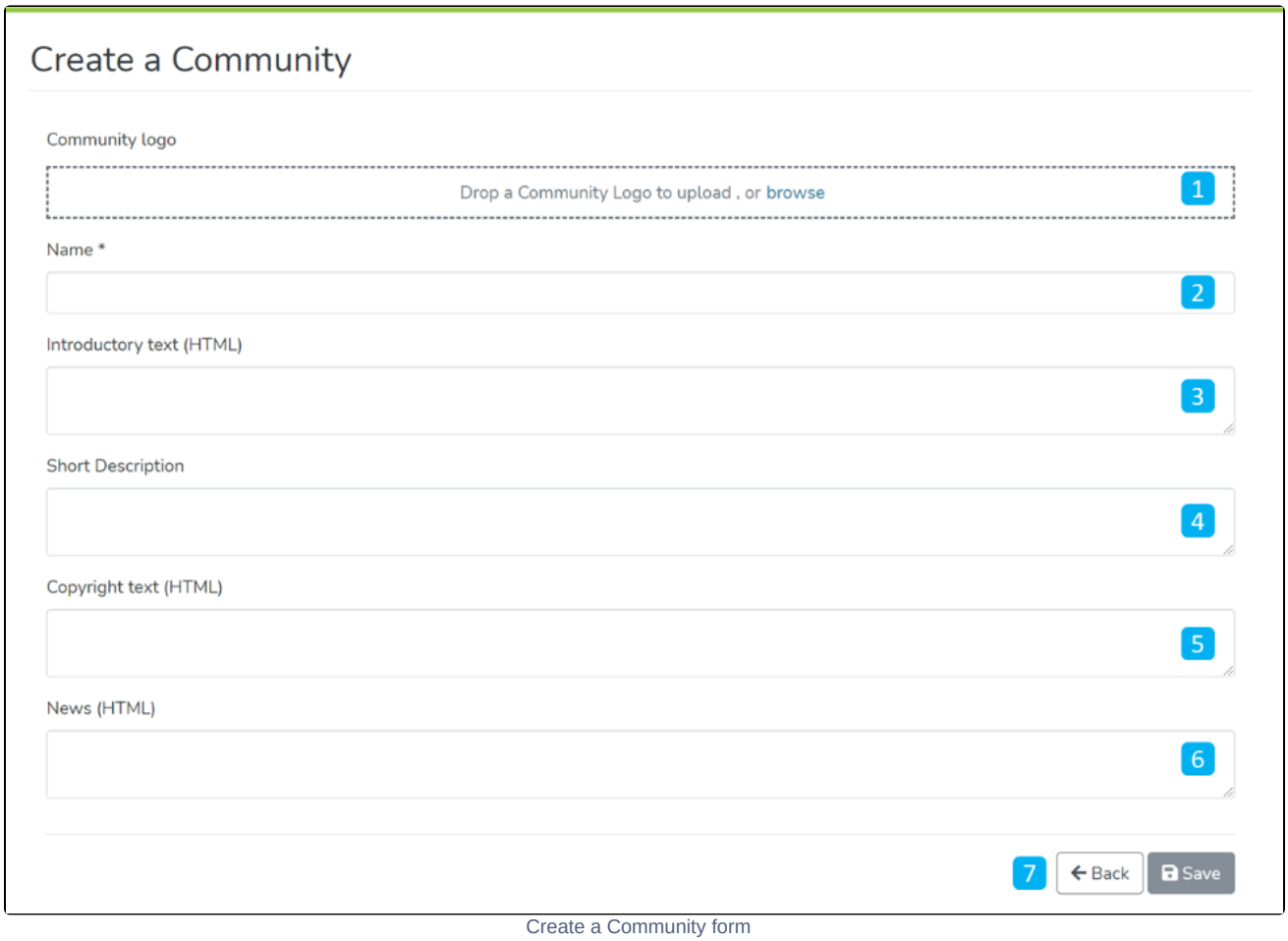


Button to create a new top-level community and typeahead menu to select an existing community

Step 5: As per the user’s choice in the previous step, the application will open the "Create a Community" form or the "Create a Sub-Community for Community [name here]" form to populate information regarding the new community. Below is the explanation of the information that needs to be populated in this form.

The “Create a Community” and “Create a Sub-community” forms are identical. The critical difference is that the “Create a Community” form creates a top-level community while the latter creates a sub-community within a community or sub-community.

The description provided below the following screenshot remains identical for both Community and Sub-Community creation.



Create a Community form

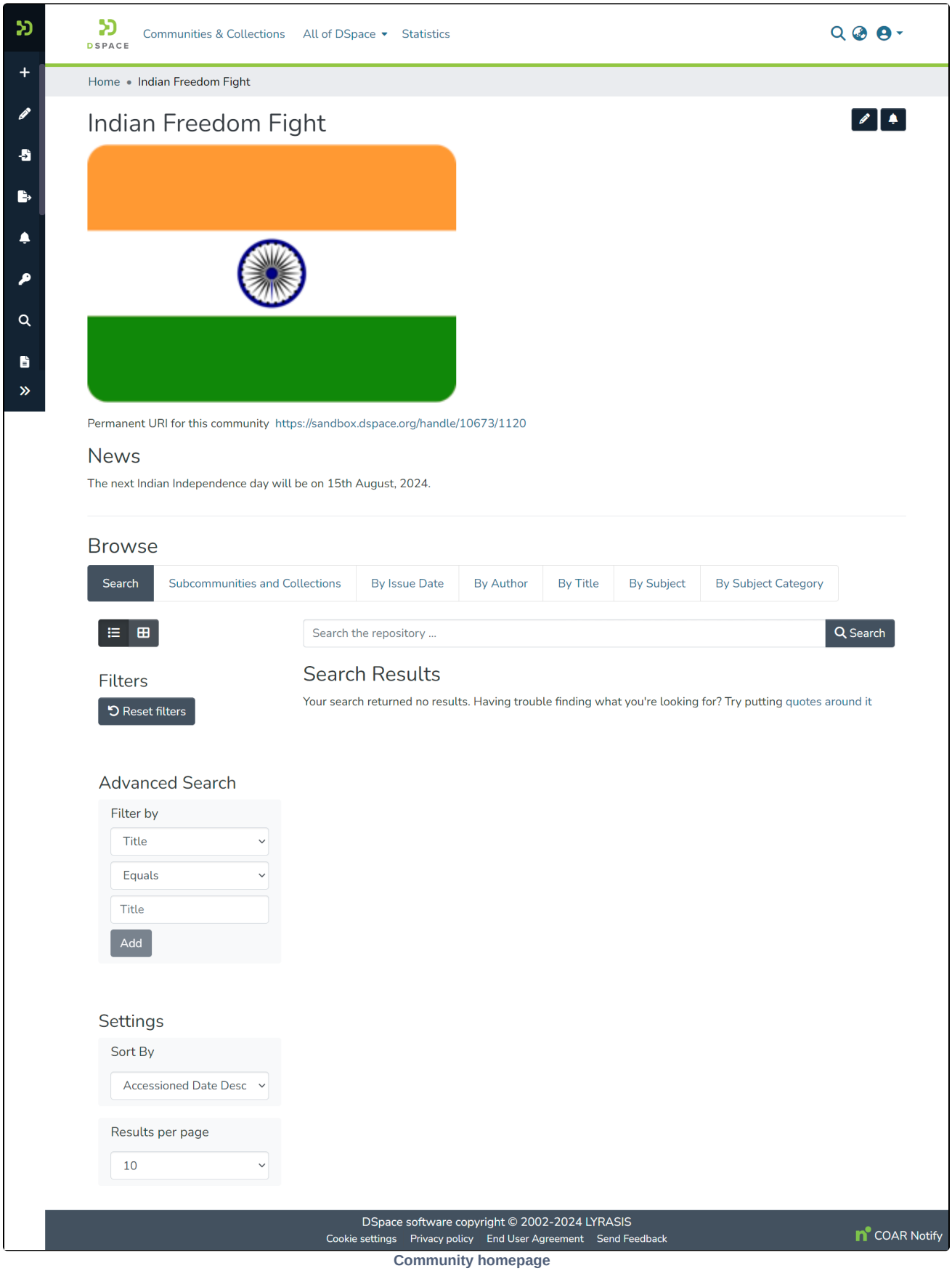
1. **Community logo – Select the community's logo by clicking on the 'browse' link to select an image file. It is advisable to maintain uniform dimensions of the logo across the repository.**
2. **Name – Enter the community's name. It is the only mandatory field, as indicated by the '*.'**
3. Introductory text (HTML) – Users can add introductory text providing an overview of the contents stored in the community. Users can utilize HTML tags to format the text or continue entering the plain text content.
4. **Short Description – This field can have a one-liner description of the community that appears with the community name in the list of communities on the parent community page (or on the DSpace homepage in the case of a top-level community).**
5. **Copyright text (HTML) – Users can enter copyright information here. Users can utilize HTML tags to format the text or continue entering the plain text content.**
6. **News (HTML) – Enter news about this community. Users can update this by regularly going to this section via edit community.**
7. **Action Buttons – Users can click on the appropriate button as desired. Clicking on the Save button will add the community to the repository.**

Step 6: Click on the 'Save' button to complete the Community creation. A success prompt will pop up upon community creation, and the user will be re-directed to the community homepage.



Successfully created the Community banner appears in the upper right corner when the Community is created

Success prompt upon community creation



Community homepage

Community homepage

Edited and approved by DCAT Documentation Working Group, December 2024.

Delete Community

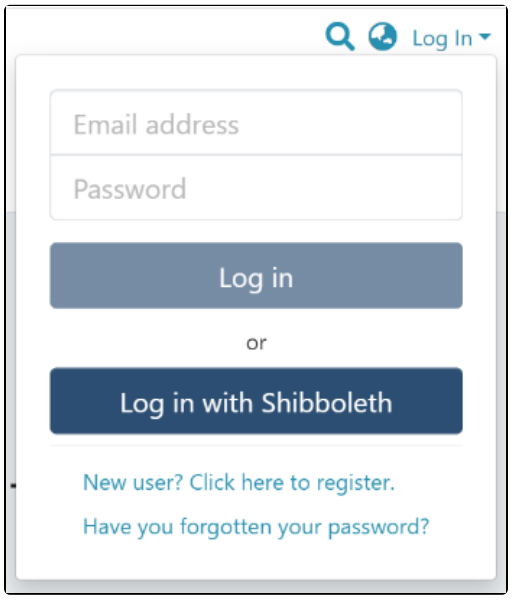
- Audience
- Edit Community

Audience

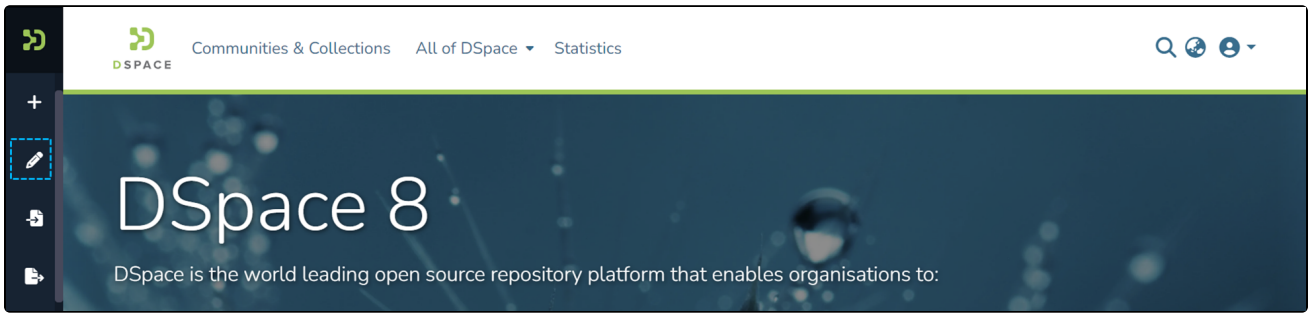
1. Repository Administrator
2. Community Administrator

Edit Community

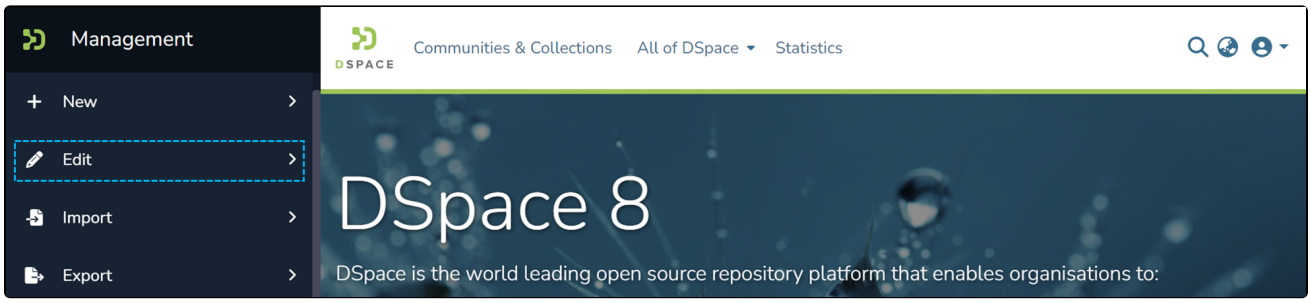
Step 1: Login using your credentials

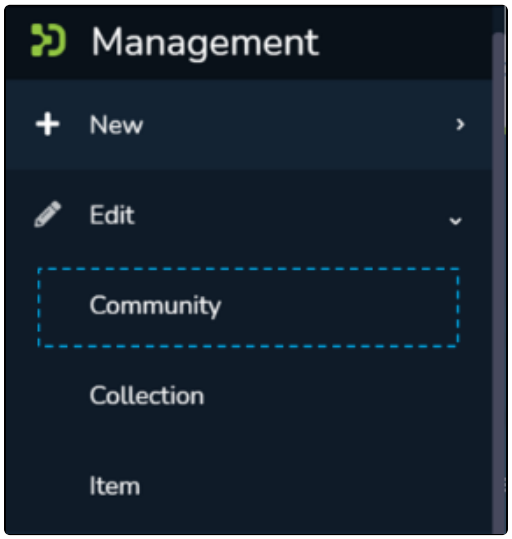


Step 2: There are multiple ways to navigate to the controls to delete a community. One of them is by going to the target community and clicking on the button with the pencil icon next to the community title ie the ‘Edit community’ button. Alternatively, follow the steps provided here.

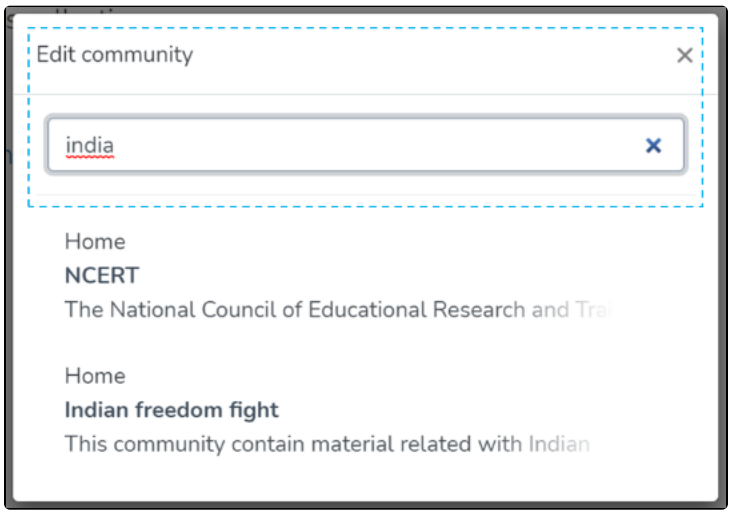


Step 3: Click on “Edit” and click on “Community” to proceed with the edit community process.

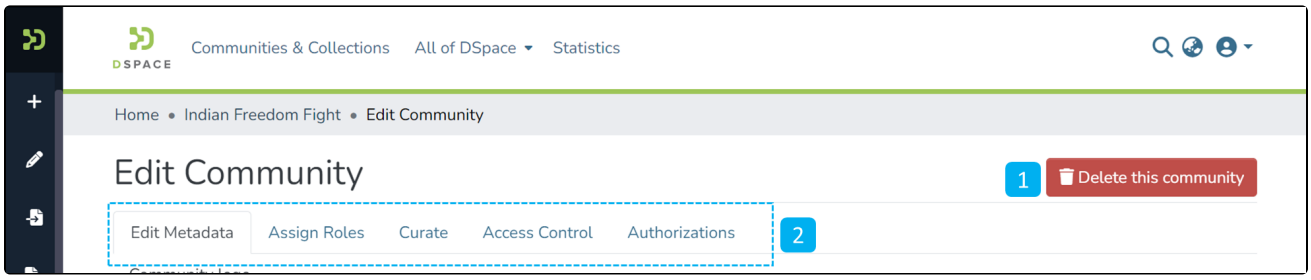




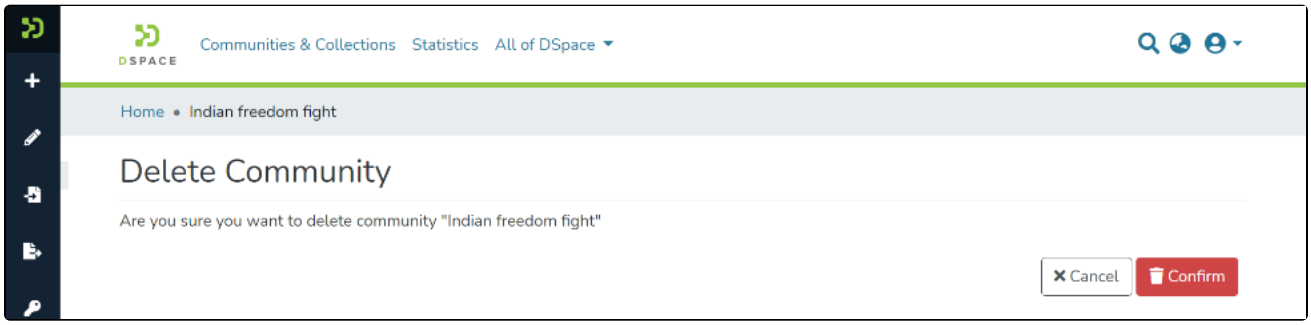
Step 4: A popup showing a search box and a list of communities will appear. Type the name of the community you want to edit in the search field. Upon typing a few letters of the community's name, a list of the communities having those letters or words will appear. Click on the target community to initiate editing.



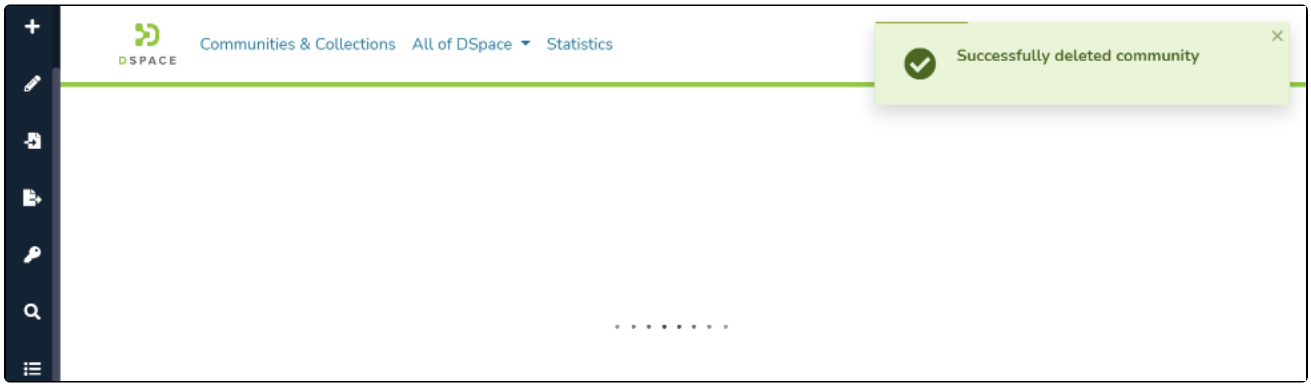
Step 5: The application will take the user to the edit community form. To initiate the community deletion process, the user needs to click on the 'Delete this community' button.



Step 6: Click on the Confirm button to continue with the community deletion or click on the Cancel button to return to the previous page.



Users will be redirected to the homepage of DSpace upon successful completion of the community deletion, and a popup confirming the community deletion will appear.



Edit Community

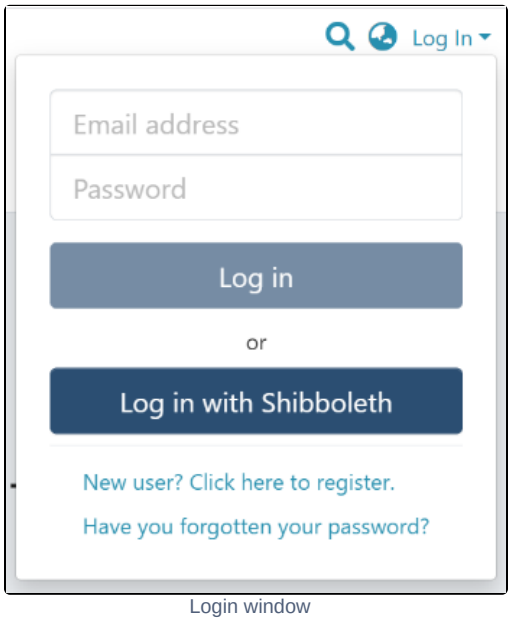
- Audience
- Edit Community
- Edit Metadata
- Assign Roles
- Curate
- Access Controls
- Authorizations

Audience

1. Repository Administrator
2. Community Administrator

Edit Community

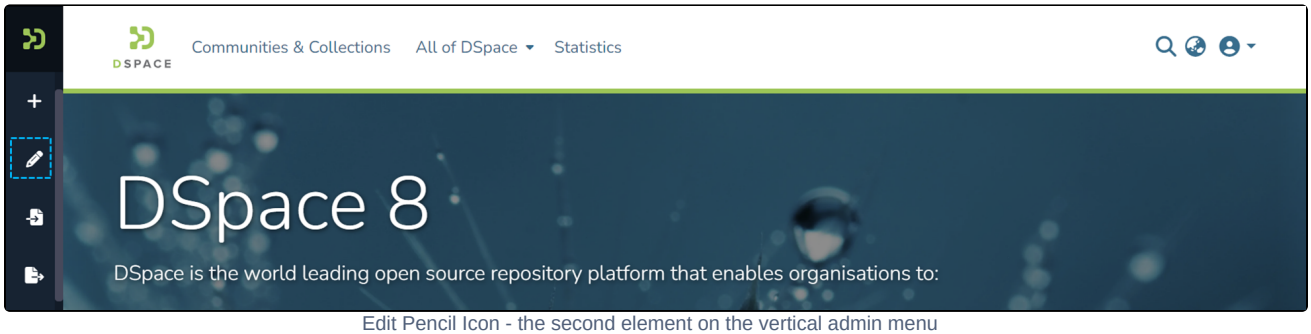
Step 1: Login using your credentials



Login window

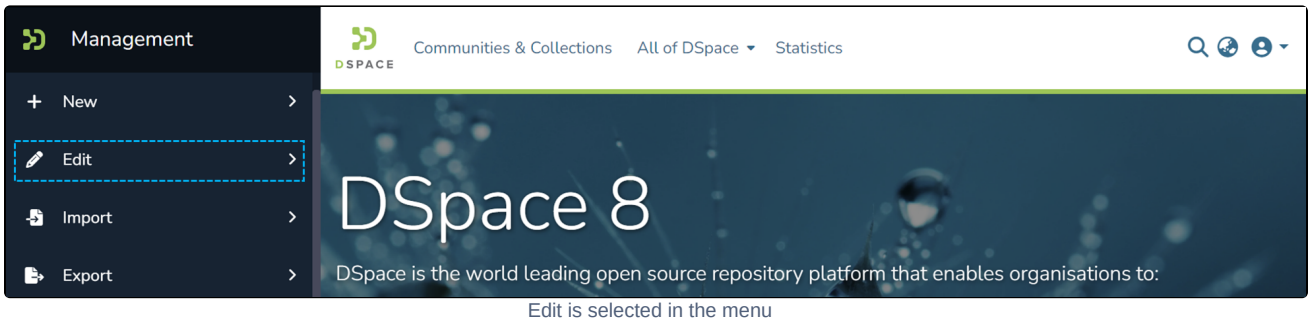
Step 2: There are multiple ways to start editing a community. One of them is by going to the target community and clicking on the Edit button, the button with the pencil icon, beside the page title. Alternatively, follow the steps provided here.

Rollover cursor on the pencil icon in the admin menu.

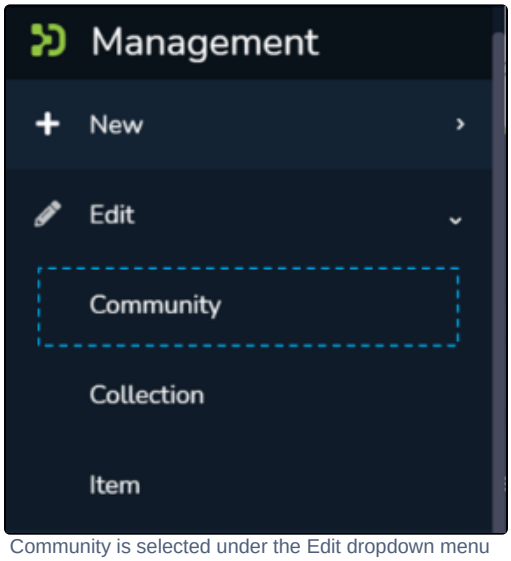


Edit Pencil Icon - the second element on the vertical admin menu

Step 3: Click on “Edit” and click on “Community” to proceed with the edit community process.

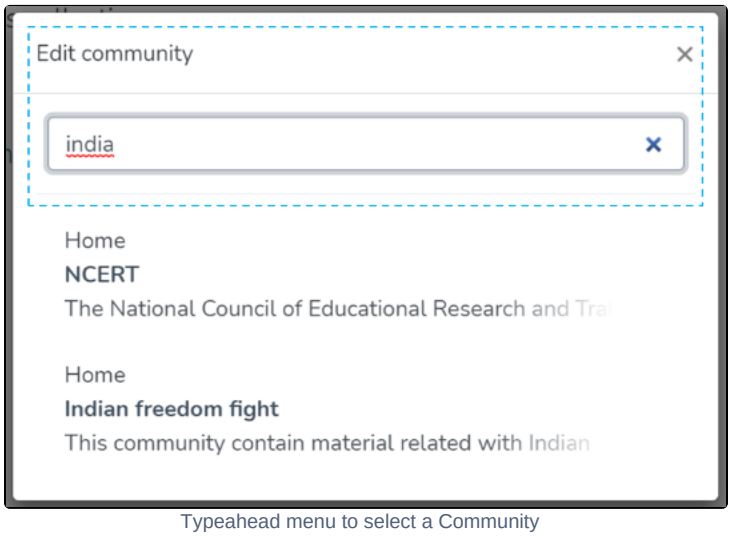


Edit is selected in the menu



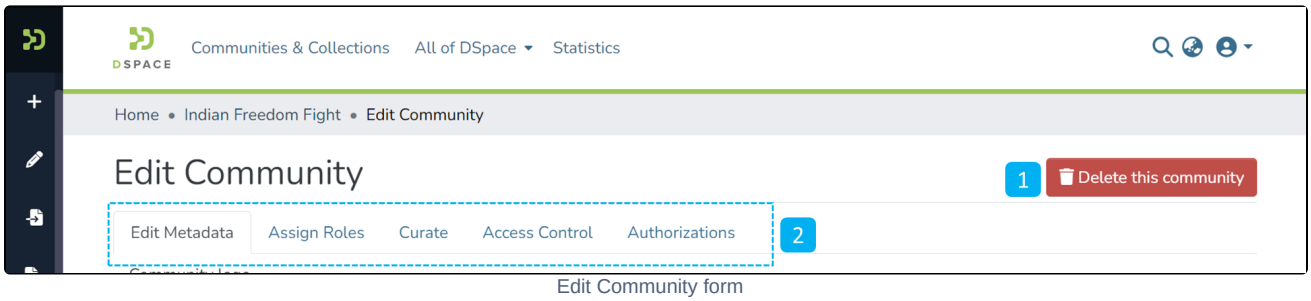
Community is selected under the Edit dropdown menu

Step 4: A popup showing a list of the communities in the DSpace and a search box will appear. If you are already on the page of the community, it will appear at the top of the list, so you can select it by clicking on it. Otherwise, type the name of the community you want to edit in the search field. Upon typing a few letters of the community's name, a list of the community(ies) having those word(s) will appear. Click on the target community to initiate editing.



Typeahead menu to select a Community

The application will take the user to the edit community form to perform various actions to edit the community. Each tab is explained in a separate process in this document.



Edit Community form

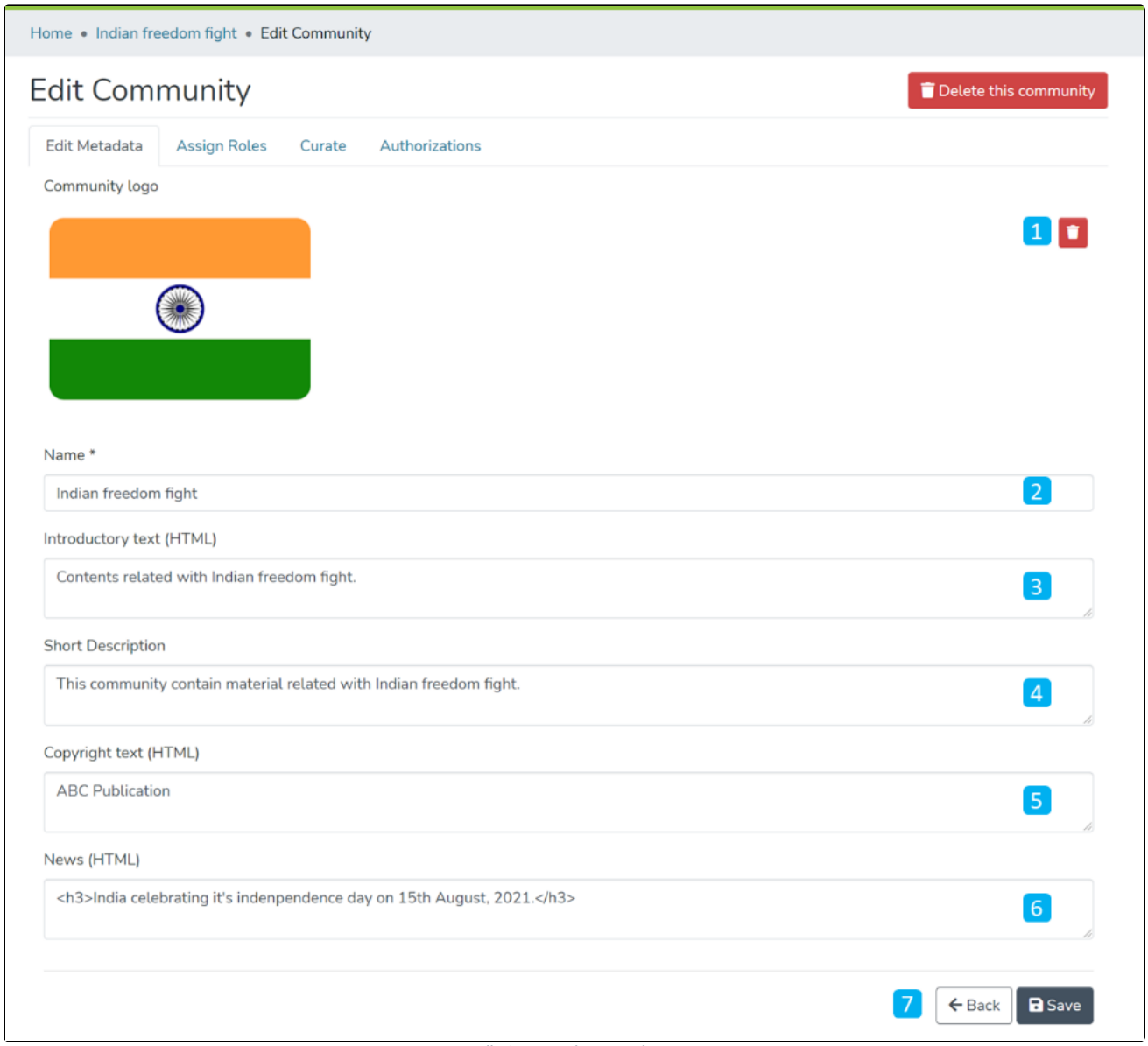
1. Delete this community – The button is provided for deleting the community. Detailed steps are explained in the latter part of this page.
2. Tabs – The edit community has a variety of functions, which are grouped logically across various tabs. Below is the summary of these tabs

    a. Edit Metadata – The tab covers activities related to editing the community's profile information.
    b. Assign Roles – This tab allows users to create specific roles for the community, usually, the role of Administrator of the community, see further detail below.
    c. Curate – Users can set up various workflows related to content curation in this tab
    d. Access Control – The tab allows users to perform changes to the access conditions of all the items owned by the community. Changes may be performed to either all Item metadata or all content (bitstreams).
    e. Authorizations – Under this tab, users can manage various groups and their different access rights in the community, for example, this tab could be used to grant an individual the administrator role, see further detail below.

Edit Metadata

The Edit Metadata tab allows users to update the community's profile-related information, a.k.a. community metadata.

Various actions on this tab are explained immediately after the Edit Metadata illustration is added below.



Edit Community metadata

1. Community logo – Click on the delete button to remove the existing logo. If no logo exists, then a widget allowing the user to add a logo is displayed here.
2. Name – Update the existing community's name in this field.
3. Introduction text (HTML) – Update introductory text if already added or add new text. One can utilize HTML tags to format the text or continue entering plain text content.
4. Short Description – Update the description of the community or add a fresh short description for the community.
5. Copyright text (HTML) – Update copyright-related information in this field. This is usually displayed at the foot of the community landing page. Fields marked with (HTML) support HTML tags-based formatting.
6. News (HTML) – Add/Update news specific to this community in the field. This is usually displayed with the heading 'News', underneath the community's introductory text, and above the list of collections and sub-communities.
7. Action Button – Clicking on the Save button will update the metadata information for the community.

Click on the 'Save' button to save the information updated in the 'Edit Community' tab. A success prompt will appear, confirming the successful edit of the community.

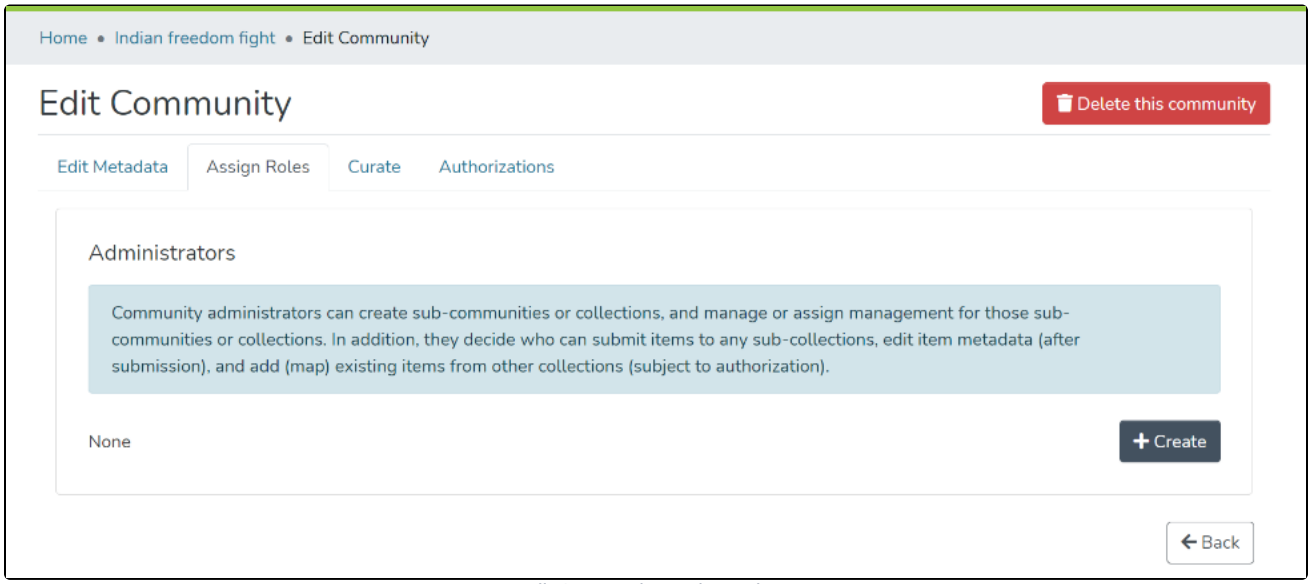


Successfully saved banner appears when the changes are saved in the upper right corner.

Assign Roles

This tab allows authorized users to create a Community administrator role. Click on the “create” button to assign a community administrator role.

The roles available on this tab are explained below this illustration.



Edit Community Assign Roles

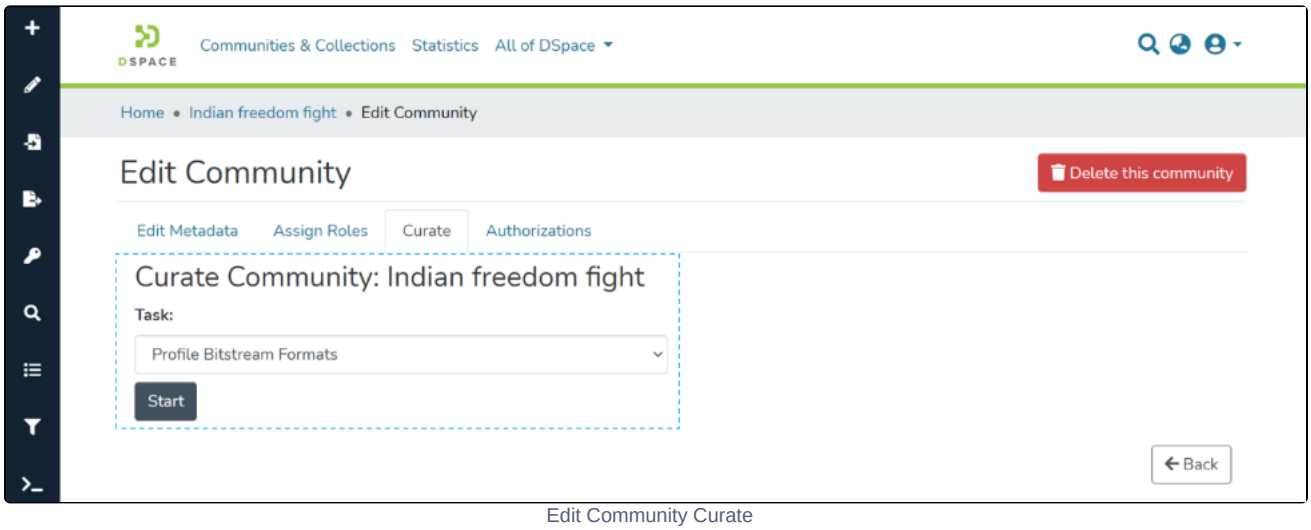
Administrators - Community administrators can create and manage sub-communities and collections. This user profile can also assign rights to edit item metadata and map existing items from other collections.

Curate

This tab provides various workflows for curating items stored in the community. Below are standard flows, and there can be customized curation workflows as well

1. Profile bitstream formats
2. Check for Required Metadata
3. Check Links in the Metadata

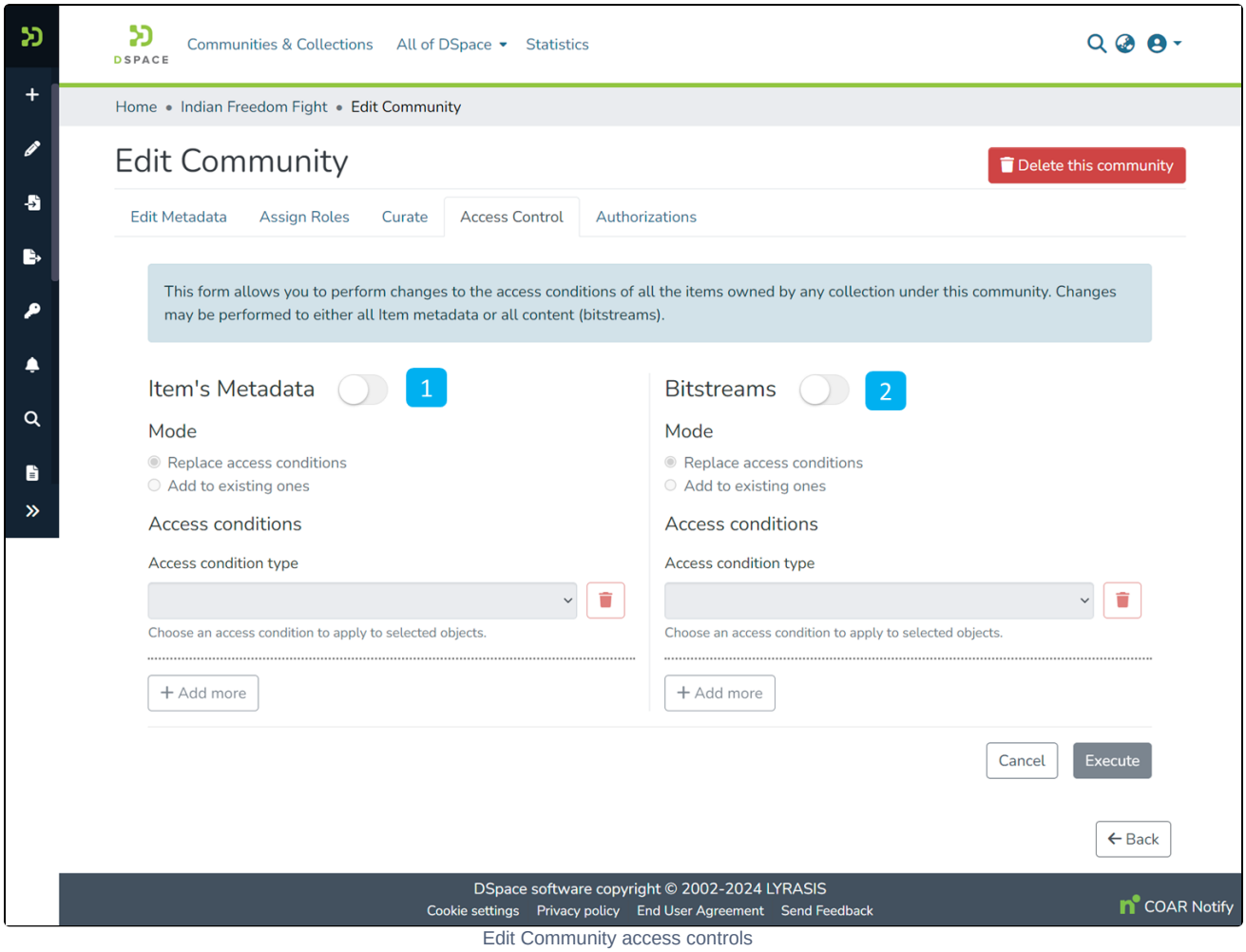
Users must select a workflow from the dropdown list and click the “Start” button to initiate the curation process.



Edit Community Curate

Access Controls

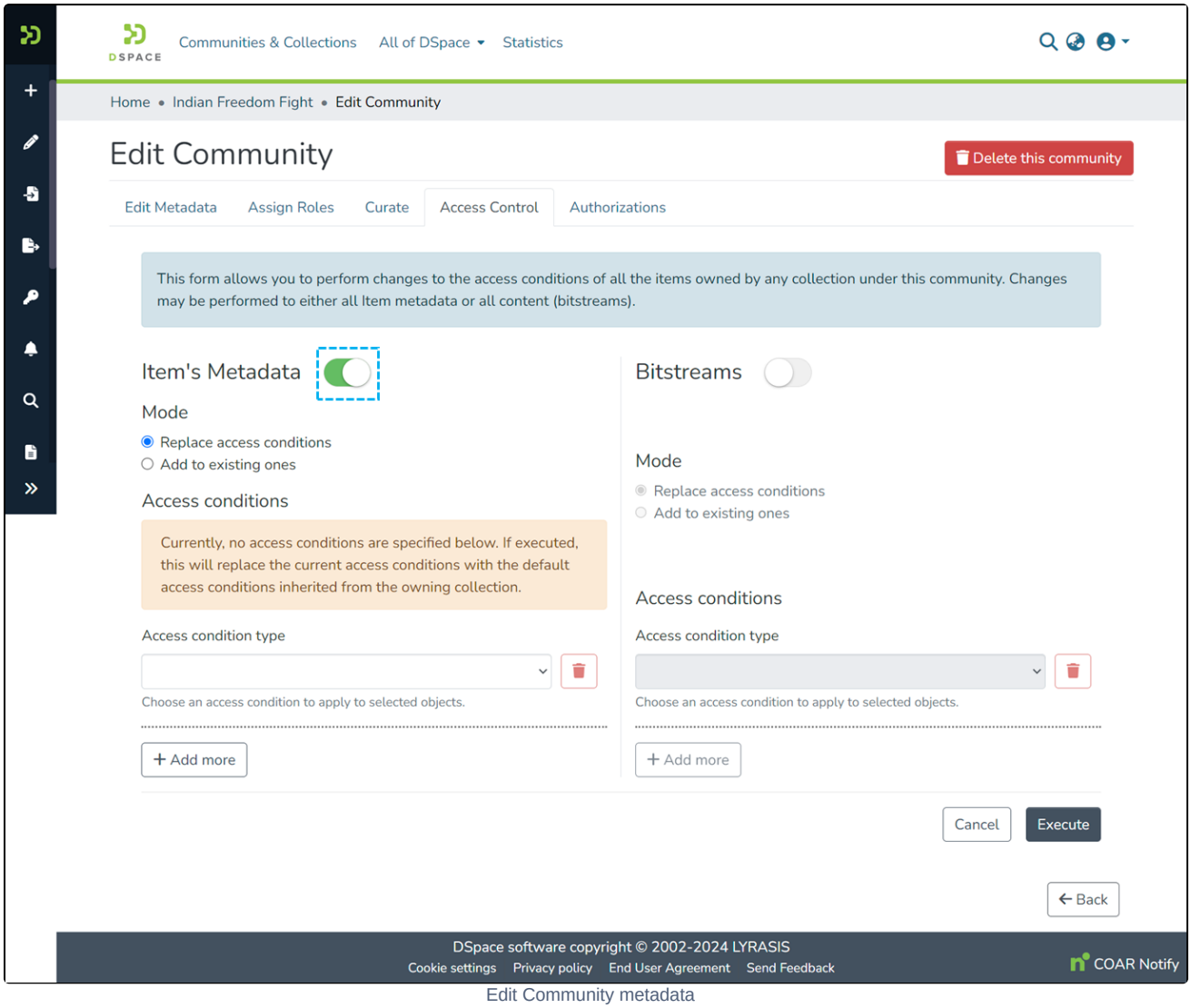
The section allows the user to change access conditions of all the items owned by the community. Changes may be performed to either all Item metadata or all content (bitstreams).



Edit Community access controls

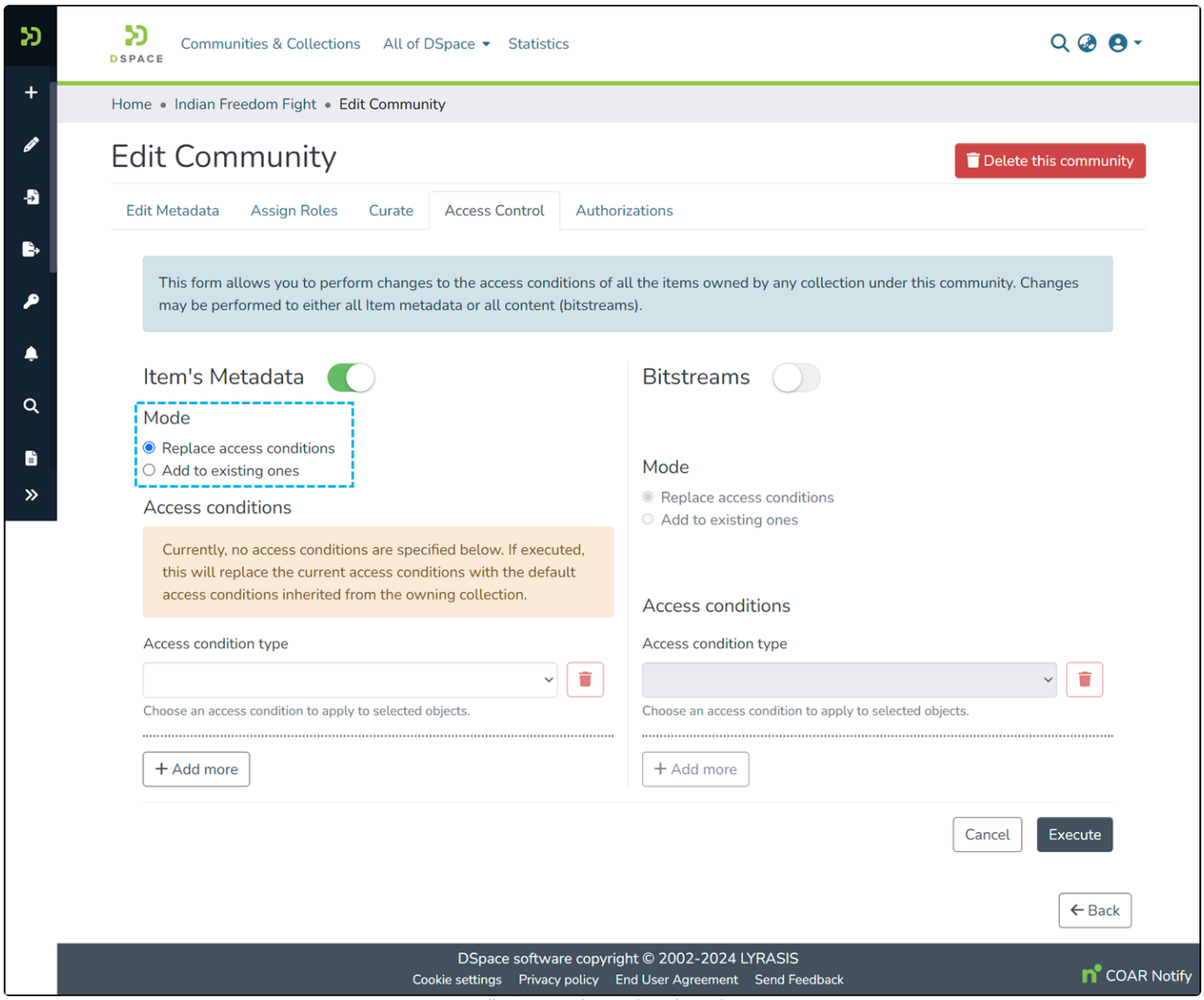
1. Item's Metadata – Click this option to manage access rights on metadata of items stored in the collection.
2. Bitstreams – Click this option to manage access rights on bitstreams (attachments) of items stored in the collection.

Click the switch next to the Item's Metadata header to initiate changes to the access rights on the metadata of items.



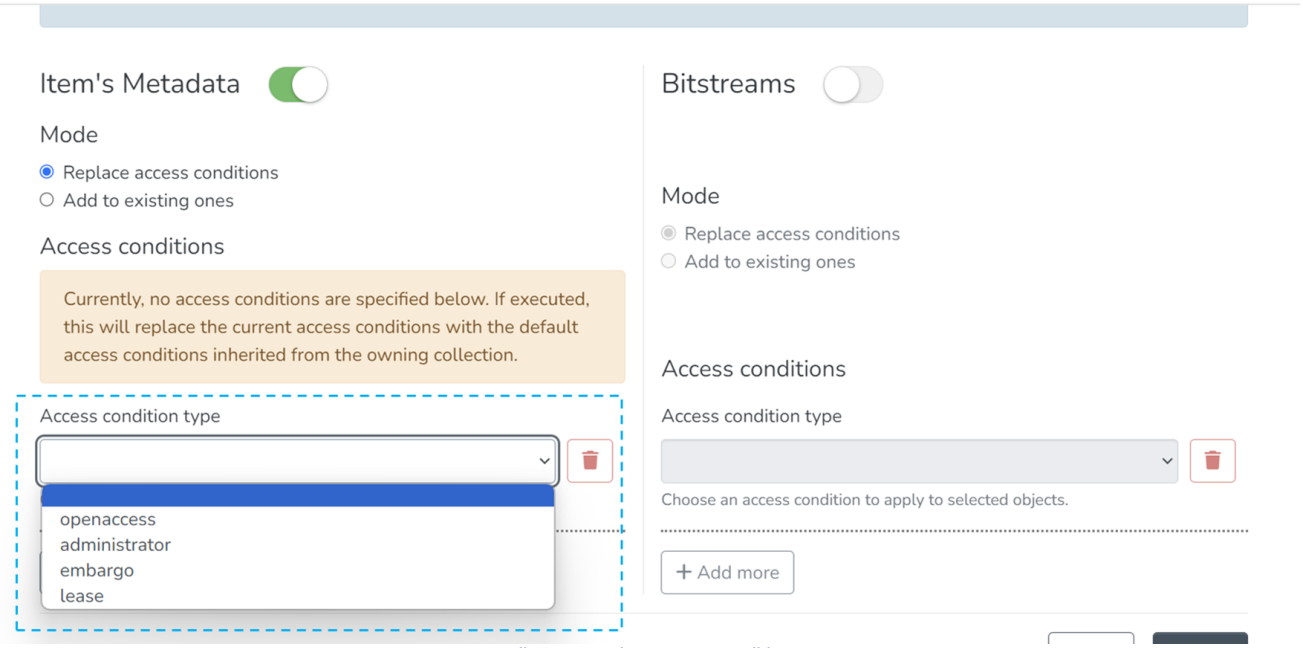
Edit Community metadata

Select a Mode from the following options as per update requirements.



Edit Community Mode selected

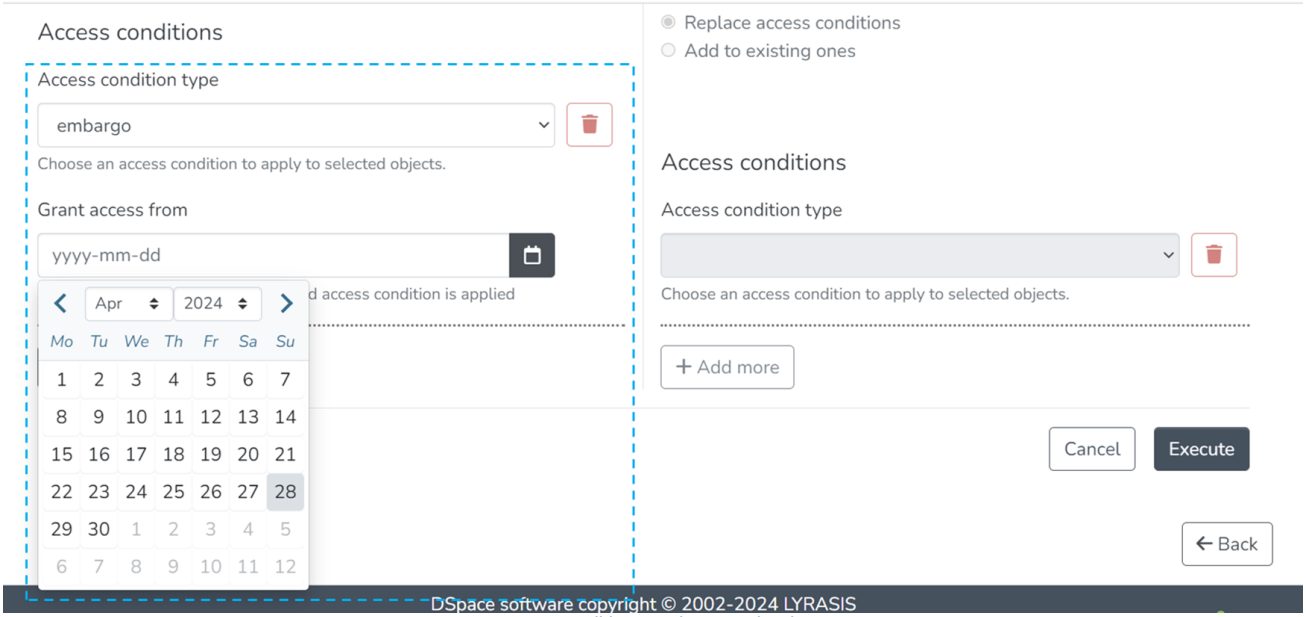
Select the access condition type from the drop-down list as required.



Edit Community access condition

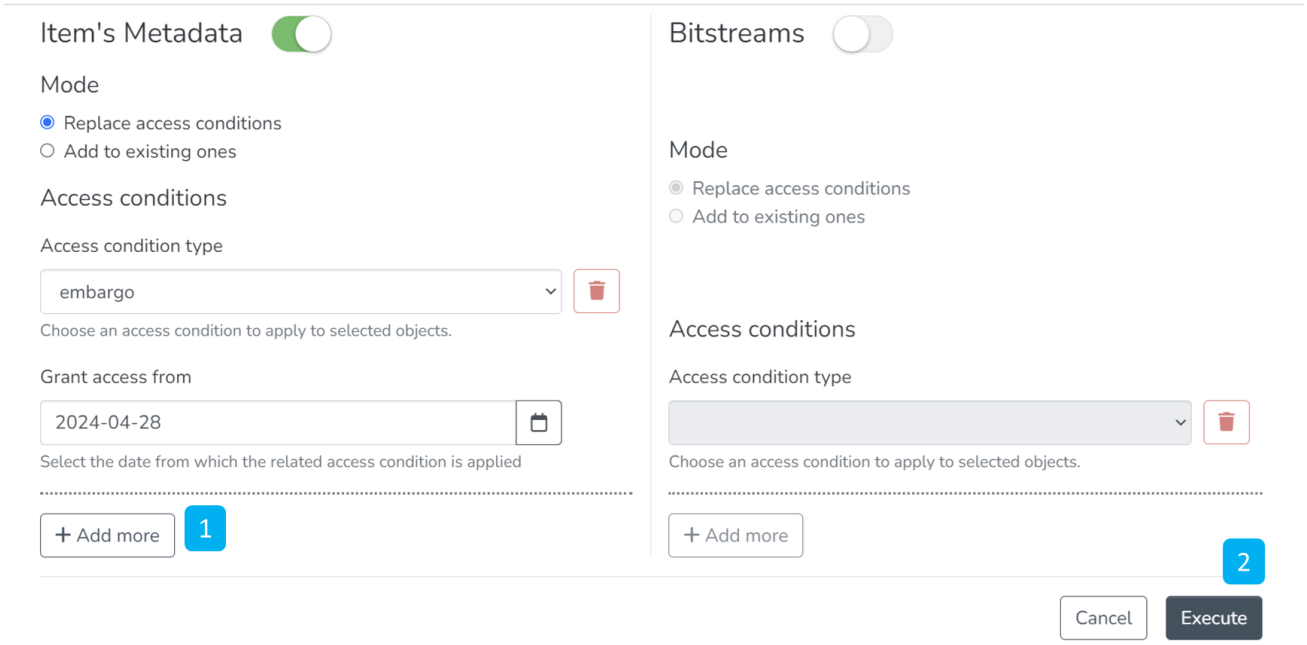
1. openaccess – Select the option to make the Item's metadata available to everyone.
2. administrator – This option will limit items' access to the administrator user group.
3. embargo - Embargo will restrict access to items until the selected future date, as explained in the following steps.
4. lease - The lease will restrict access to items after the selected future date, as explained in the following steps.

Select the 'Embargo' from the drop-down, and in the following field, select an embargo date to limit access to items until the selected date.

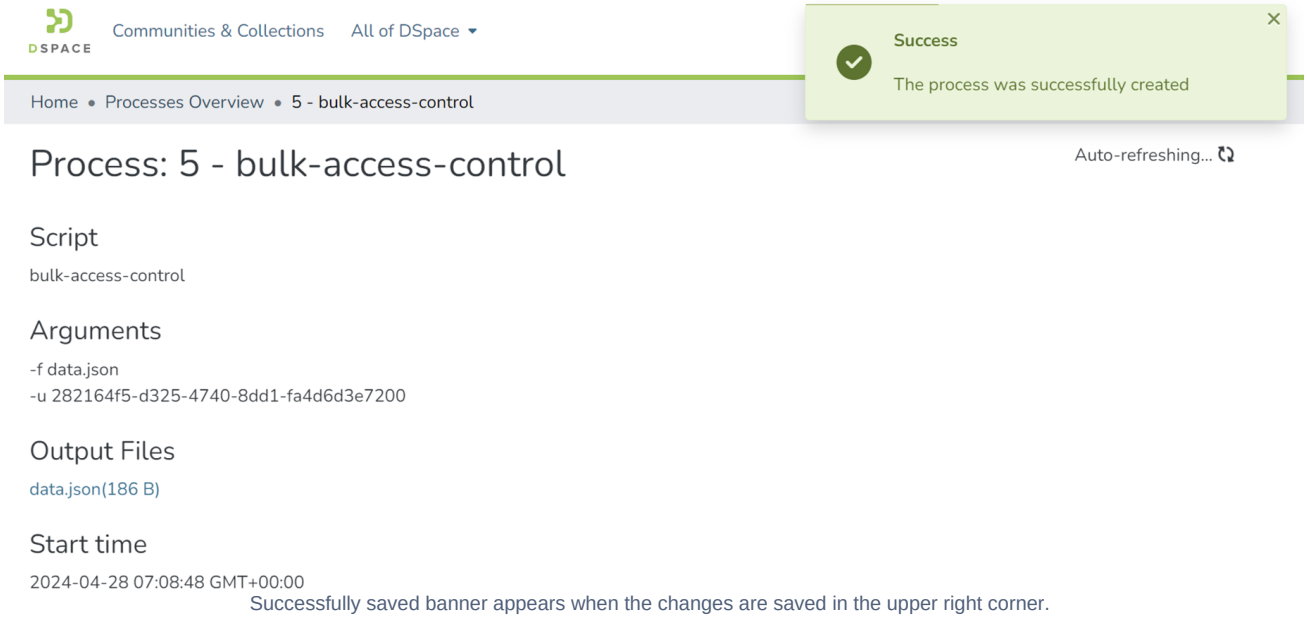


Access conditions embargo calendar

Click the 'Add more' button to add another policy by repeating the above steps, and click the 'Execute' button to apply policies to the community.

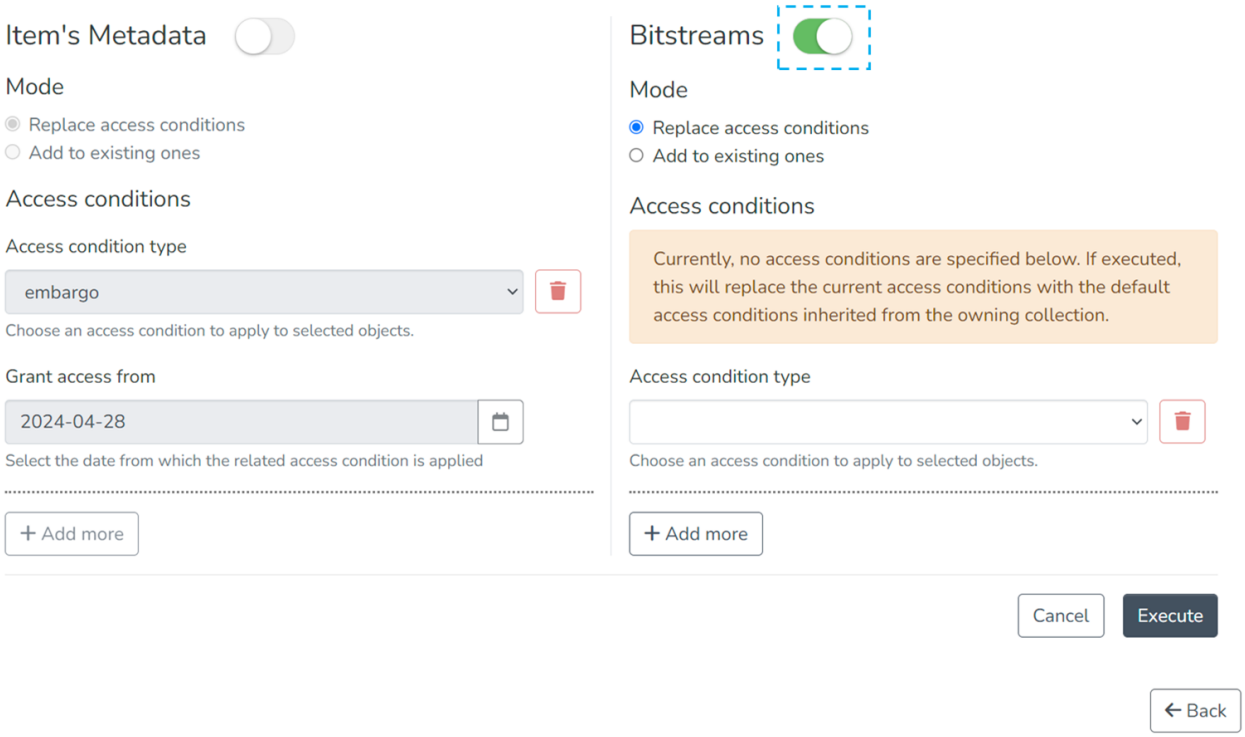


Users shall see the following screen upon the successful execution of policies.



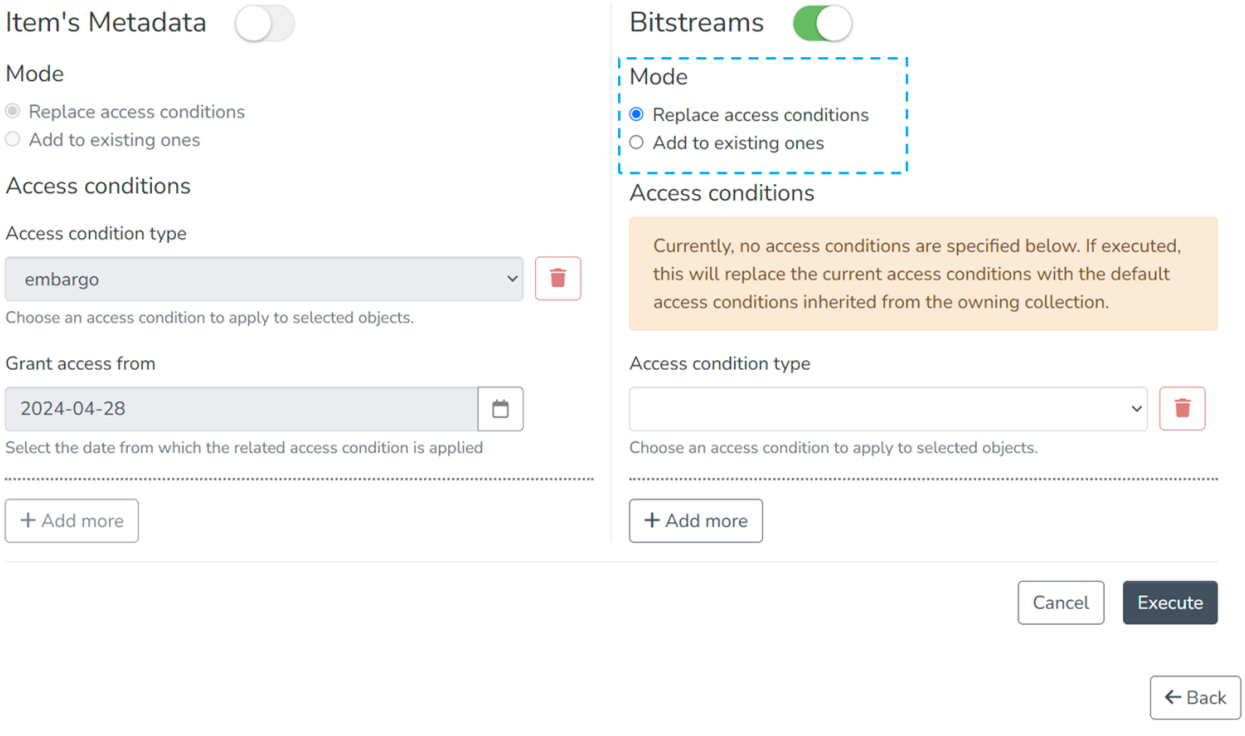
Successfully saved banner appears when the changes are saved in the upper right corner.

Click the switch next to the Bitstream header to initiate changes to the access rights on the bitstreams of items.



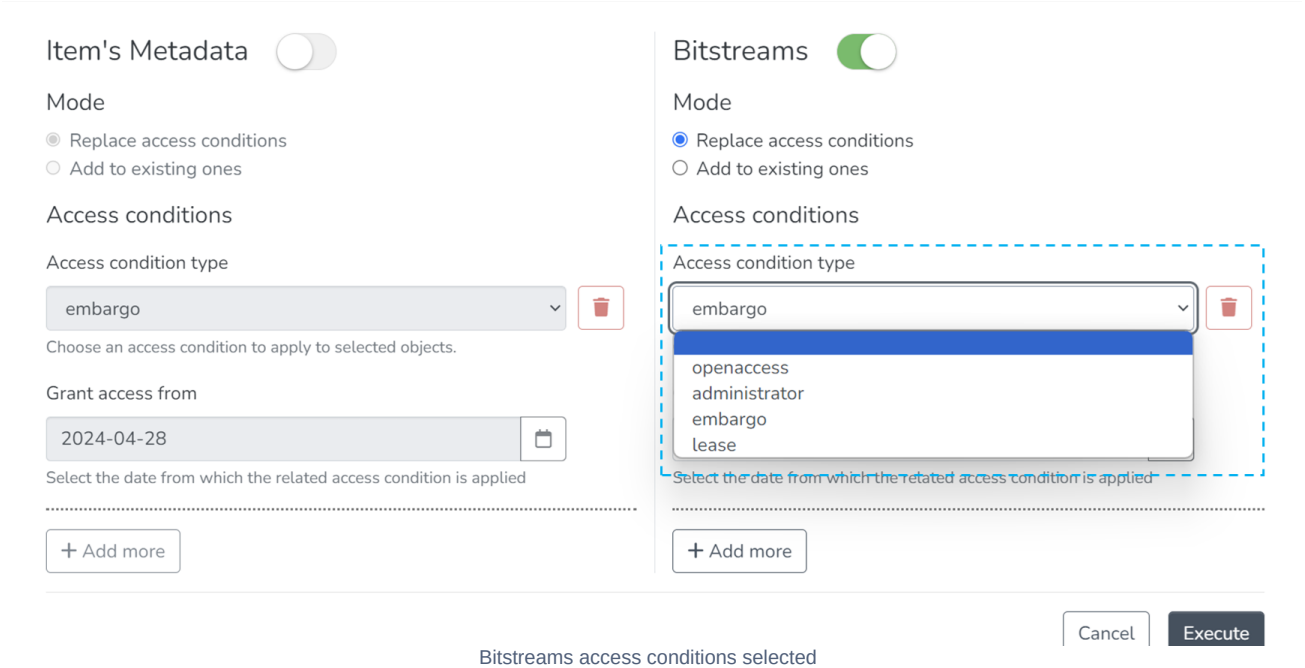
Bitstreams selected

Select a Mode from the following options as required.



Bitstreams Mode selected

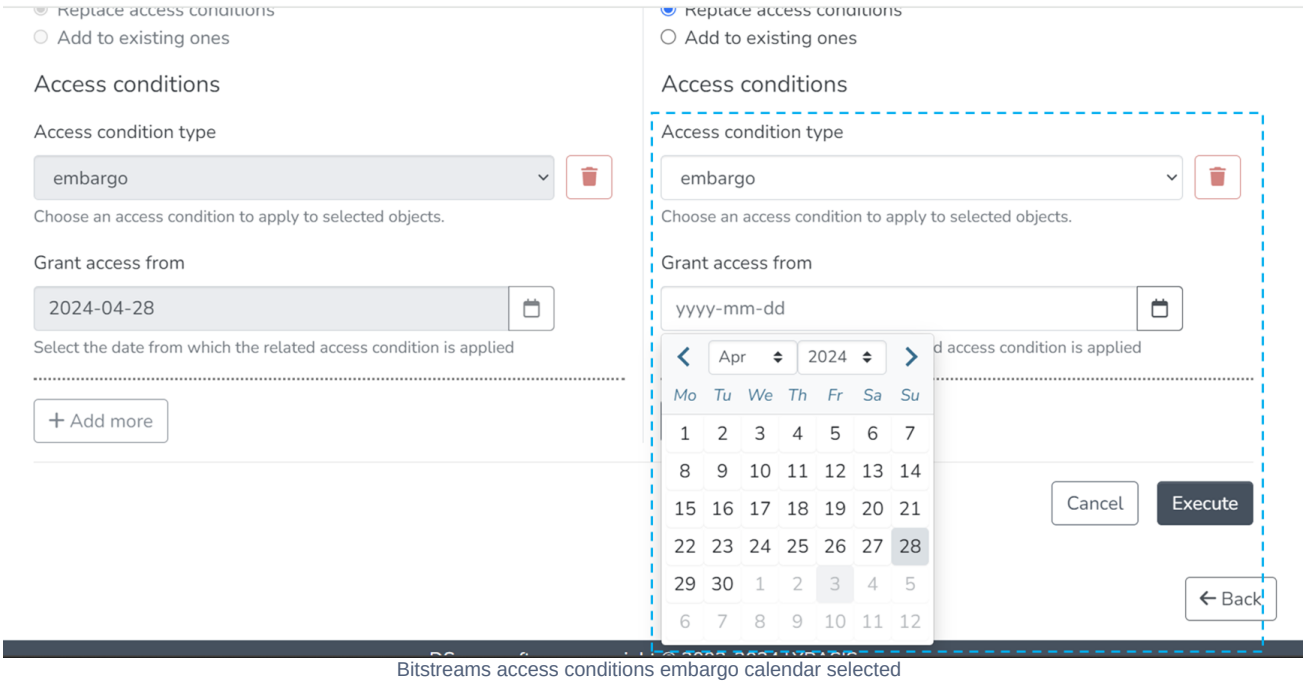
Select the access condition type from the drop-down list as required.



Bitstreams access conditions selected

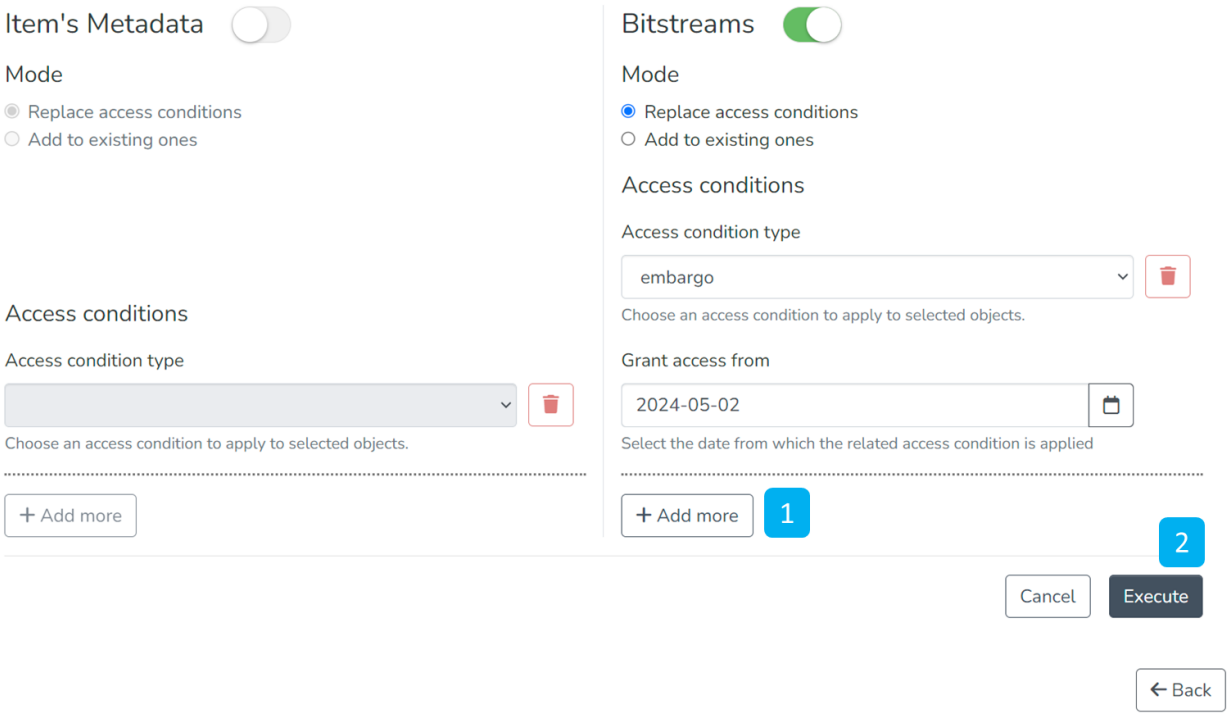
1. openaccess – Select the option to make the bitstream's metadata available to everyone.
2. administrator – This option will limit bitstream access to the administrator user group.
3. embargo - Embargo will restrict access to bitstreams until the selected future date, as explained in the following steps.
4. lease - The lease will restrict access to bitstreams after the selected future date, as explained in the following steps.

Select the 'Embargo' from the drop-down, and in the following field, select an embargo date to limit access to bitstream until the selected date.

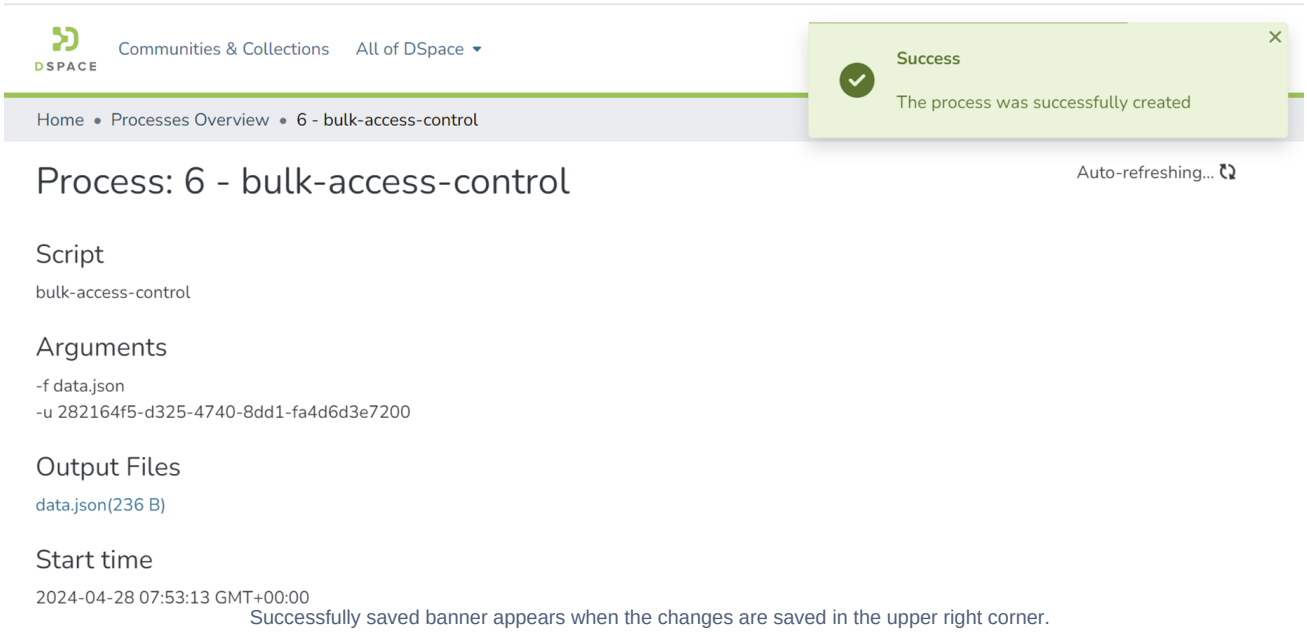


Bitstreams access conditions embargo calendar selected

Click the 'Add more' button to add another policy by repeating the above steps, and click the 'Execute' button to apply policies to the community.



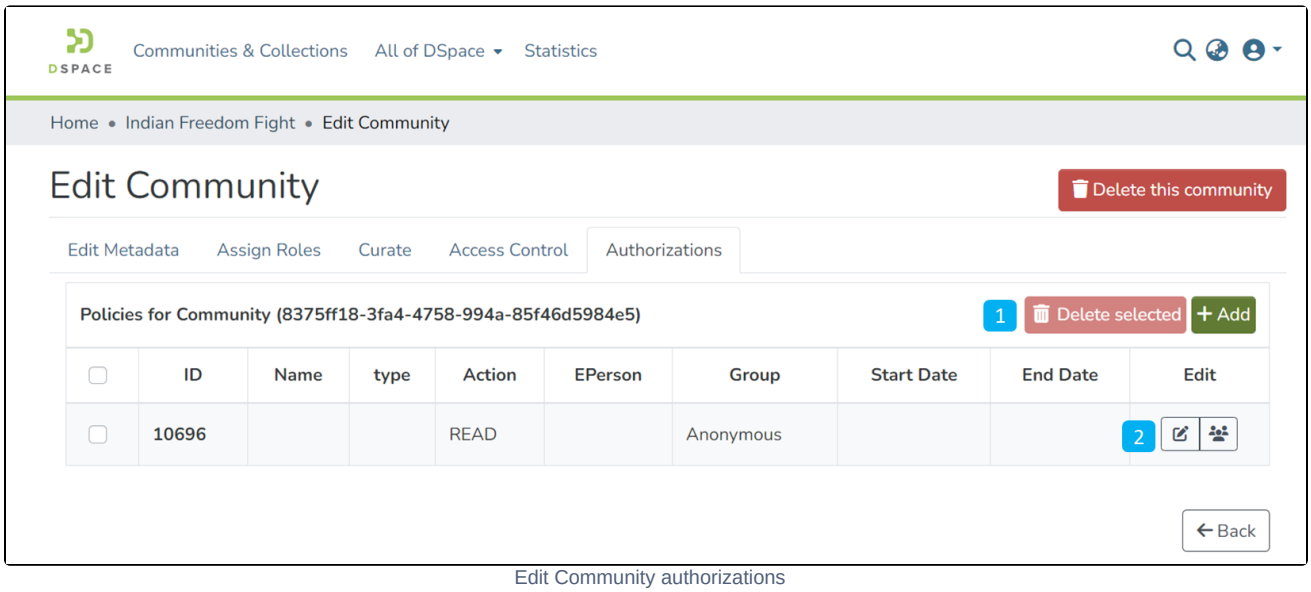
Users shall see the following screen upon the successful execution of policies.



Successfully saved banner appears when the changes are saved in the upper right corner.

Authorizations

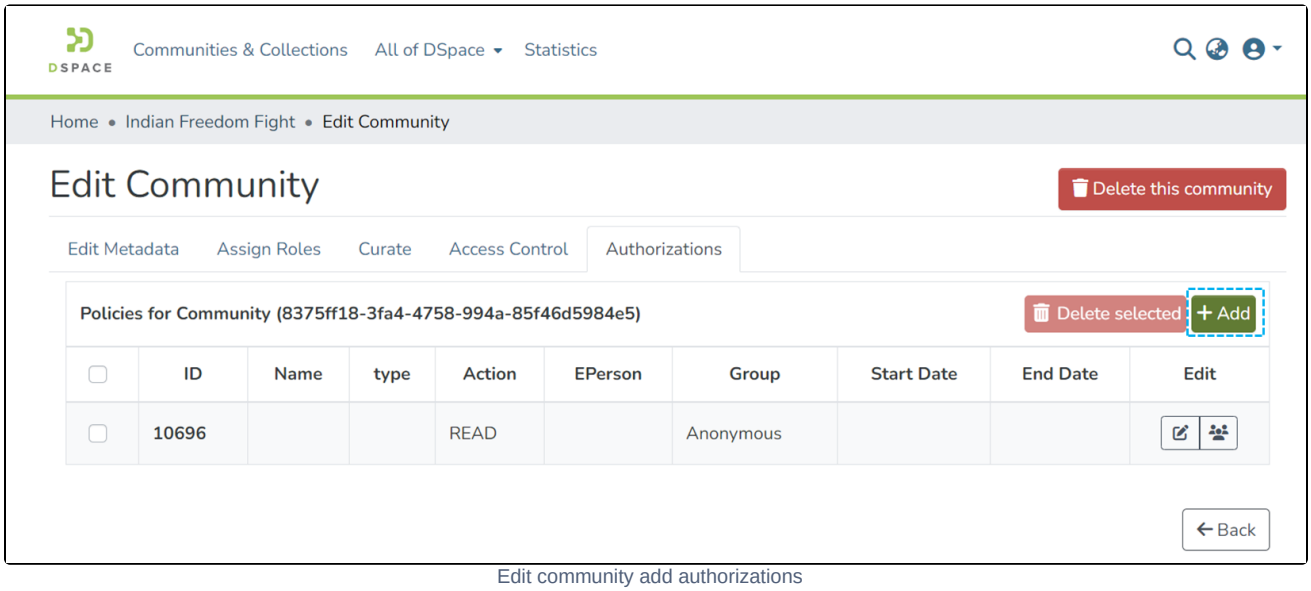
Users can view and edit community resource policies defined for the community, in the Authorizations tab. Users can create policies in addition to the standard policies created from the Assign Roles tab. Following are the key actions in this tab.



Edit Community authorizations

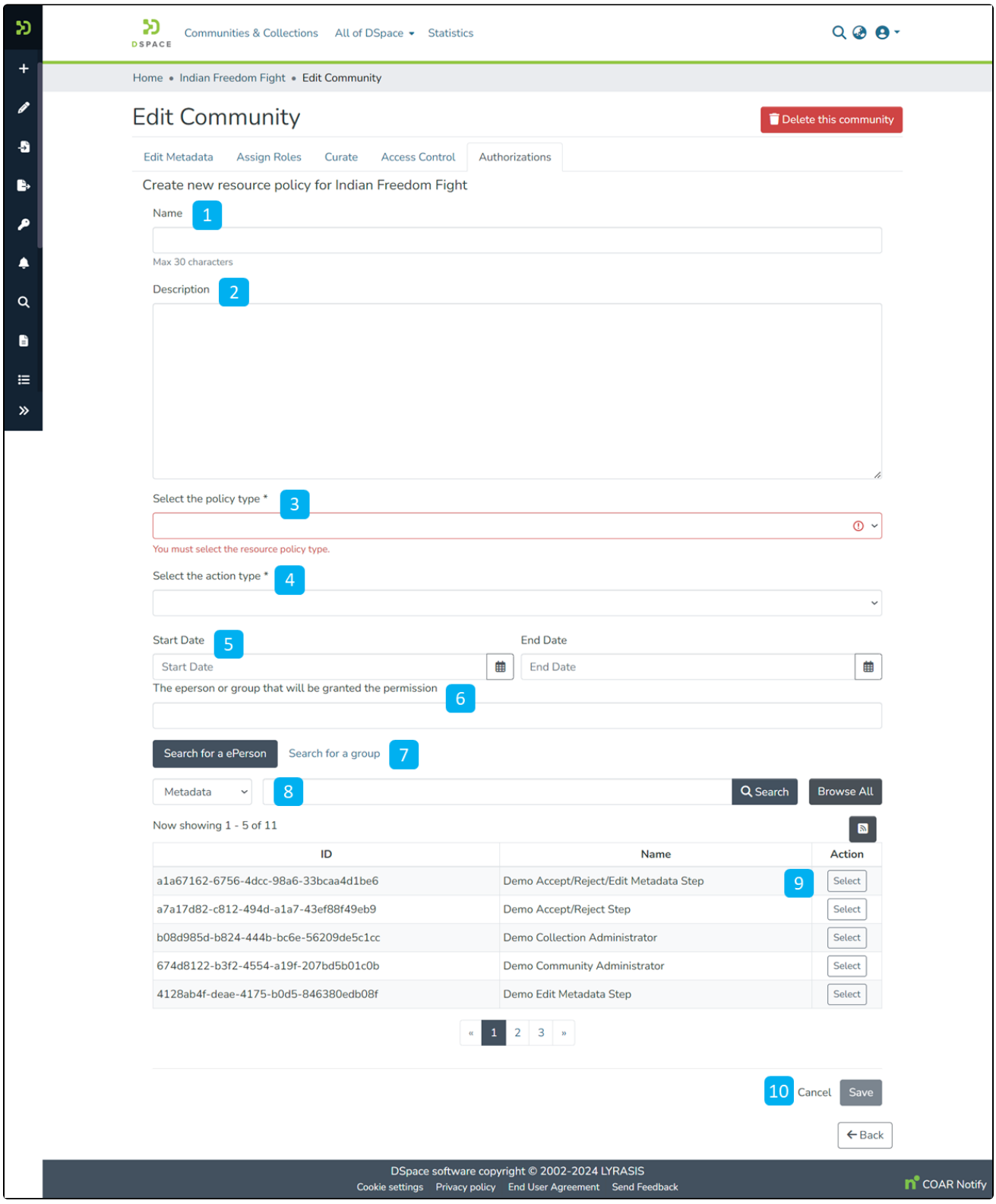
1. Manage Policies
   – Click on the Add button to create a new resource policy or select policies from the table, see further detail below.
   - Alternatively, click on the Delete selected button for a batch deletion of the policies.
2. Edit policy and members in a policy – Click on the edit button to edit an individual policy or click on the group icon to edit the user group eg to add or remove individual ePersons.

Click on the Add button to create a new Authorization policy



Edit community add authorizations

Users can enter the information to create the policy and click on the submit button. Please see the description of each field followed by the below screenshot.



1. **Name:** Enter the Policy name in this field.
2. **Description:** Enter the Policy description here for future reference and understanding of other users.
3. **Select the policy type:** The user can select one of the following policy classification types from the list
    a. TYPE_SUBMISSION: a policy in place during the submission
    b. TYPE_WORKFLOW: a policy in place during the approval workflow
    c. TYPE_INHERITED: a policy that has been inherited from a container (the community)
    d. TYPE_CUSTOM: a policy defined by the user during the submission or workflow phase
4. **Select the action type:** The user can select one of the following actions from the dropdown list. For example, select “READ” to assign read rights to the user or user group.:

    a. READ - Ability to view the specific object
    b. WRITE - Ability to edit the specific object
    c. ADD - Ability to add an object to a parent object (e.g. add a Bitstream to a Bundle, add a Bundle to an Item, add an Item to a Collection , add a Collection to a Community, etc.)
    d. REMOVE - Ability to remove an object from a parent object (i.e. the opposite of "ADD")
    e. ADMIN - Full Admin permissions on the specific object
    f. DELETE - Ability to delete the specific object
    g. WITHDRAWN_READ - Ability to view a withdrawn Item (only valid for Items)
    h. DEFAULT_BITSTREAM_READ - Default "READ" permissions added to any Bitstreams of a newly submitted Item (only valid for Collections)
    i. DEFAULT_ITEM_READ - Default "READ" permissions added to any newly submitted Item (only valid for Collections)
5. **Start date – end date:** The user can select the start date and end date of the period for which the policy will be active, should they want to apply this policy for a fixed period only. If the start date is left blank, the policy comes into effect immediately.
6. **The ePerson or group that will be granted the permission:** List of users/groups selected for granting permission under the policy
7. **Search for an ePerson / Search for a group:** Select ePerson or group to add
8. **Search field:** Enter keywords for searching the ePerson/Group
9. **ePerson/Group list:** Click on the select button against the user/group you want to add to the policy
10. **Submit/Cancel button:** Click on the Submit button to complete policy creation or click on the Cancel button to cancel the entire process.

Upon successfully creating the policy, you'll see a confirmation prompt, and the user will be back on the Authorizations screen.

Export Metadata by Collection or Community

- Scope and Use Case
- Audience
- Exporting a collection

Scope and Use Case

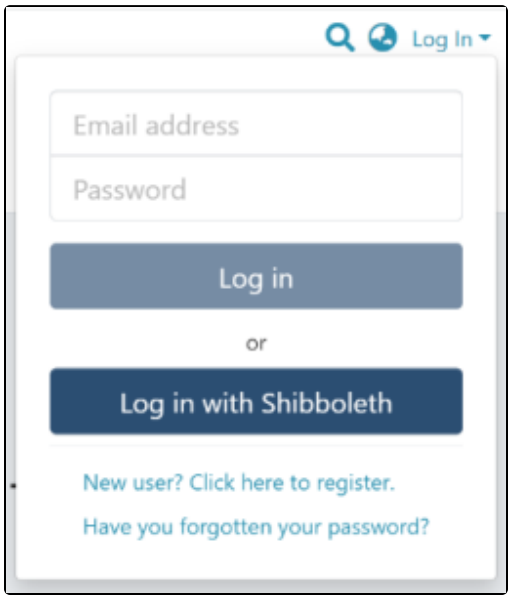
DSpace provides a feature of exporting the metadata of all items within a collection or community into CSV format. Users can utilize this CSV file for multiple purposes like creating ad-hoc reports, importing metadata into other systems, or for any use case as per its requirements. Use these instructions to export the metadata of all items within a single collection or community.

Audience

- Repository Administrator

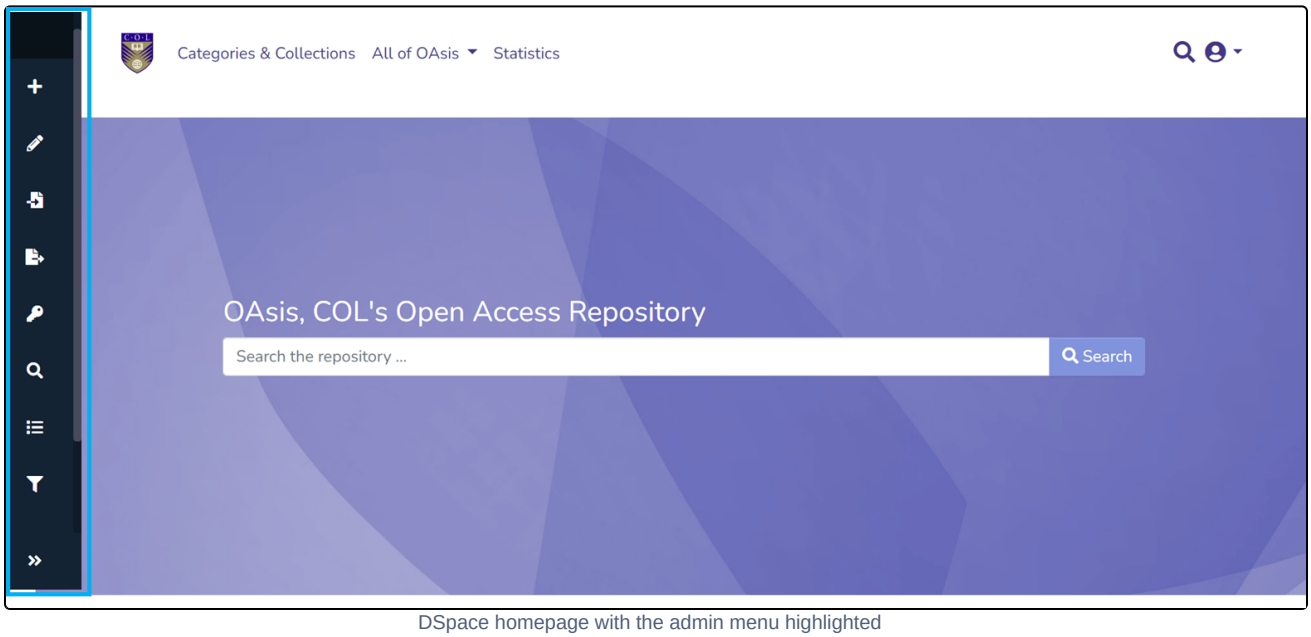
Exporting a collection

Step 1: Go to the DSpace home page and click on the “Log In” link, often in the top right corner, as illustrated below.



Login window

Step 2: Locate the admin menu, on the left-hand side of the screen in the illustration below.



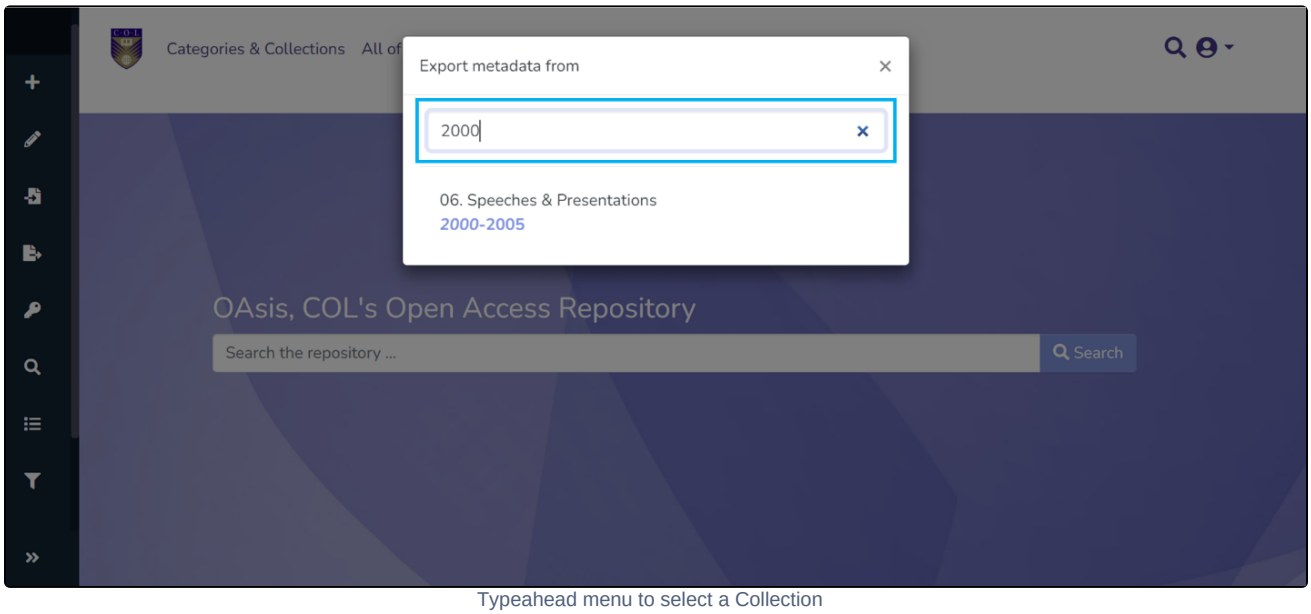
DSpace homepage with the admin menu highlighted

Step 3: Move your cursor over the "Export" menu and click on "Metadata."



Metadata is selected in the Export menu

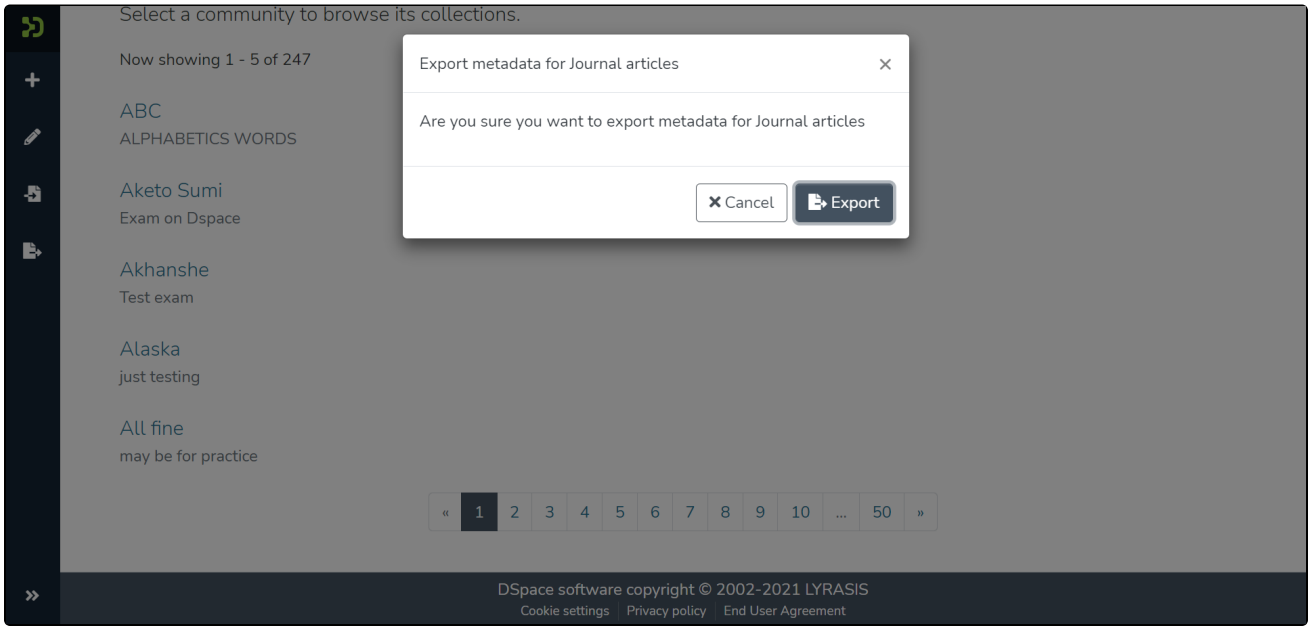
Step 4: Type the collection or community name in the textbox and click on the target collection or community from the list appearing in the popup.



Typeahead menu to select a Collection

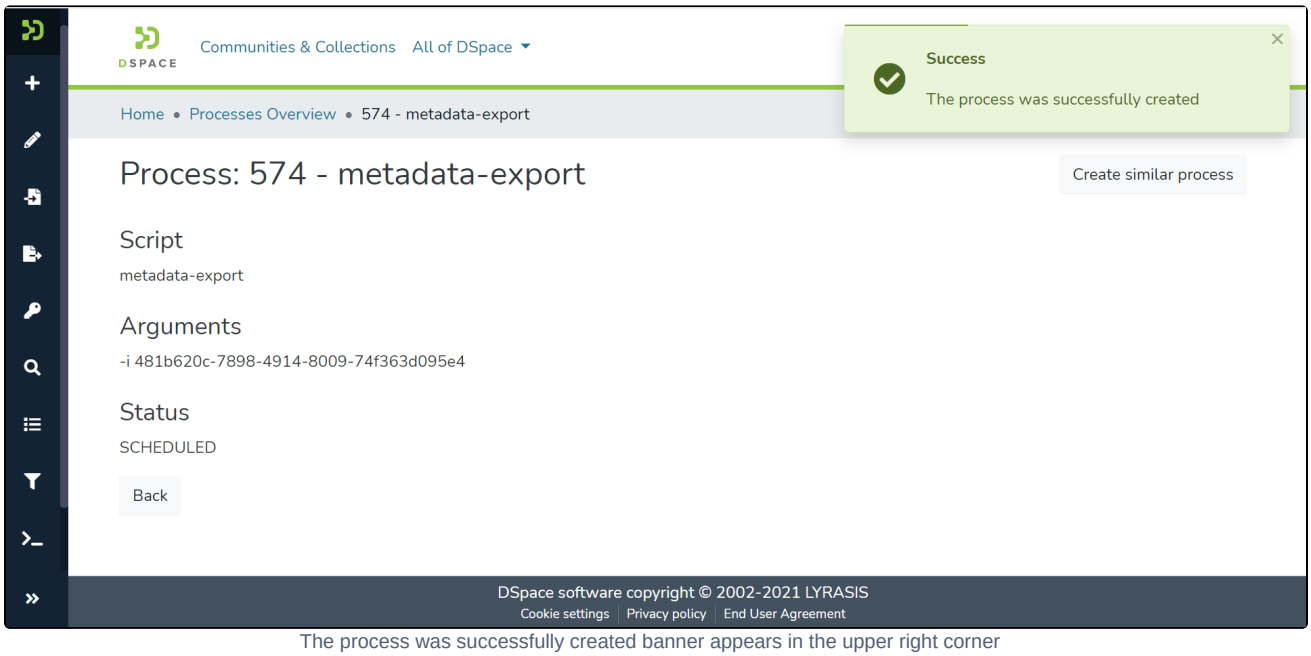
Note: If a user is on the collection or community page they wish to export metadata from when they access the admin menu in Step 2, that collection or community will appear at the top of the list without typing anything.

Step 5: Click the “Export” button in the popup to continue with the metadata export or click the “Cancel” button to cancel the process.



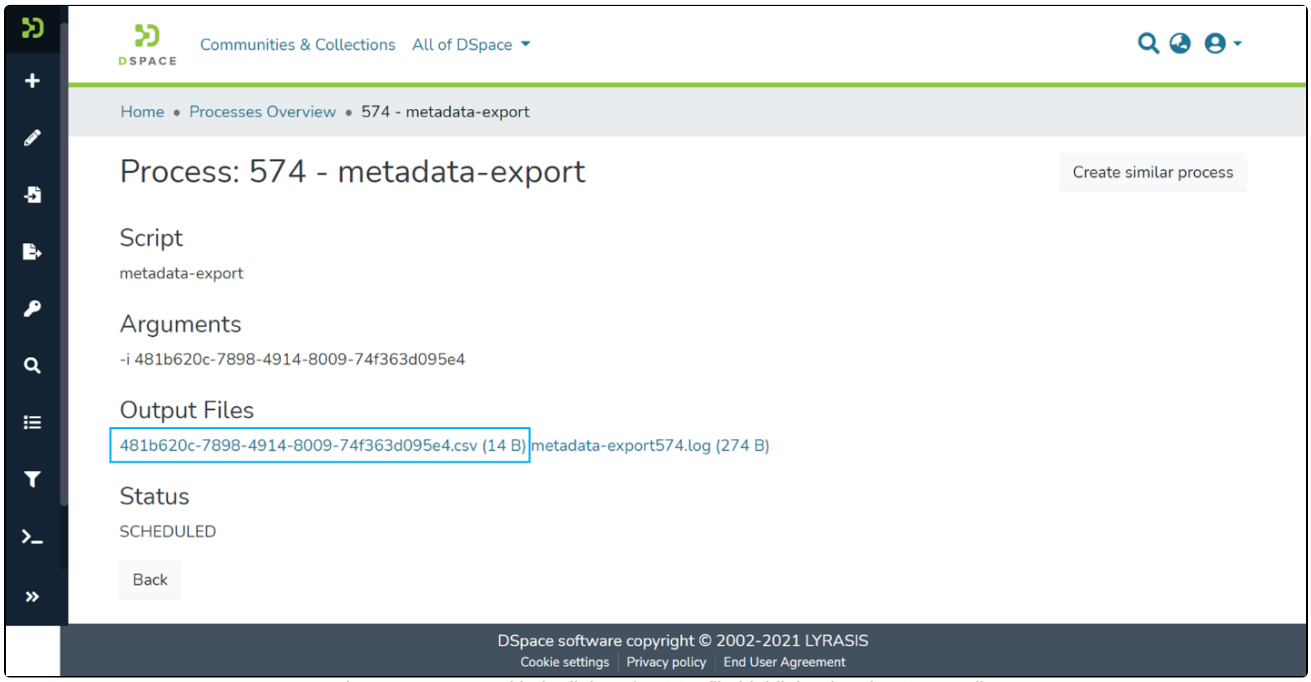
Export metadata popup with Export and Cancel buttons

Users will see the success prompt confirming the creation of the export process upon successful completion of the process, or else the application will show the failure promptly.



The process was successfully created banner appears in the upper right corner

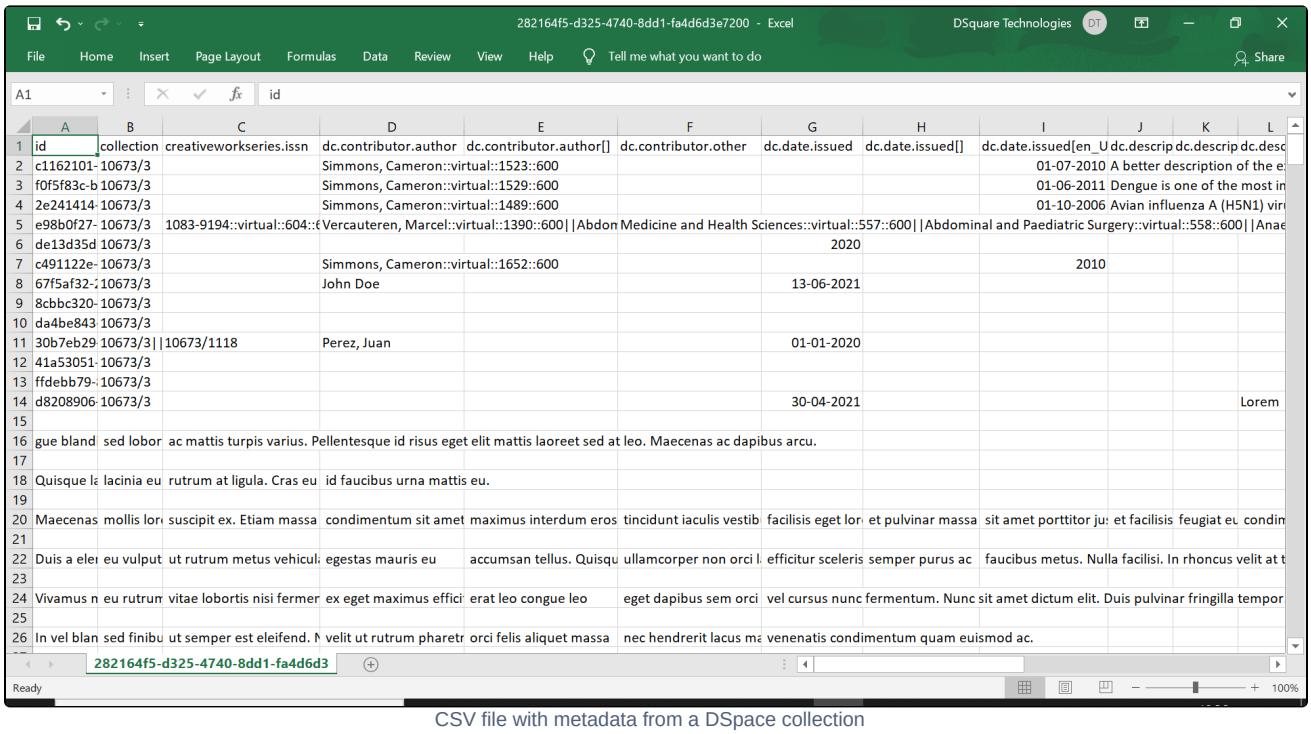
Step 6: Users will be redirected to the metadata export page with a csv download link, as highlighted in the screenshot below. If the download link does not appear right away, refresh the page.



Metadata export page with the link to the CSV file highlighted under Output Files

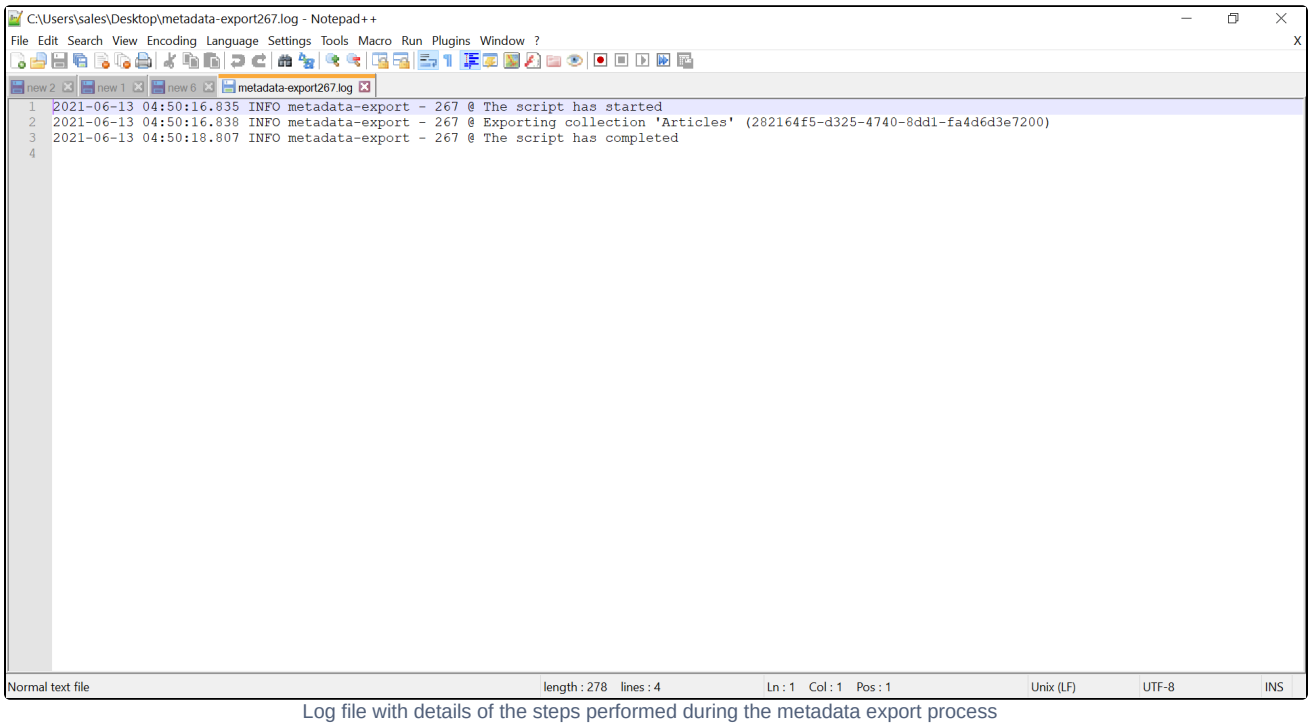
Users can perform the following actions on this page:

- Click on the CSV file link to download the metadata CSV. This file contains metadata of all items stored in the exported collection or community.



CSV file with metadata from a DSpace collection

- Click on the log file link to download. The Logfile contains details of steps performed during the export job.



Log file with details of the steps performed during the metadata export process

Content (Item) management

Documentation for repository managers.

- Add item
- Delete item
- Edit Item
    - Authorizations (Manage access to an item)
    - Collection Mapper
    - Edit Bitstream
    - Edit Metadata
    - Edit Relationship
    - Make an item discoverable
    - Make an item non-discoverable
    - Move an Item
    - Versioned Item
    - Withdraw an item
- Embargo an item
- Lease an item

Add item

- Target Audience
- Overview
- Submission Form Highlights
- Item Submission Process

Target Audience

Content Submitters

Overview

The item submission process lets authorized users deposit contents using metadata and bitstreams. It primarily consists of components.

1. Target collection where the item needs to be submitted
2. Submission form using which metadata and bitstreams related to the item are submitted
3. The submission form also helps in defining Access rights around an item

Submission Form Highlights

1. Bitstream upload section

This section allows users to upload bitstream (s) by browsing or drag & drop mechanism.

2. Target Collection

It is a location where the item will be submitted.

3. General Metadata section

Users can define general or primary metadata about an item in this section.

4. Additional Metadata section

This section allows users to add secondary or additional metadata.

5. Bitstreams Management

Bitstreams uploaded by users list in this section. Using various options, they can further define these bitstreams or remove them from the attachment list.

6. Deposit License

Users need to accept the license in this section to submit the item to the repository.

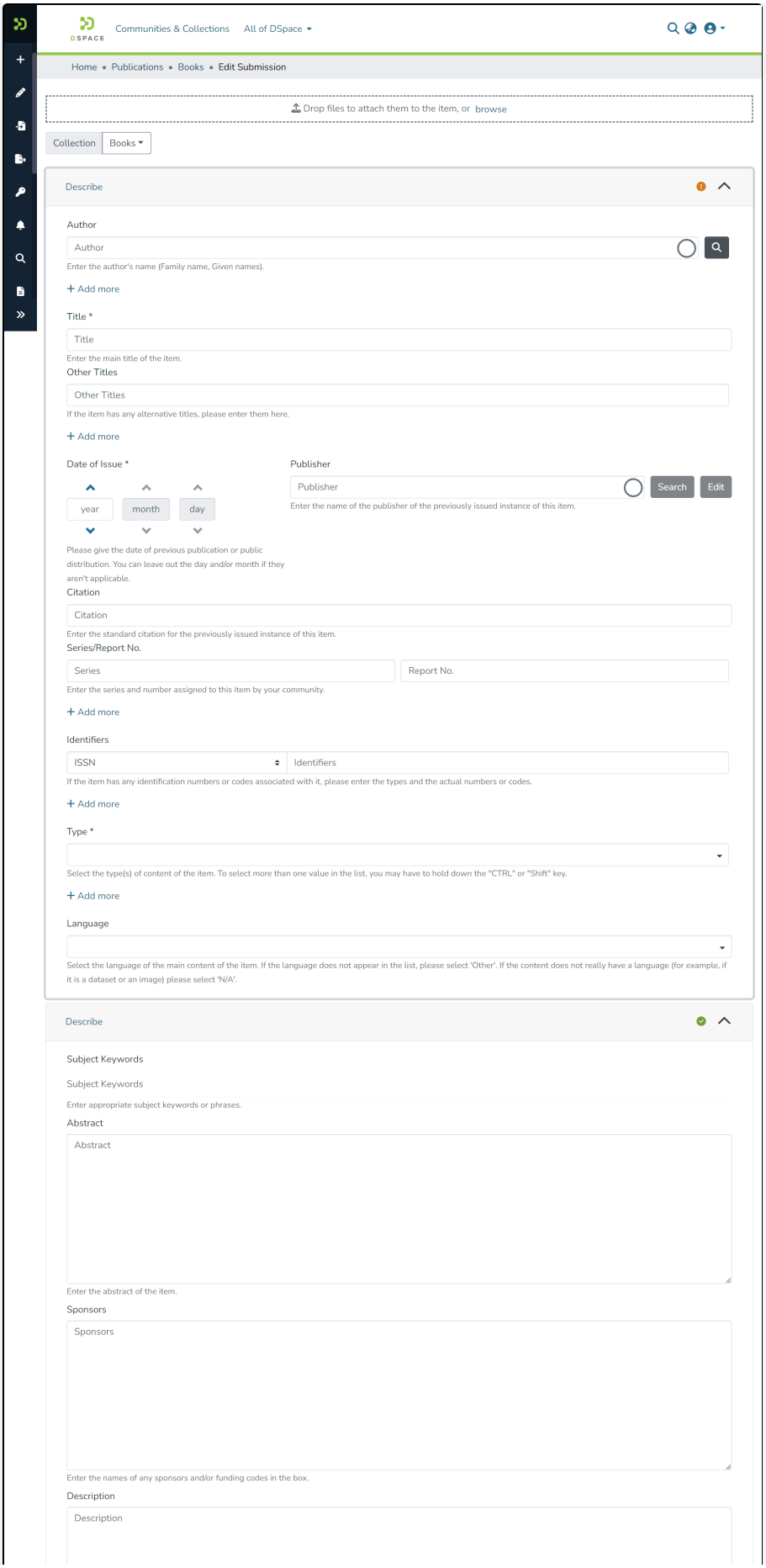
7. Manage Item Submission

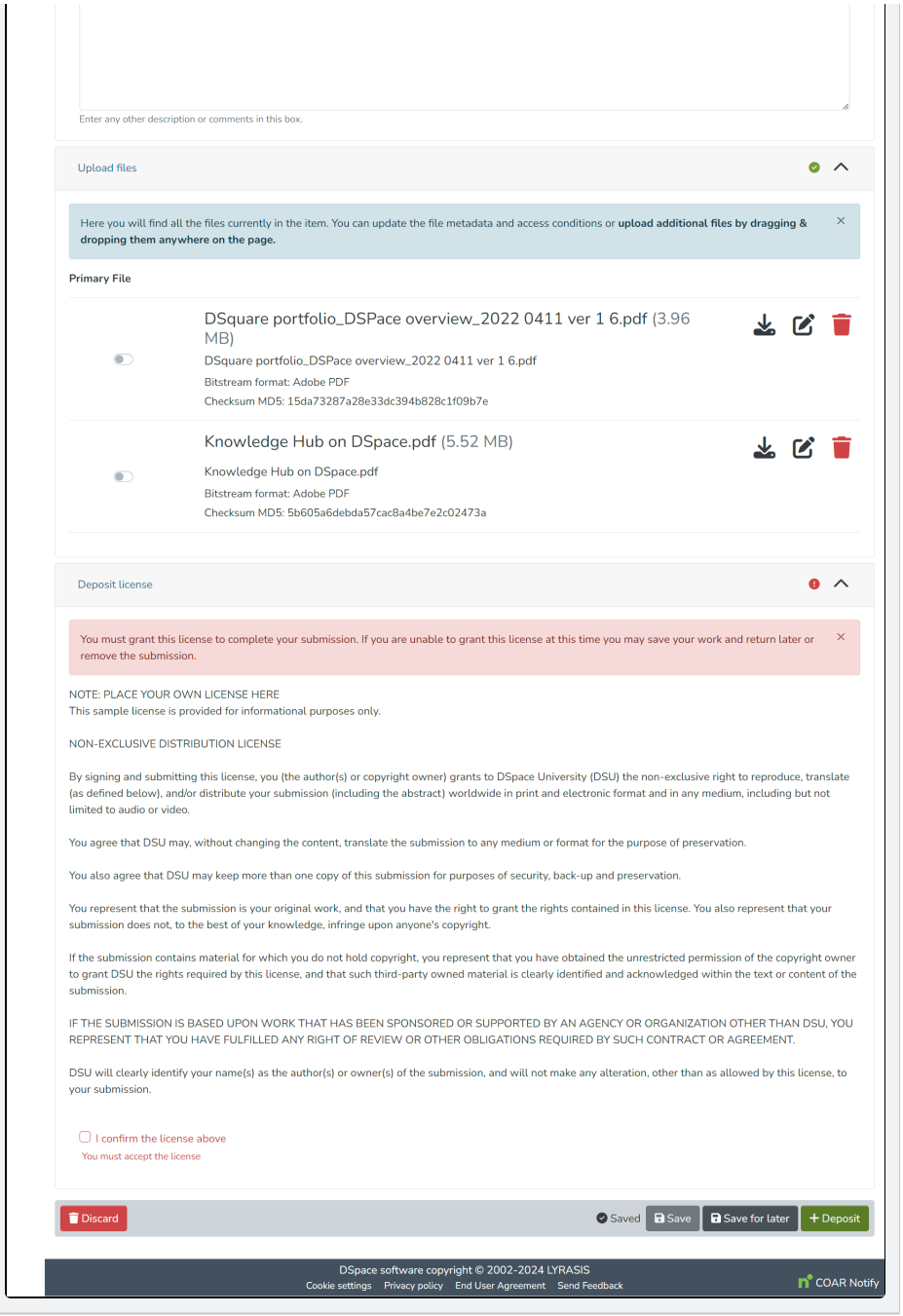
a. *Discard:* The user can discard the submission by clicking this button. Action will delete all information populated in the form permanently.

b. *Save:* This button helps save information in the submission form and helps resume information update should the process gets interrupted.

c. *Save for Later:* Save information in the MyDSpace section to update later.

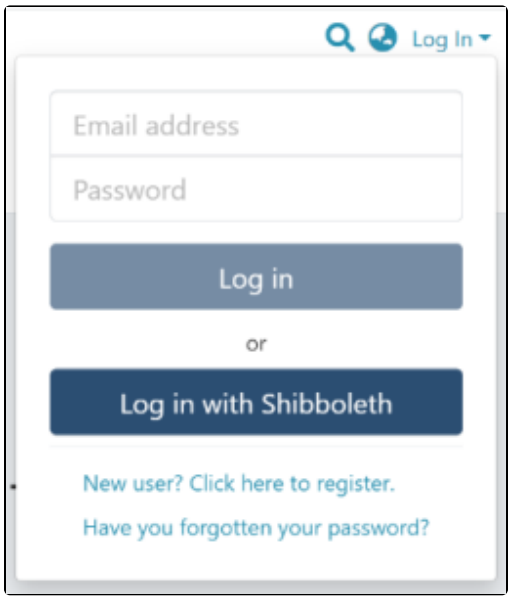
d. *Deposit:* Click this button to complete the submission. The item will go to the next step as per the workflow defined for the collection.



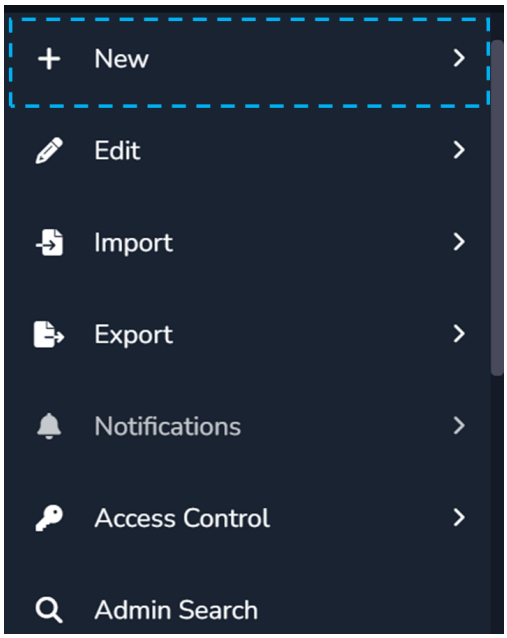


Item Submission Process

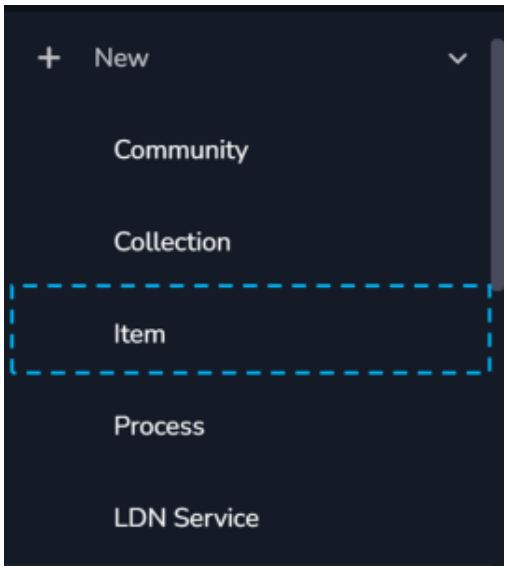
Step 1: Login using your credentials



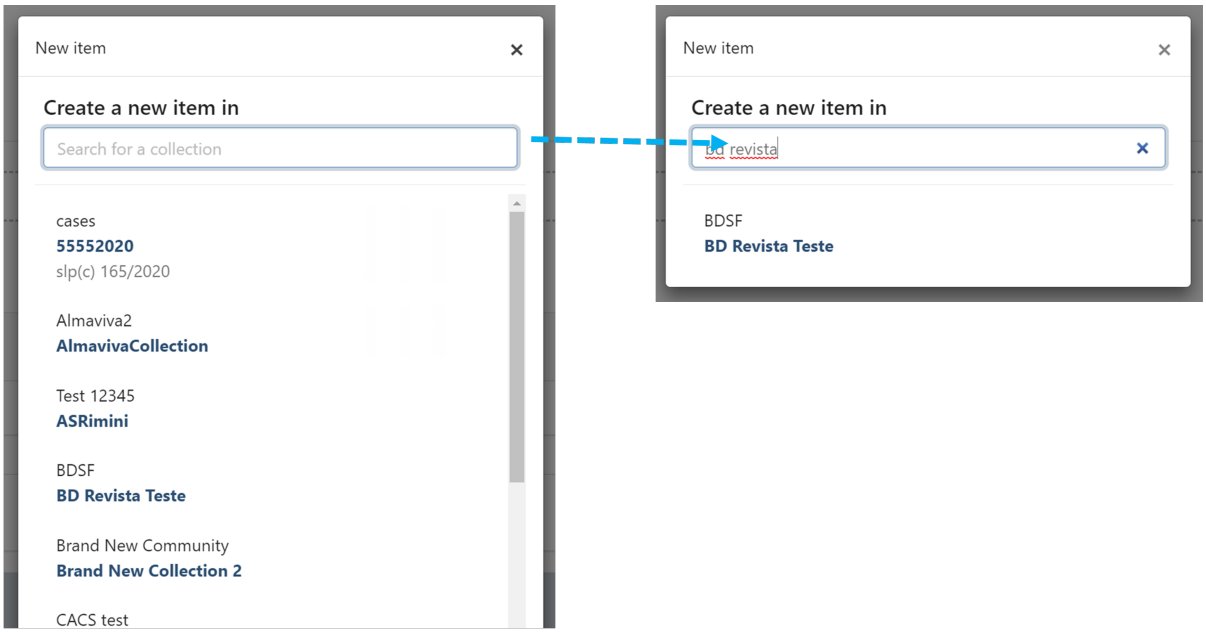
Step 2: Roll over the cursor on the "New" link.



Step 3: Click the “Item” link to continue with the process.

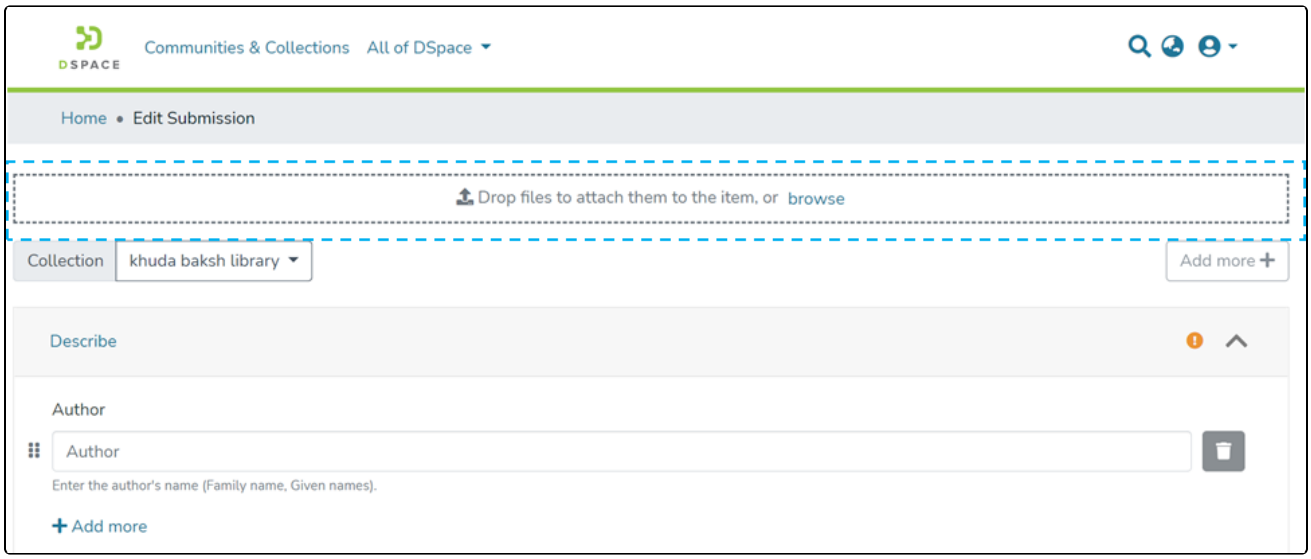


Step 4: A popup window with a collection list will appear. The user can select the target collection by typing its name or scrolling down the collection list. Then, click on the collection to initiate item submission.

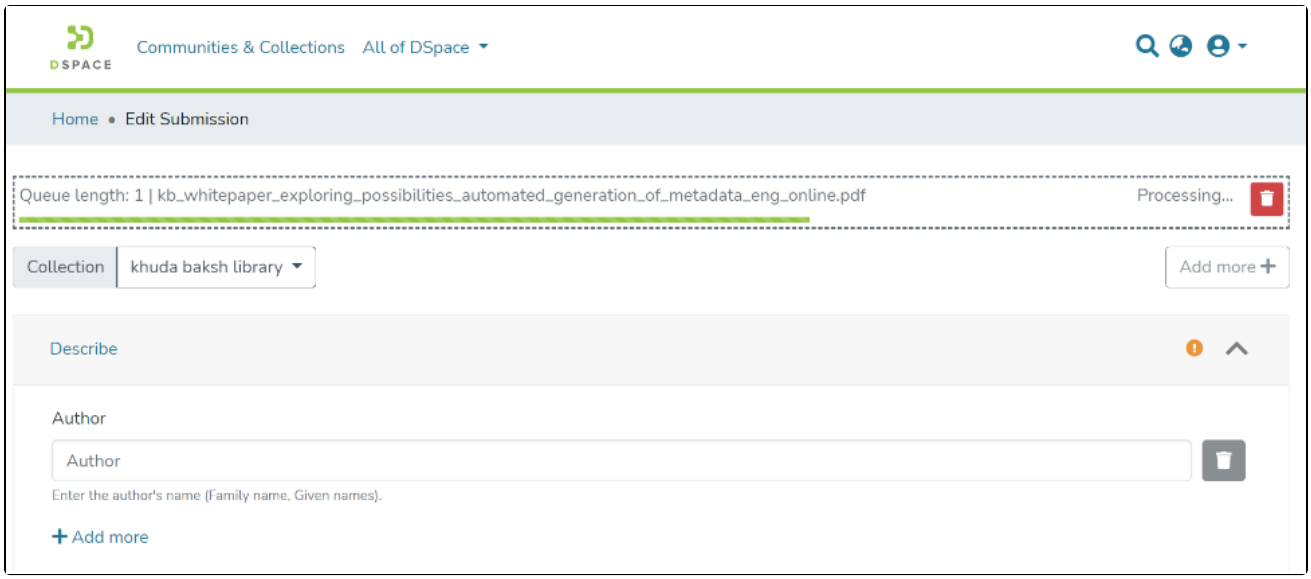


Step 5: Users will see the item submission form after selecting the target collection. The first step is to upload the attachment(s) in the item. In DSpace terminology, an attachment is known as a “bitstream”.

Click on the “browse” link to upload attachment(s). Users can upload multiple files by selecting them together or dragging in the space.



A bitstream upload progress bar will appear, as demonstrated in the illustration below. In addition, a prompt confirming success or failure will appear after a successful bitstream upload.



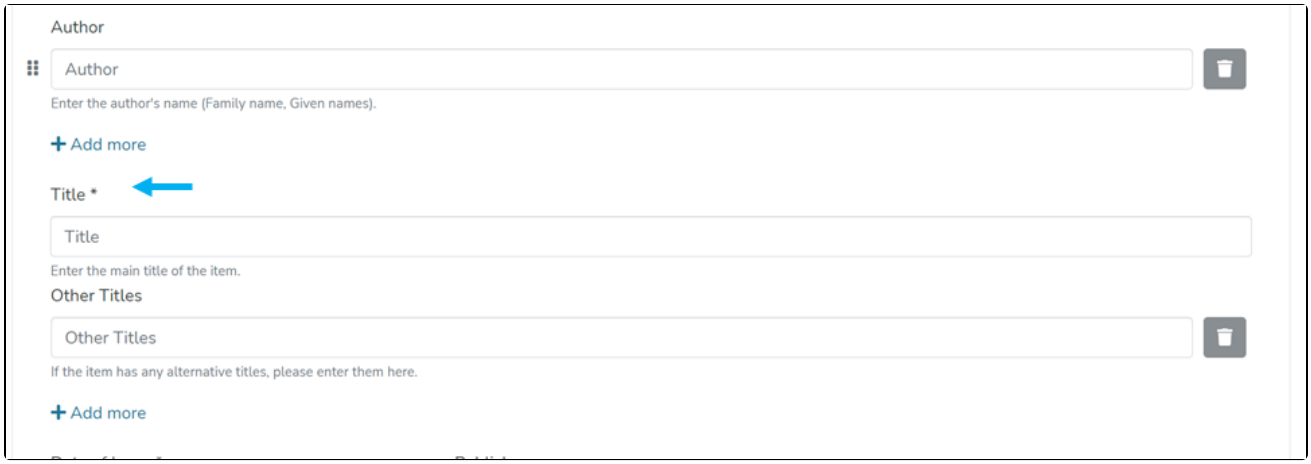
Bitstream upload in progress



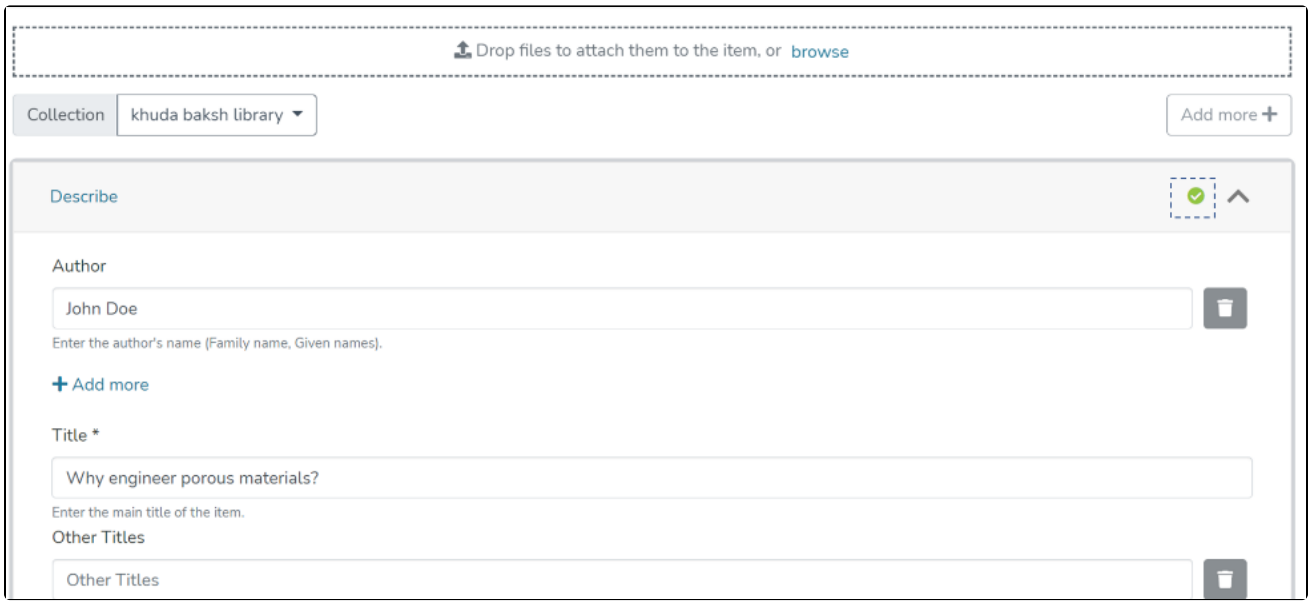
Bitstream Upload Successful

Step 6: After bitstream upload, the next step is to describe the item by adding metadata.

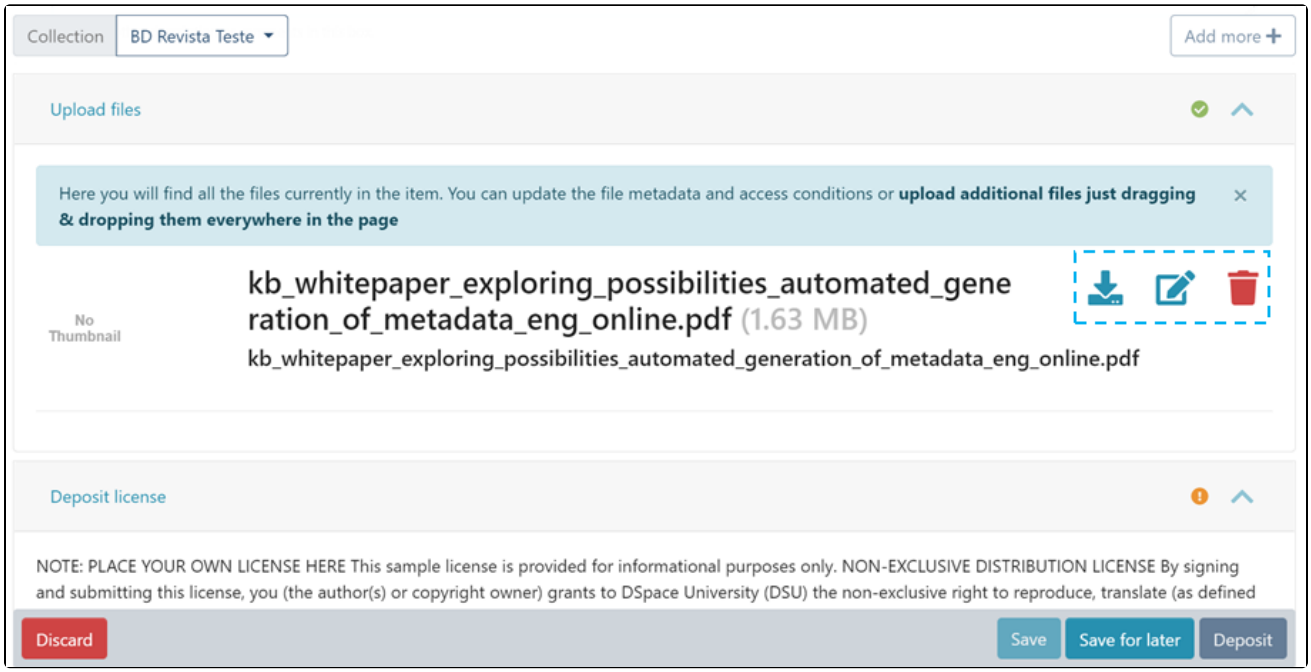
Metadata fields marked with "*" are mandatory, and users need to populate information in these fields to complete the submission mandatorily. A few examples in the standard submission form are Author, Title, and Date of Issue.



Users will notice an alert mark at the top right of the "Describe" tab turning from Amber to green once mandatory fields have values. Below is an illustration showing the state of the "Describe" section having values in all mandatory fields.



Step 7: The user can further update bitstreams by clicking on Buttons appearing next to the bitstream title.



Download: Click this button to download Bitstream on a local machine.

Edit: Update bitstream details and access rights using this button. More explanation is provided below.

Delete: Clicking this button will delete the bitstream from the submission form.

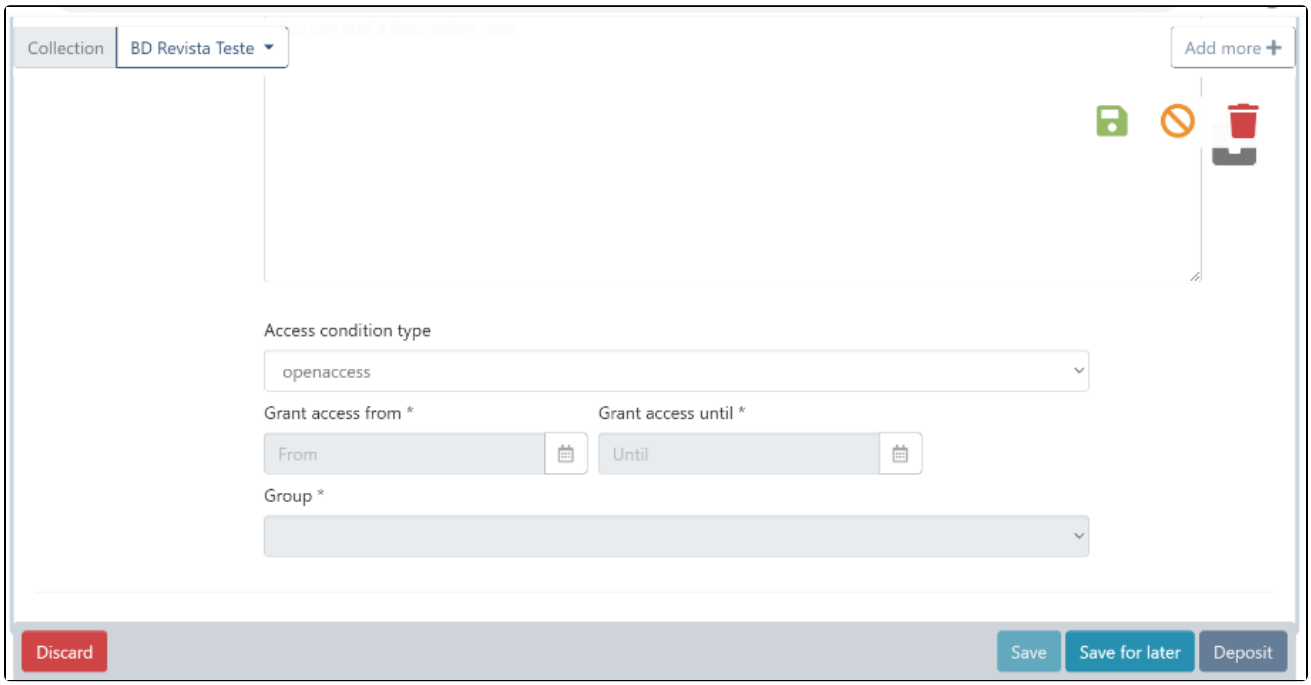
Step 8: By clicking the edit button next to the bitstream, users can update bitstream information, as explained below.

Update the bitstream title and add descriptions to describe the attachment further. Please refer to the illustration demonstrating both functions.

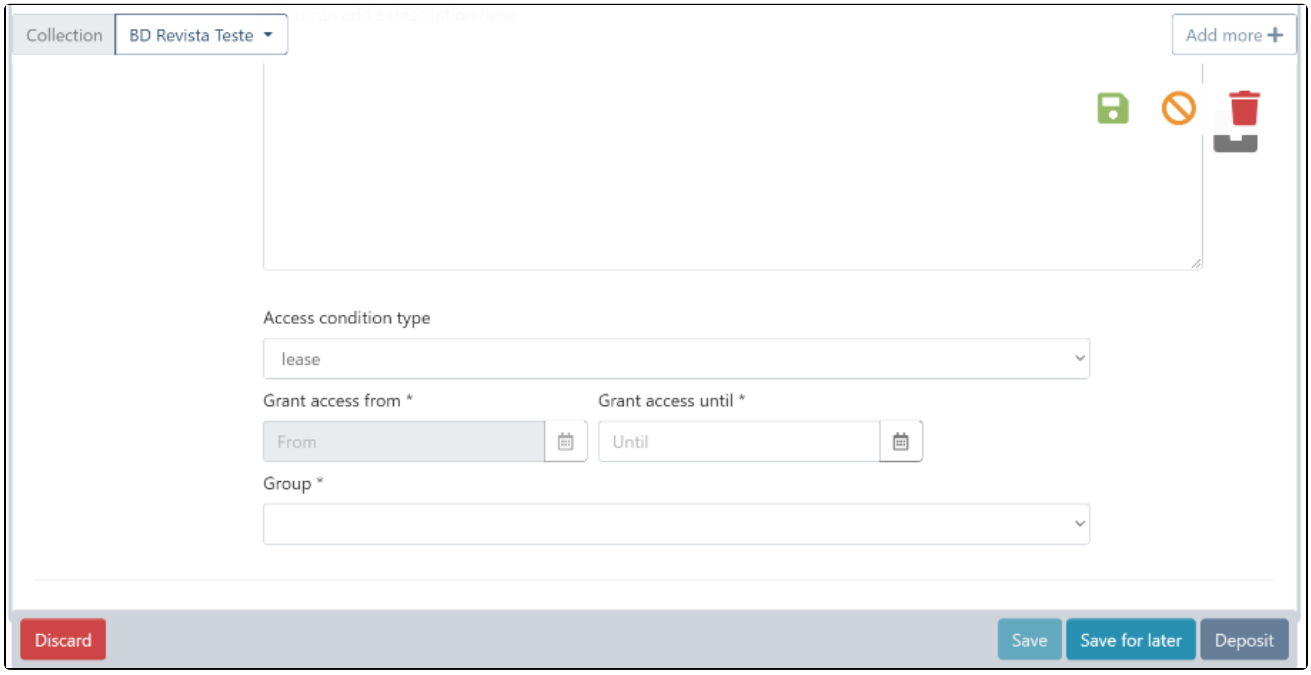


Users can define access conditions for the bitstream by selecting the appropriate option from the dropdown list. These options are:

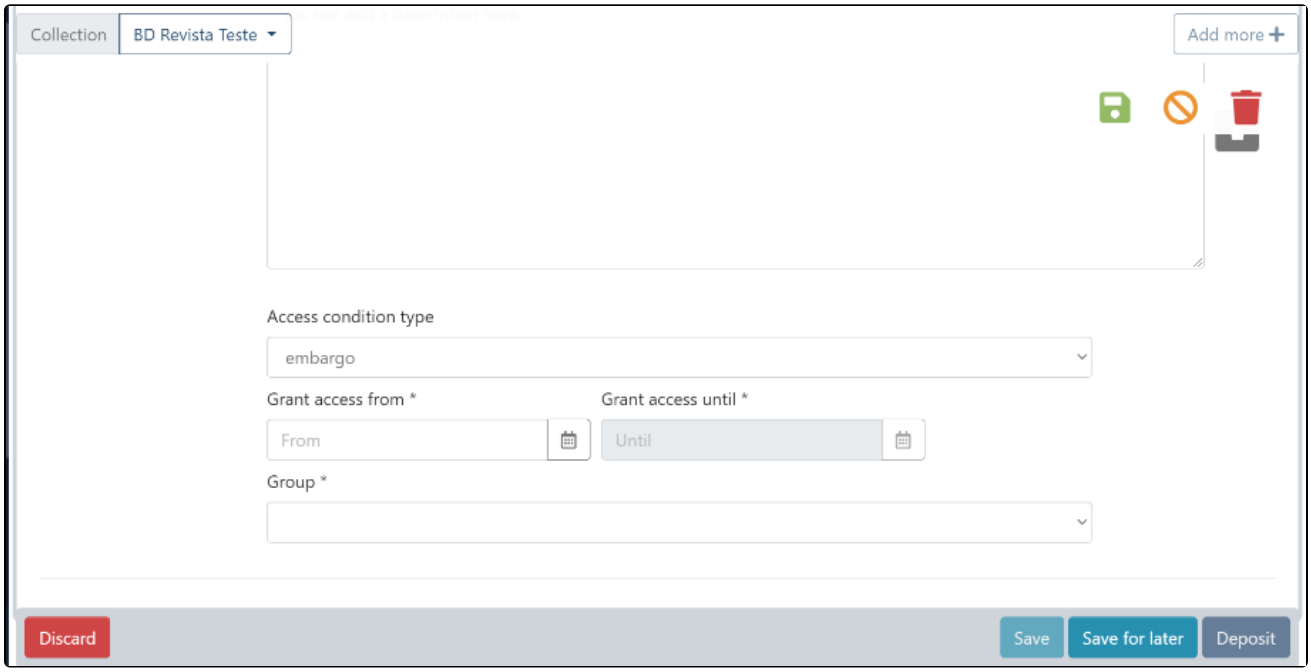
- Open Access: Select this option to make the bitstream available without restriction.



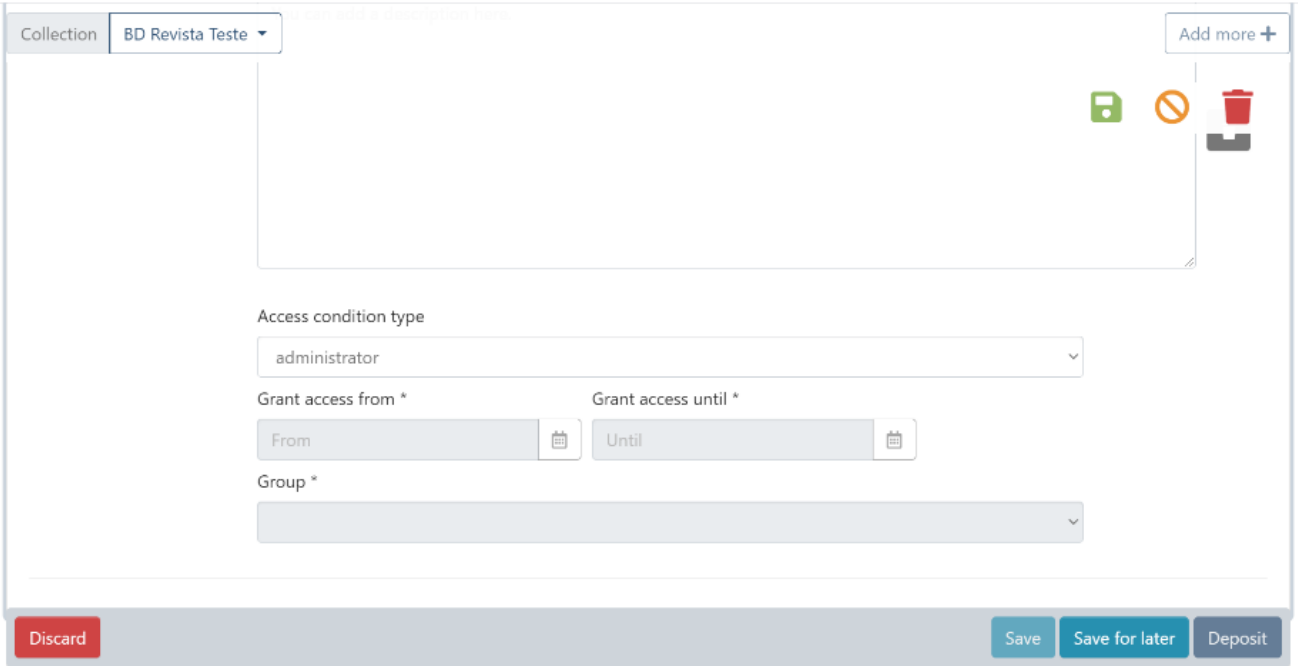
- Lease: This option is applicable when a user wants to keep Bitstream accessible until a specific date in the future. The bitstream will not be available as open-access content after the defined date under the “Grant access until” option.



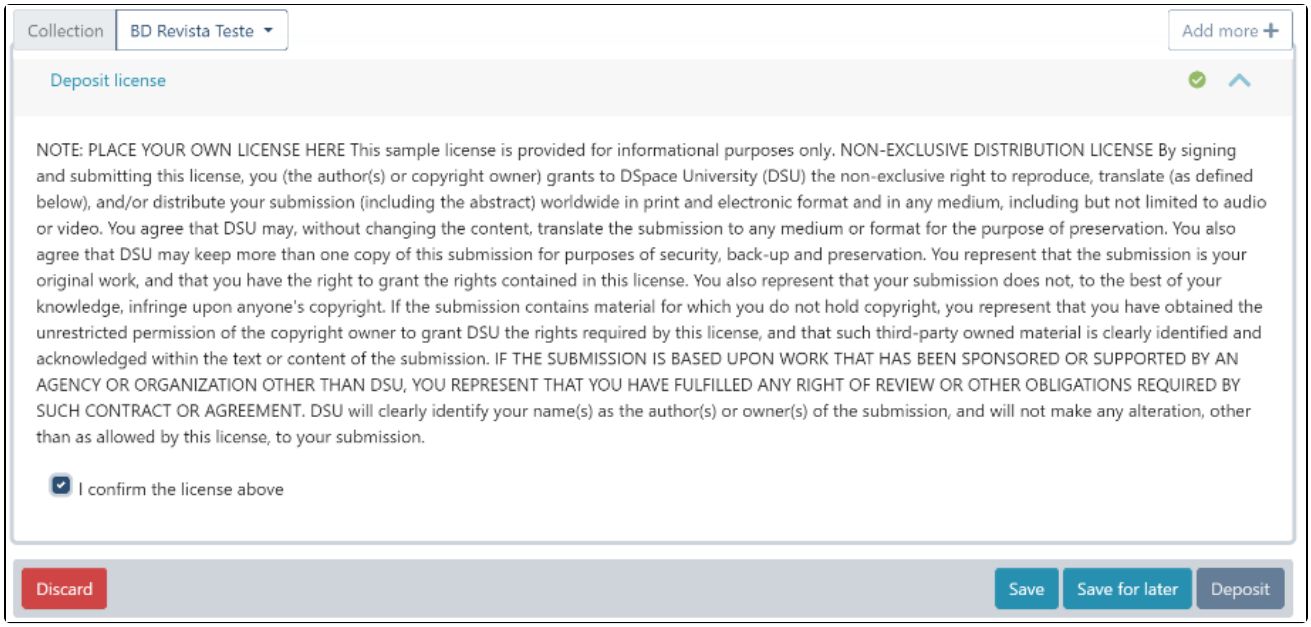
- Embargo: In contrast to a lease, an embargo allows the user to keep bitstream access restricted until a future date. This date is defined in the “Grant access from” field. The bitstream will be available as open-access content to users after this date.



- Administrator: Select this option if the bitstream’s access remains limited to administrators.



Step 9: Finally, users must click on the “I confirm the license above” checkbox to accept the deposit license and click the “Deposit” button to complete the item submission.



Delete item

- Scope and Use Case
- Target Audience
- Item Delete Process

Scope and Use Case

The Permanently Delete option is exercised when an authorized user wants to permanently delete an item (metadata + bitstreams) from the repository.

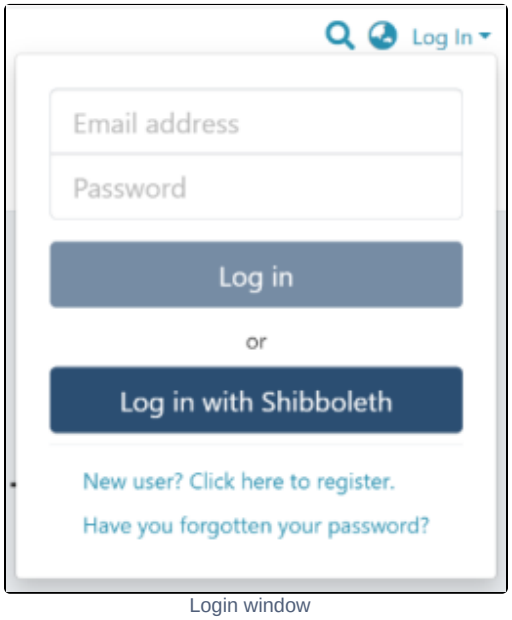
Apart from permanently deleting an item, options like “Withdraw item from the repository” and “Make item non-discoverable” can temporarily help disable the content access.

Target Audience

- Repository Administrators
- Community Administrators
- Collection Administrators

Item Delete Process

Step 1: Login using your credentials



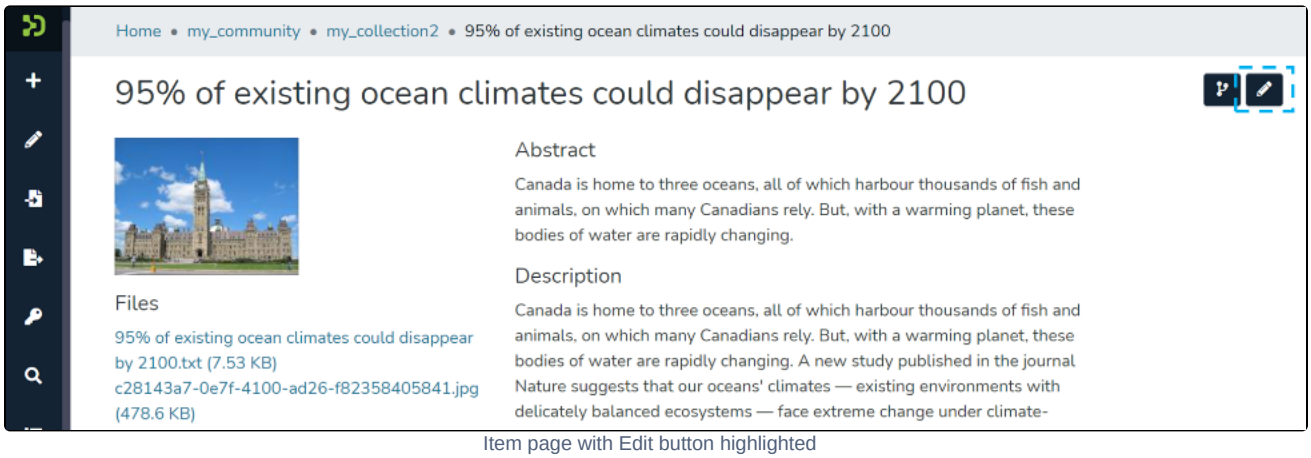
Login window

Step 2: Go to the item you want to move.

Users can reach an item through multiple methods, including those listed below:

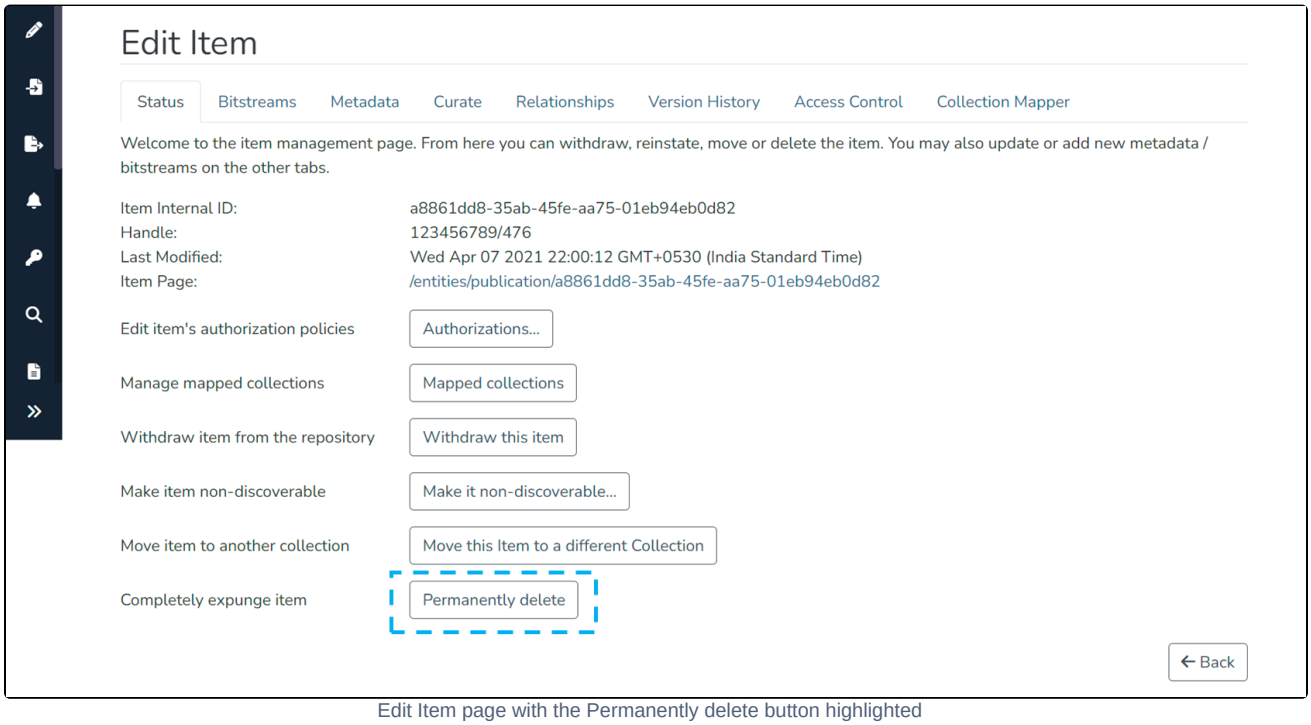
1. Search for an item
2. Browse communities and collections
3. Find an item in the Administration section at Edit > Item
4. Use the existing item URL

Step 3: Click on the “Edit” button as highlighted on the screen below. This button will appear to any user who has edit rights on the target item.



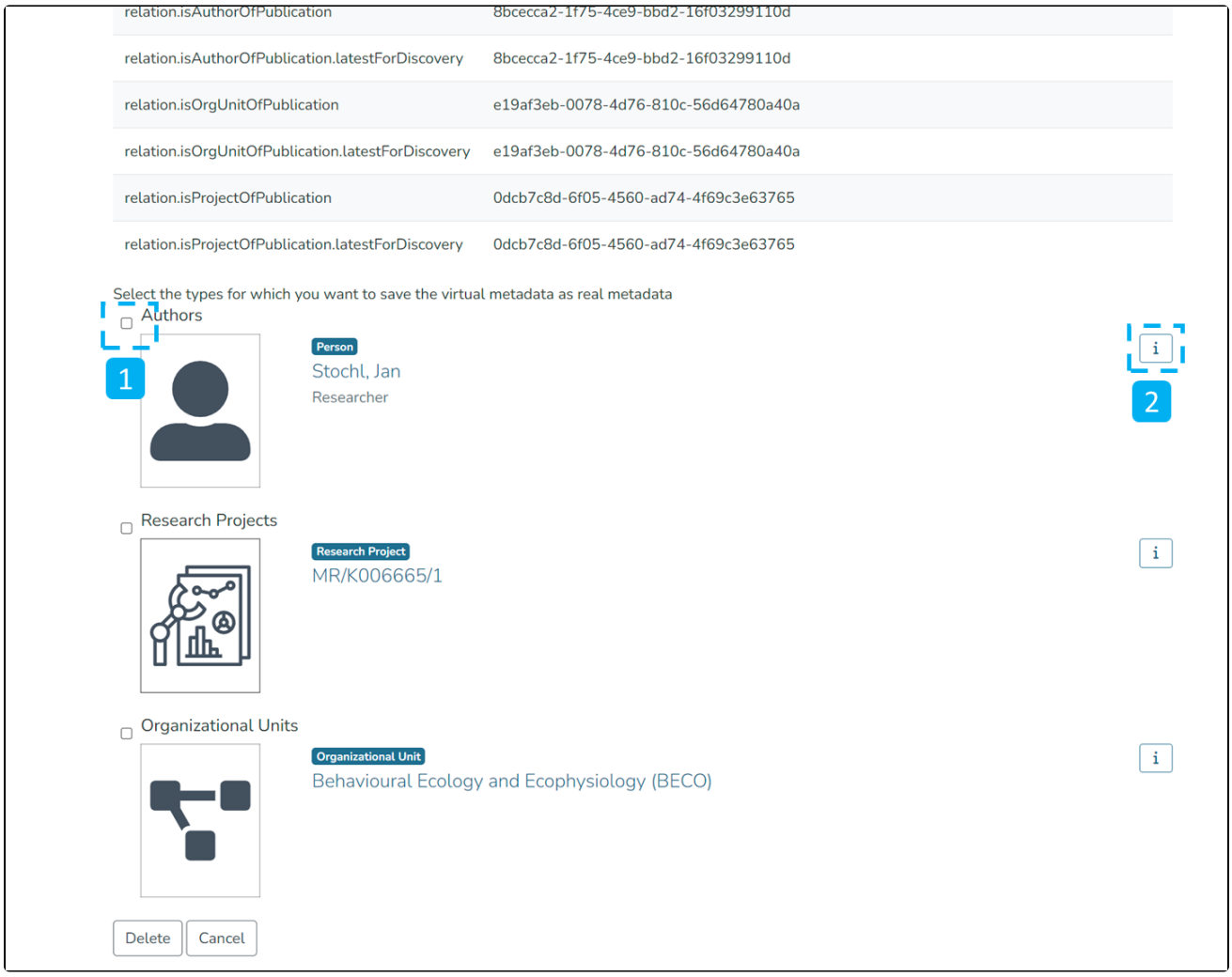
Item page with Edit button highlighted

Step 4: Click the “Permanently delete” button to delete the item.



Edit Item page with the Permanently delete button highlighted

Step 5: Select entities for which virtual metadata needs to be retained, as highlighted below. To see information about the entity, click the "*i*" icon appearing next to the target entity, as highlighted in the below screenshot.



Step 6: Click the Delete button on the confirmation screen to permanently delete the item. Click the Cancel button to cancel the deletion.



Screenshot showing the Delete and Cancel buttons

Edit Item

- Scope and Use Case
- Audience
- Access the Edit Item Page
- Edit Item Overview
    - Status tab
    - Bitstreams tab
    - Metadata tab
    - Curate tab
    - Relationships tab
    - Version History tab
    - Access Control tab
    - Collection Mapper tab

Scope and Use Case

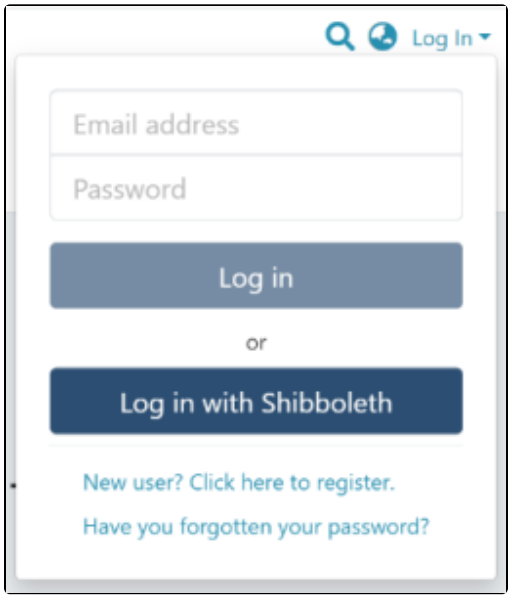
Items within DSpace (including bitstreams and metadata) can be edited after submission by authorized users. This page describes how to access the Edit Item page and provides an overview of the Edit Item page. It shows screenshots of each tab in the Edit Item page and lists the functions that are available on each tab. For details about using specific functions, see the specific pages within this wiki (linked below).

Audience

- Repository Administrators
- Community Administrators
- Collection Administrators

Access the Edit Item Page

Step 1: Login using your credentials

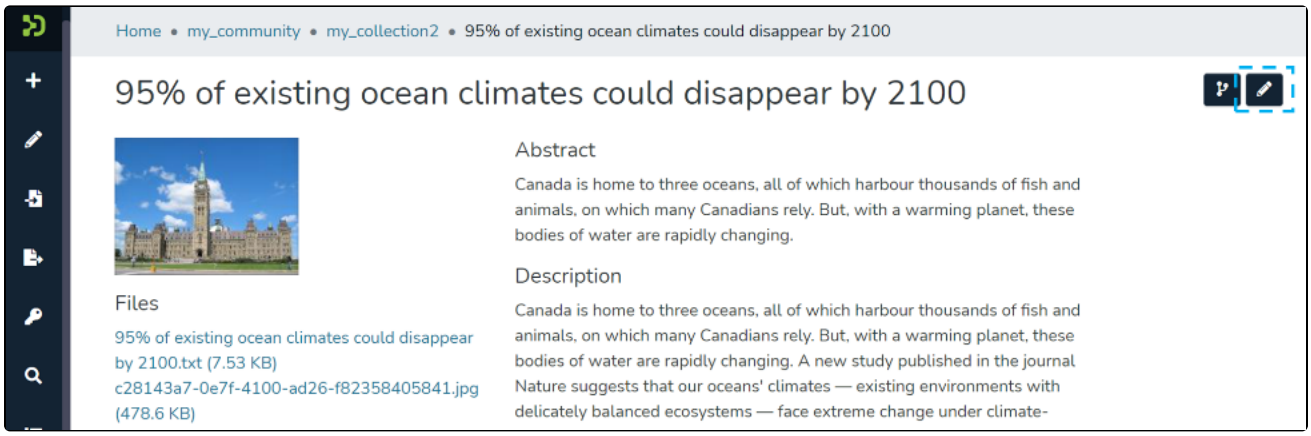


Step 2: Go to the item you want to move.

Users can reach an item through multiple methods, including those listed below:

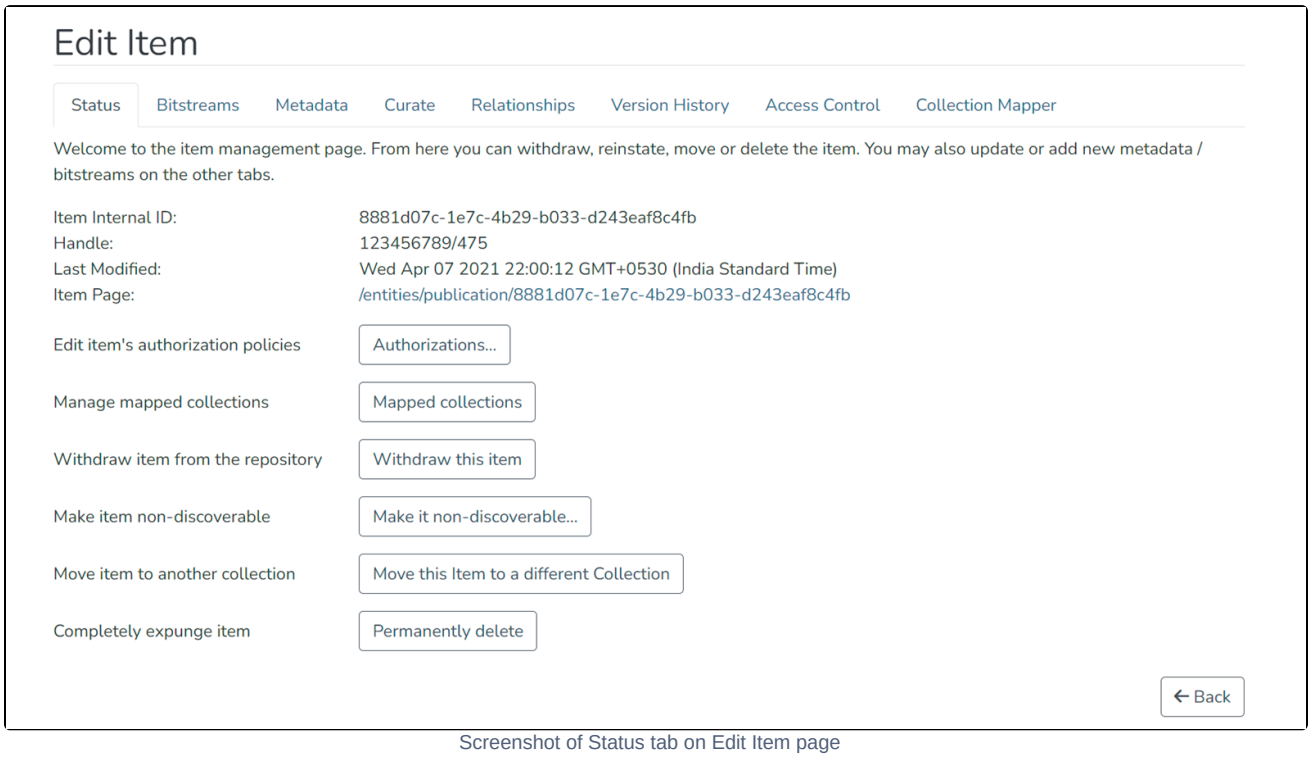
1. Search for an item
2. Browse communities and collections
3. Find an item in the Administration section at Edit > Item
4. Use the existing item URL

Step 3: Click on the “Edit” button as highlighted on the screen below. This button will appear to any user who has edit rights on the target item.



Edit Item Overview

Status tab

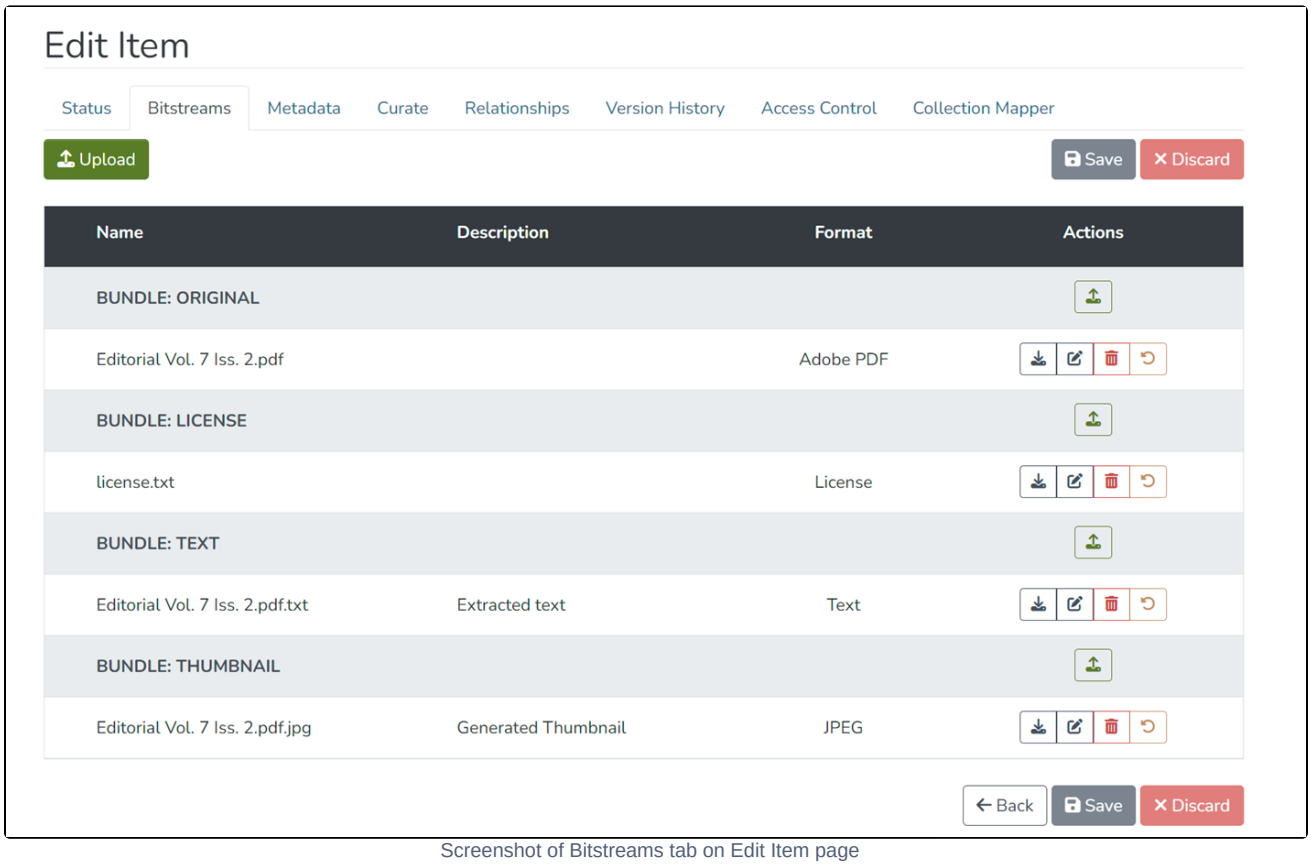


Screenshot of Status tab on Edit Item page

- Edit item's authorization policies - Authorizations (Manage access to an item)
- Manage mapped collections - Collection Mapper
- Withdraw item from the repository - Withdraw an item
- Make item non-discoverable - Make an item non-discoverable (or make it discoverable after it's been made non-discoverable - Make an item discoverable)
- Move item to another collection - Move an Item
- Completely expunge item (permanently delete) - Delete item

Bitstreams tab

See Edit Bitstream page

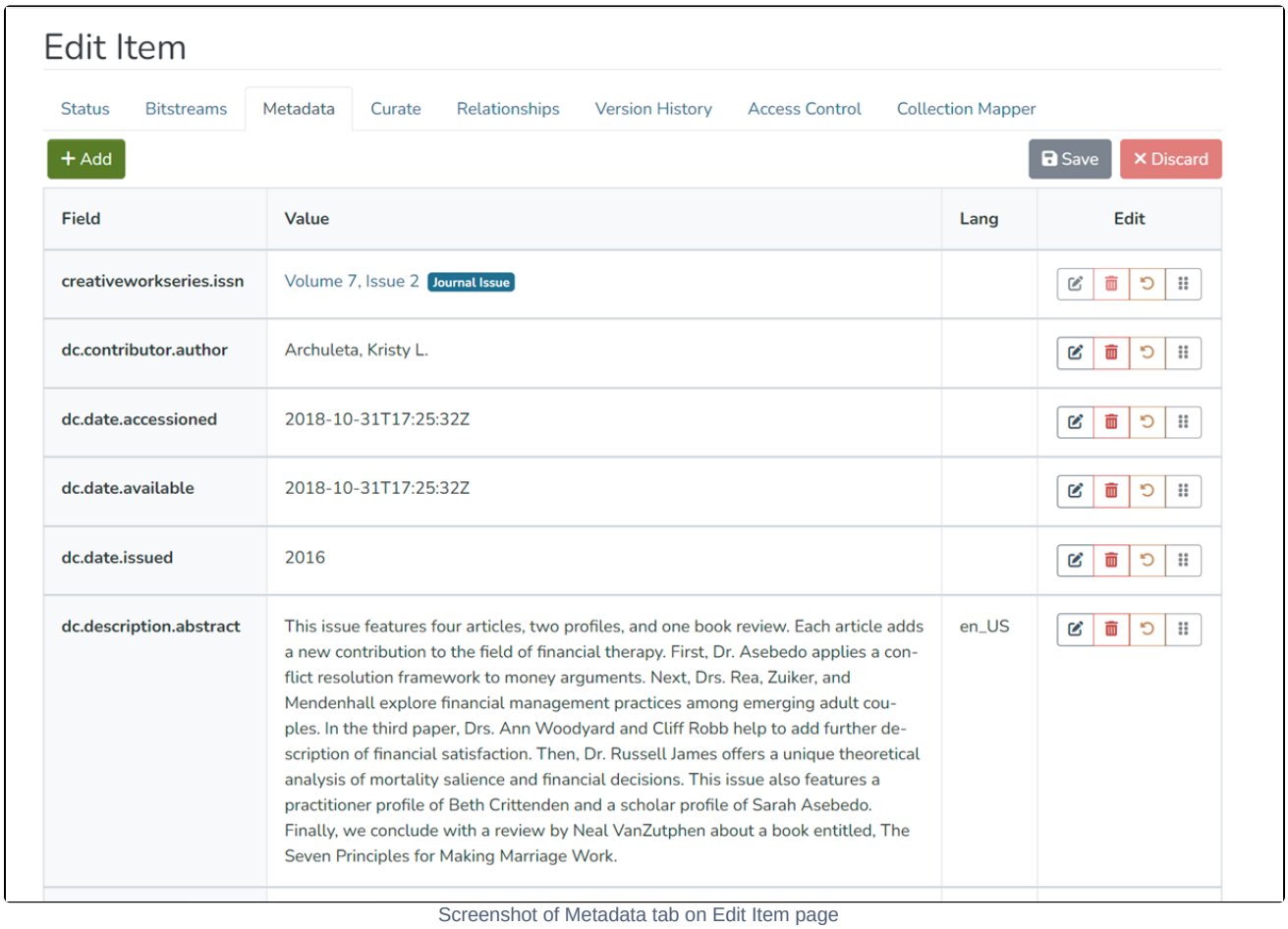


Screenshot of Bitstreams tab on Edit Item page

- Add or remove licenses or other bitstreams (i.e., files) attached to the item
- Edit metadata associated with the bitstreams, including filename, file format, and file description
- Reorder bitstreams attached to the item

Metadata tab

See Edit Metadata page

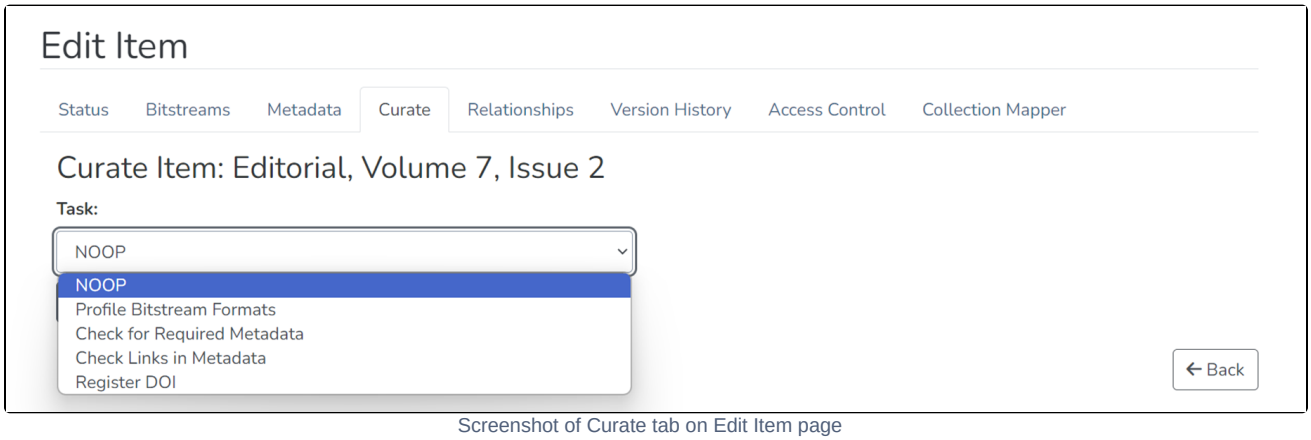


Screenshot of Metadata tab on Edit Item page

- Add or delete metadata fields (i.e., elements added to describe the item)
- Edit values in existing fields

Curate tab

See Curation System page (In the admin UI section)

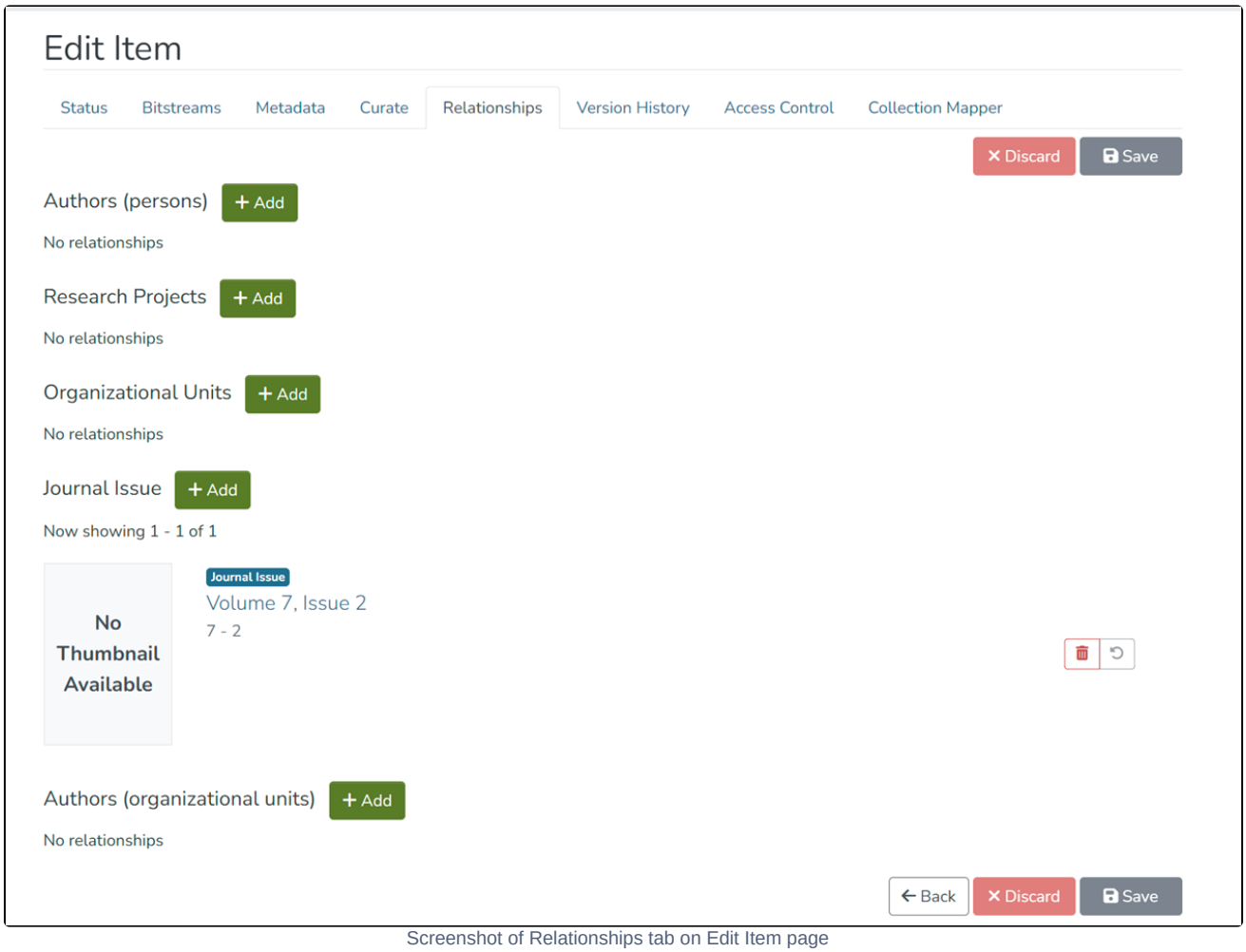


Screenshot of Curate tab on Edit Item page

- Perform various curation processes on the item

Relationships tab

See Edit Relationship page

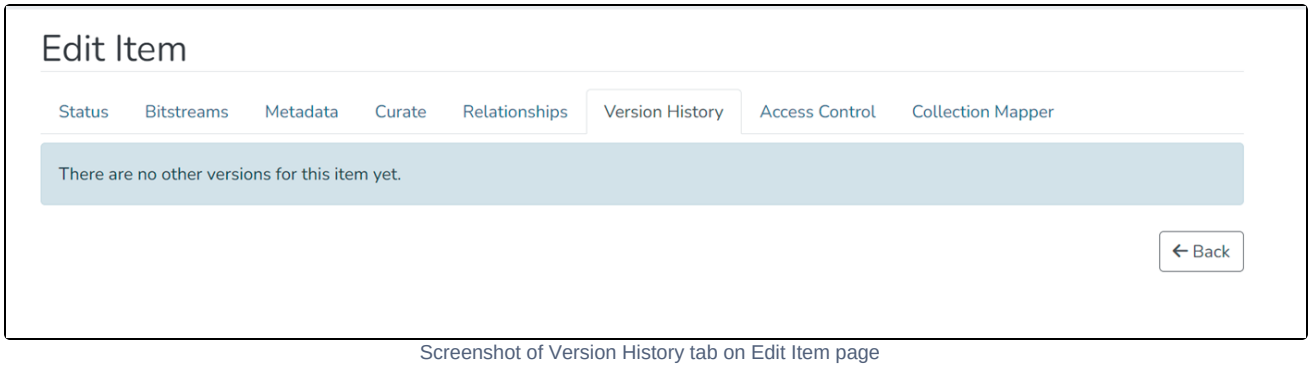


Screenshot of Relationships tab on Edit Item page

- Add or delete relationships with other items
- Edit existing relationships with other items

Version History tab

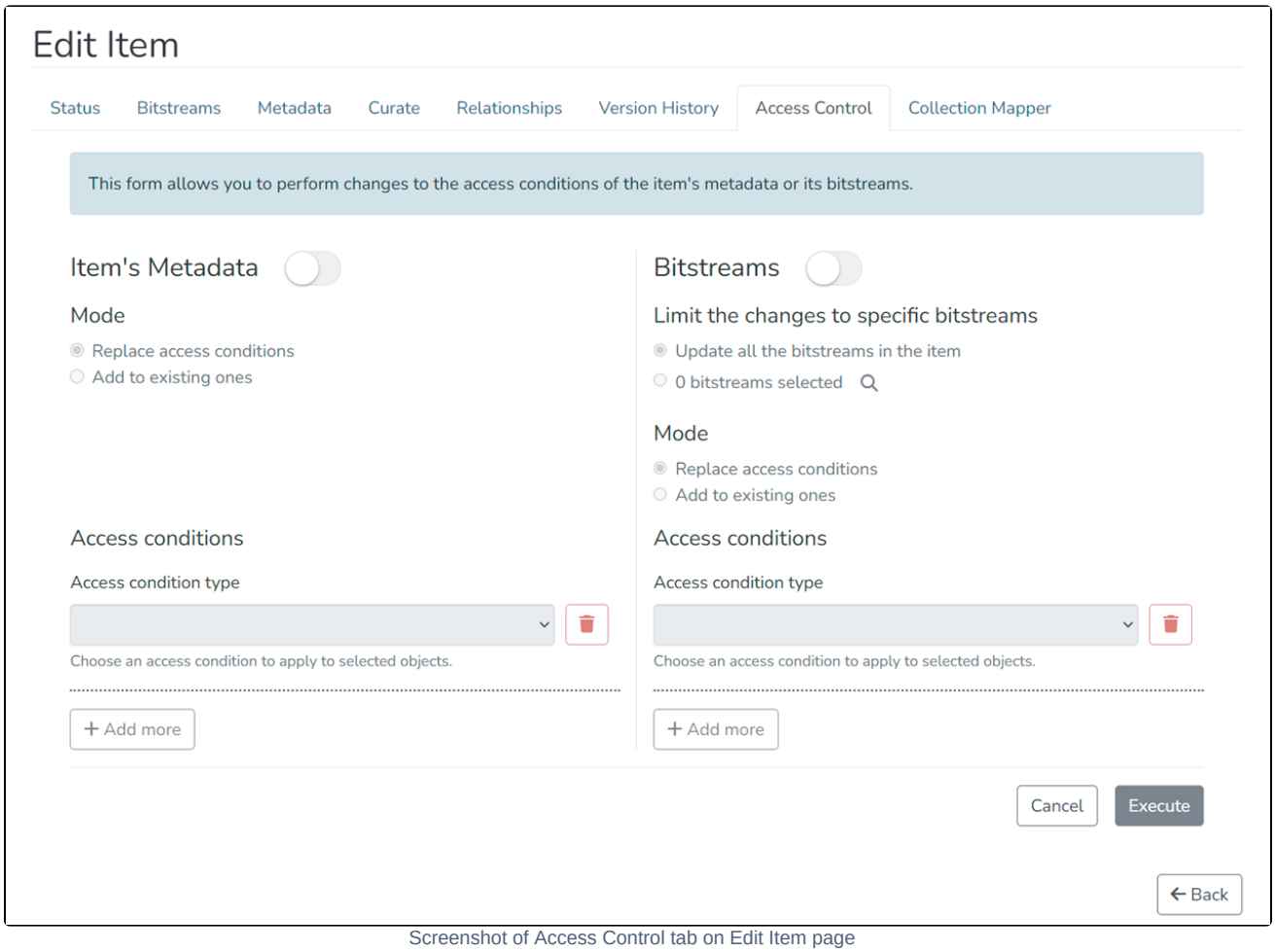
See Versioned Item page



Screenshot of Version History tab on Edit Item page

- Create and manage item versions

Access Control tab

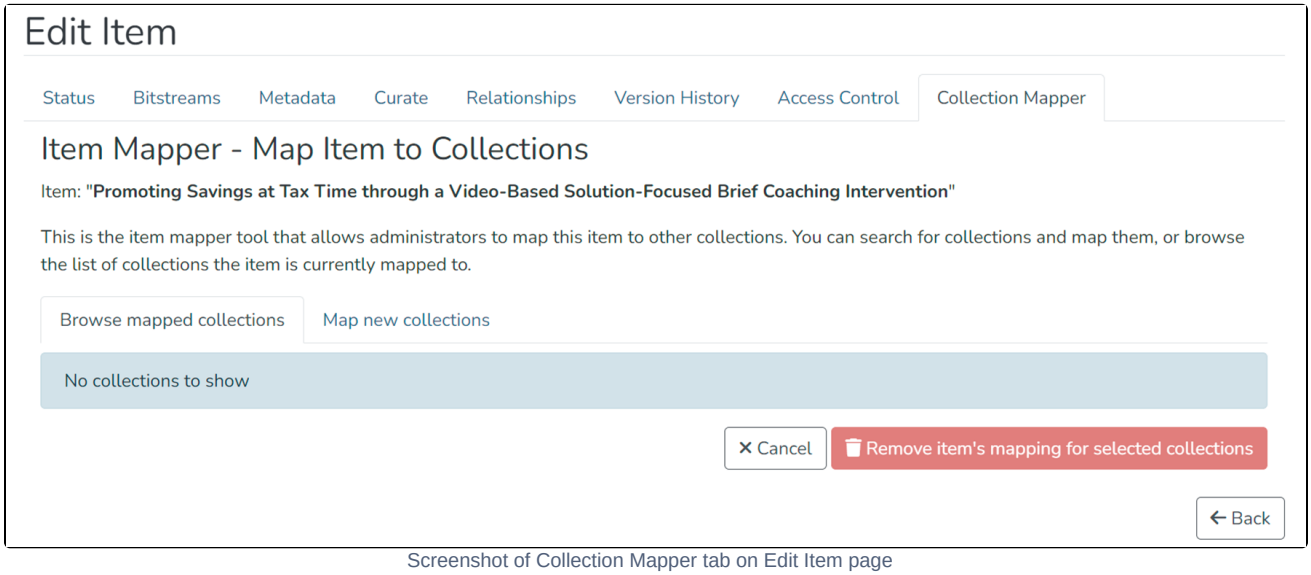


Screenshot of Access Control tab on Edit Item page

- Manage separate access policies for item's metadata and bitstreams

Collection Mapper tab

See Collection Mapper page



Screenshot of Collection Mapper tab on Edit Item page

- Map item to other collections and manage collections mapped with the item

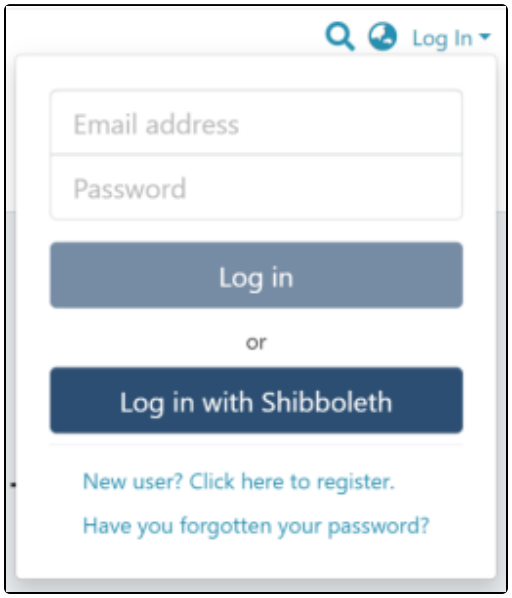
Edited and approved by DCAT Documentation Working Group, December 2024.

Authorizations (Manage access to an item)

- Overview
- Add Authorization Policy
- Manage Policy
- Delete Policy

Overview

Step 1: Login using your credentials



Step 2: Go to the Item you want to edit

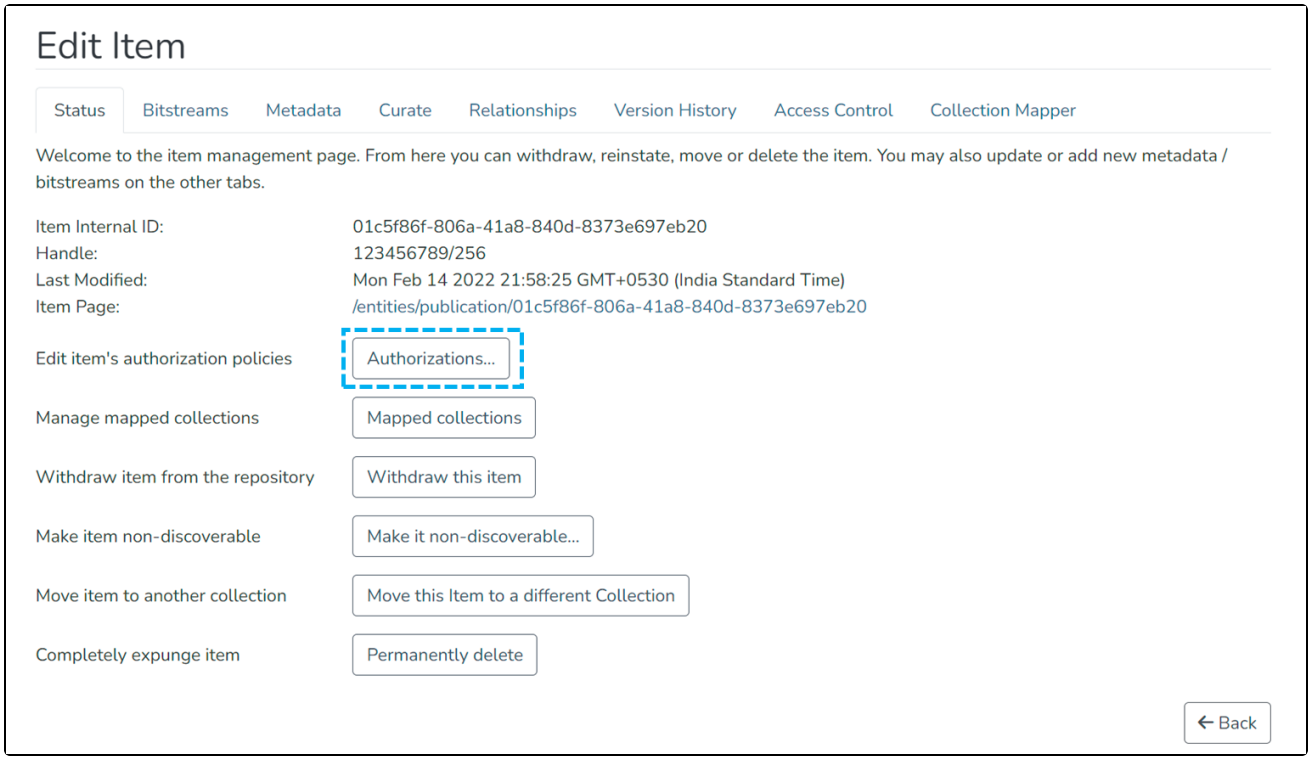
Users can reach an item through multiple methods, as listed below:

1. Search an item
2. Browse communities and collections
3. Finding an item in the Administration section at Edit > Item

Click the "Edit" button on the item title's right-hand side.



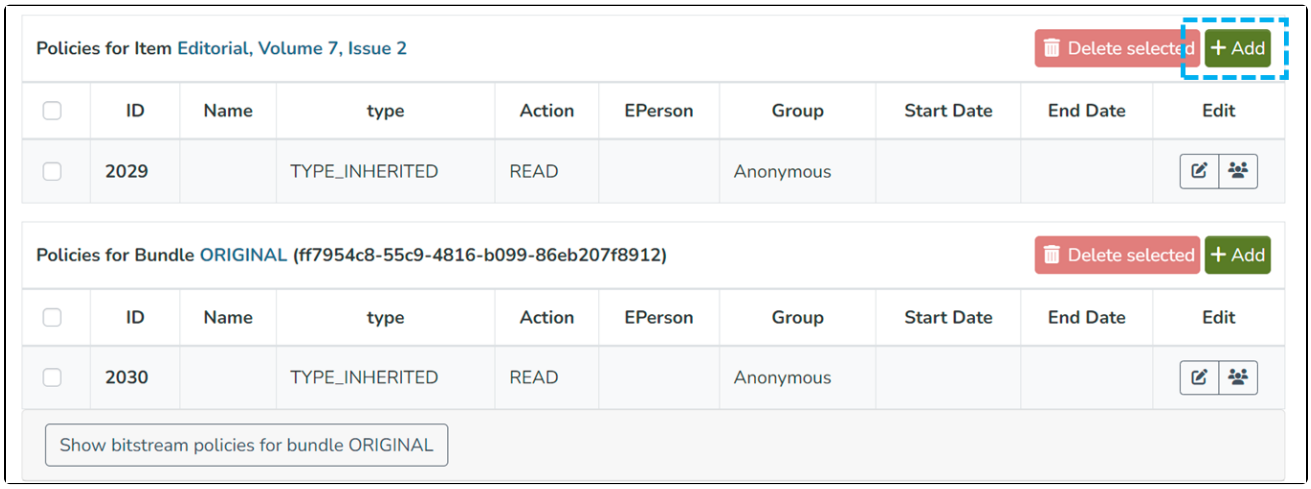
Step 3: Click the "Authorizations" button under the "Status" tab to manage the Item's authorization policies.



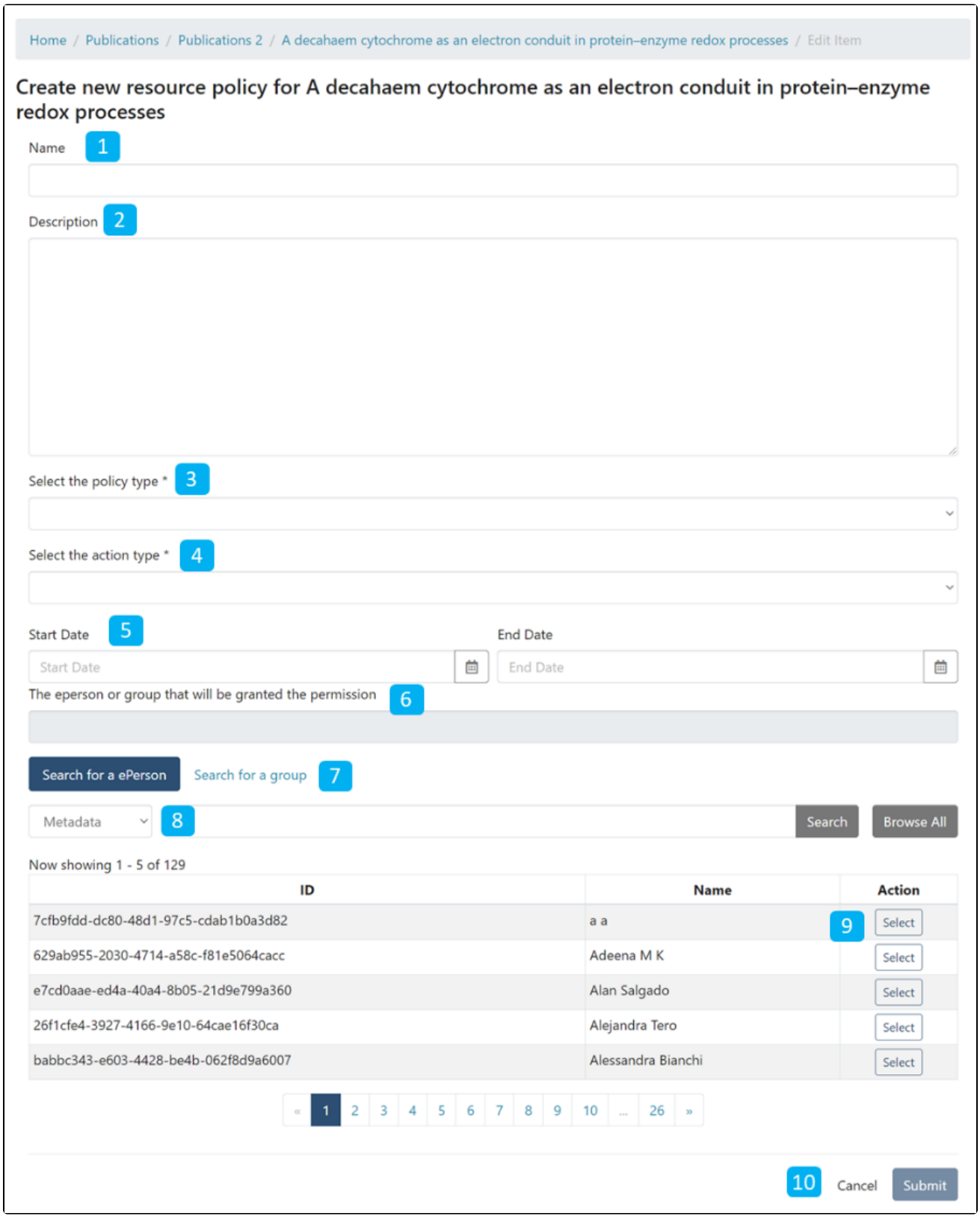
> *Users can create different policies for both Item and bitstreams. These are:*
>
> 1. *Managing ADD/REMOVE/READ/WRITE policies for an item*
> 2. *Managing READ/WRITE policies for a bitstream*

Add Authorization Policy

Step 1: Click the "Add" button to create a new resource policy for the Item.



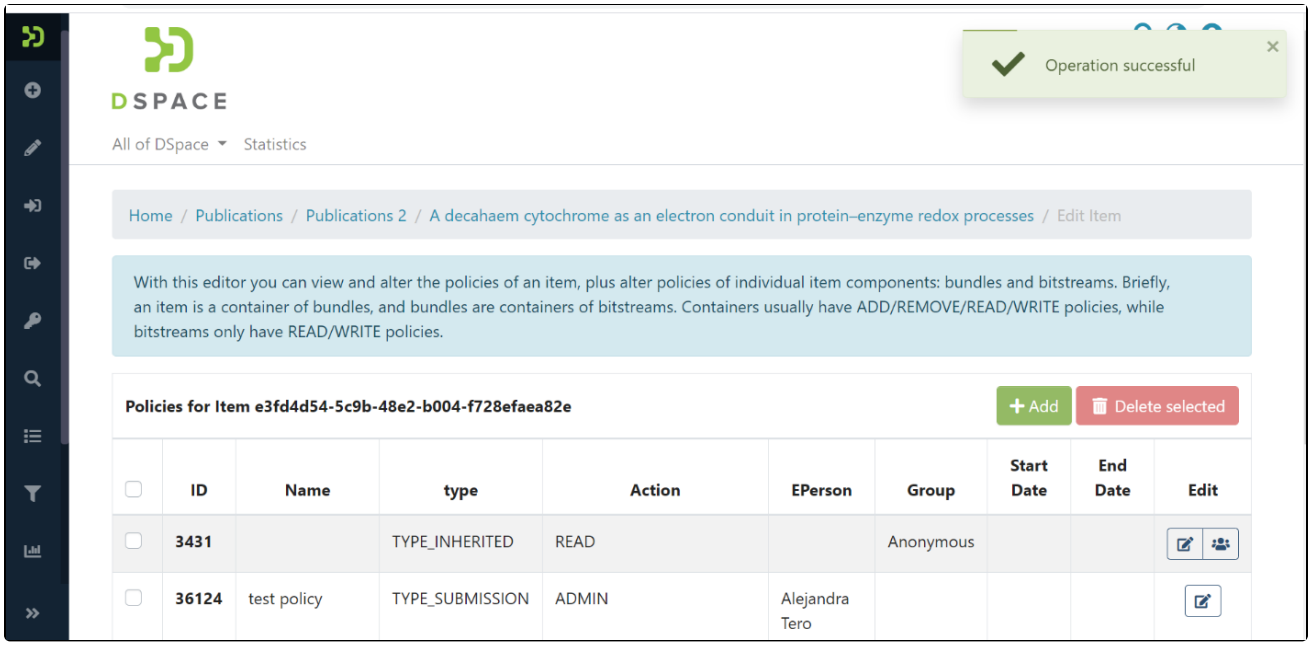
Step 2: Users can populate information on the "Create new resource policy" page about the policy and click on the "submit" button. Please see the description of each field followed by the below screenshot.



1. **Name:** Enter Policy name.
2. **Description:** Enter the policy description for future reference and understanding by other users.
3. **Select the policy type:** The user can select one of the following policy types:
    a. TYPE_SUBMISSION: a policy in place during the submission
    b. TYPE_WORKFLOW: a policy in place during the approval workflow
    c. TYPE_INHERITED: a policy that has been inherited from a container (the collection)
    d. TYPE_CUSTOM: a policy defined by the user during the submission or workflow phase

    e. null: if the information is not available
4. **Select the action type:** The user can select one of the following actions from the drop-down list:
    a. READ - Ability to view the specific object
    b. WRITE - Ability to edit the specific object
    c. ADD - Ability to add an object to a parent object (e.g. add a Bitstream to a Bundle, add a Bundle to an Item, add an Item to a Collection , add a Collection to a Community, etc.)
    d. REMOVE - Ability to remove an object from a parent object (i.e. the opposite of "ADD")
    e. ADMIN - Full Admin permissions on the specific object
    f. DELETE - Ability to delete the specific object
    g. WITHDRAWN_READ - Ability to view a withdrawn Item (only valid for Items)
    h. DEFAULT_BITSTREAM_READ - Default "READ" permissions added to any Bitstreams of a newly submitted Item (only valid for Collections)
    i. DEFAULT_ITEM_READ - Default "READ" permissions added to any newly submitted Item (only valid for Collections)
5. **Start date – end date:** The user can select a start date and end date for using the policy, should they want to apply it for a fixed period.
6. **The eperson or group granted the permission is the l**ist of users/groups granted authorization under the policy.
7. **Search for an ePerson / Search for a group:** Click on the "ePerson" or "group" to search for the entity.
8. **Search field:** Enter keywords to search for the ePerson/Group.
9. **ePerson/Group list:** Click on the select button against the user/group you want to add to the policy
10. **Submit/Cancel button:** Click on the "Submit" button to complete policy creation or click on the Cancel button to cancel the entire process.

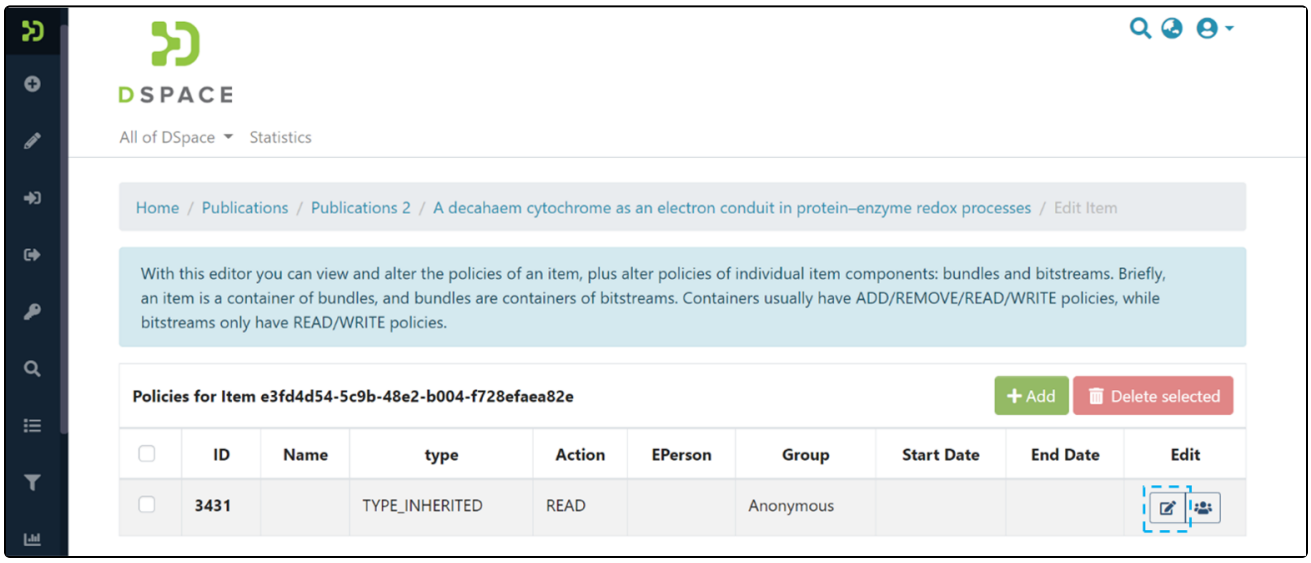
As shown below, users will see a success prompt upon the policy creation and be back on the Manage Policies screen.



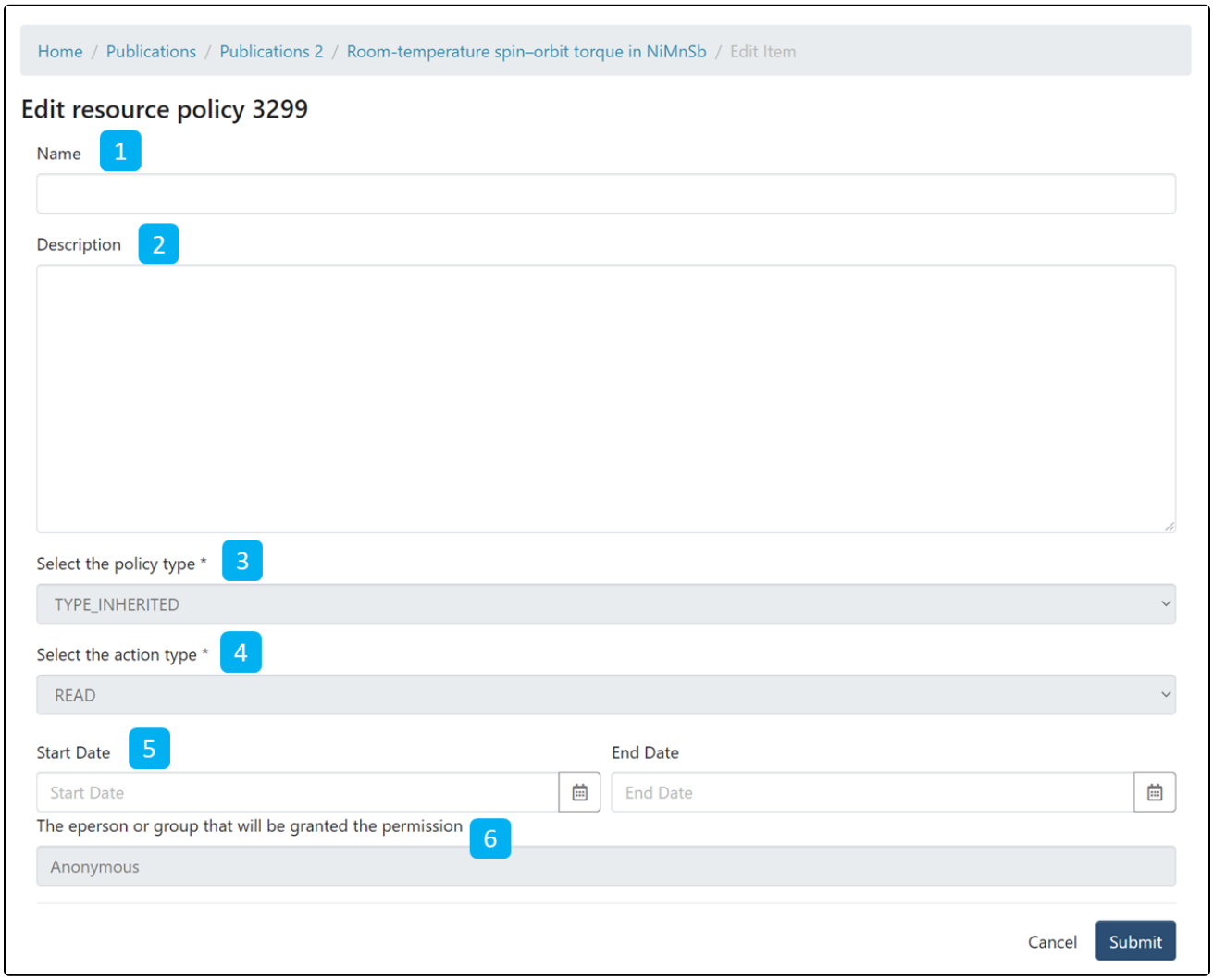
Manage Policy

Step 1: Click on the Edit policy icon appearing against each policy to update it.

The user group button next to the Edit policy icon will take users to the user group management. Please refer concerned section for more details.



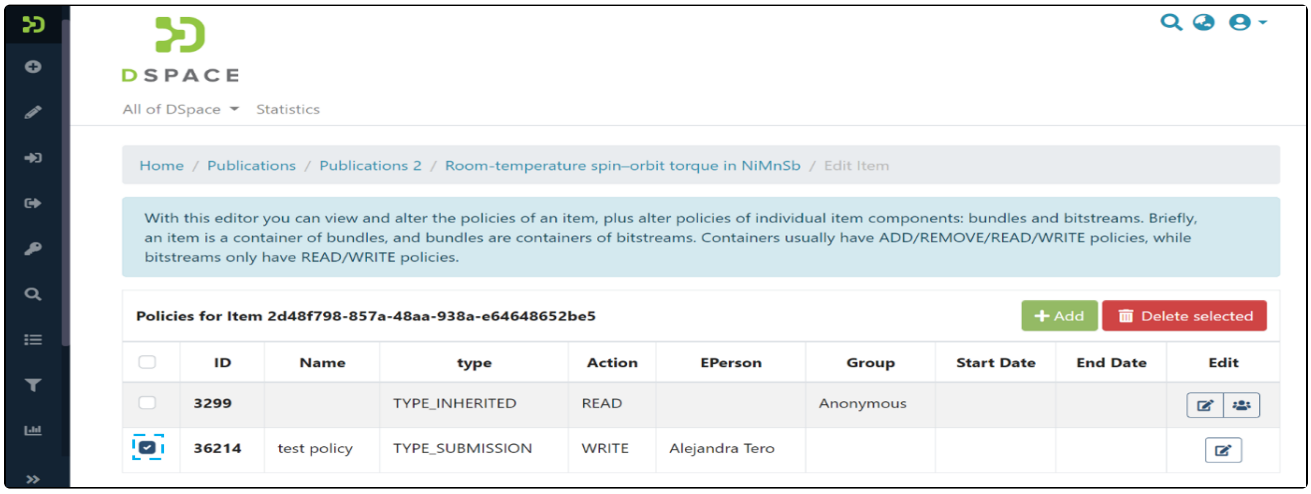
Step 2: Update policy information on the "Edit resource policy" page and click the "Submit" button. Please see the description of each field appearing on the "Edit resource policy" page.



1. **Name:** Enter Policy name.
2. **Description:** Enter the policy description for future reference and understanding by other users.
3. **Select the policy type:** The user can select one of the following policy types:
    a. TYPE_SUBMISSION: a policy in place during the submission

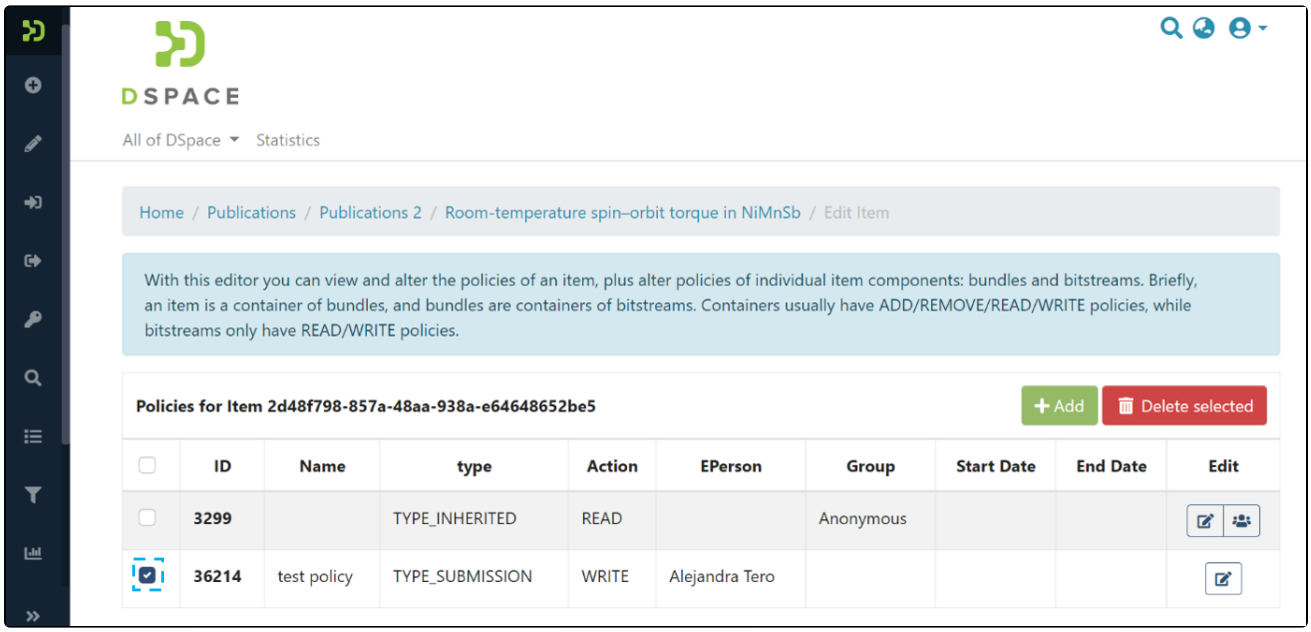
   b. TYPE_WORKFLOW: a policy in place during the approval workflow
   c. TYPE_INHERITED: a policy that has been inherited from a container (the collection)
   d. TYPE_CUSTOM: a policy defined by the user during the submission or workflow phase
   e. null: if the information is not available
4. **Select the action type:** The user can select one of the following actions from the drop-down list:
   a. READ - Ability to view the specific object
   b. WRITE - Ability to edit the specific object
   c. ADD - Ability to add an object to a parent object (e.g. add a Bitstream to a Bundle, add a Bundle to an Item, add an Item to a Collection , add a Collection to a Community, etc.)
   d. REMOVE - Ability to remove an object from a parent object (i.e. the opposite of "ADD")
   e. ADMIN - Full Admin permissions on the specific object
   f. DELETE - Ability to delete the specific object
   g. WITHDRAWN_READ - Ability to view a withdrawn Item (only valid for Items)
   h. DEFAULT_BITSTREAM_READ - Default "READ" permissions added to any Bitstreams of a newly submitted Item (only valid for Collections)
   i. DEFAULT_ITEM_READ - Default "READ" permissions added to any newly submitted Item (only valid for Collections)
5. **Start date – end date:** The user can select a start date and end date for using the policy, should they want to apply it for a fixed period.
6. **The eperson or group granted the permission is the l**ist of users/groups granted authorization under the policy.
7. **Search for an ePerson / Search for a group:** Click on the "ePerson" or "group" to search for the entity.
8. **Search field:** Enter keywords to search the ePerson/Group.
9. **ePerson/Group list:** Click on the select button against the user/group you want to add to the policy
10. **Submit/Cancel button:** Click on the "Submit" button to complete policy creation or click on the Cancel button to cancel the entire process.

As shown below, users will see a success prompt upon the policy creation and be back on the Manage Policies screen.



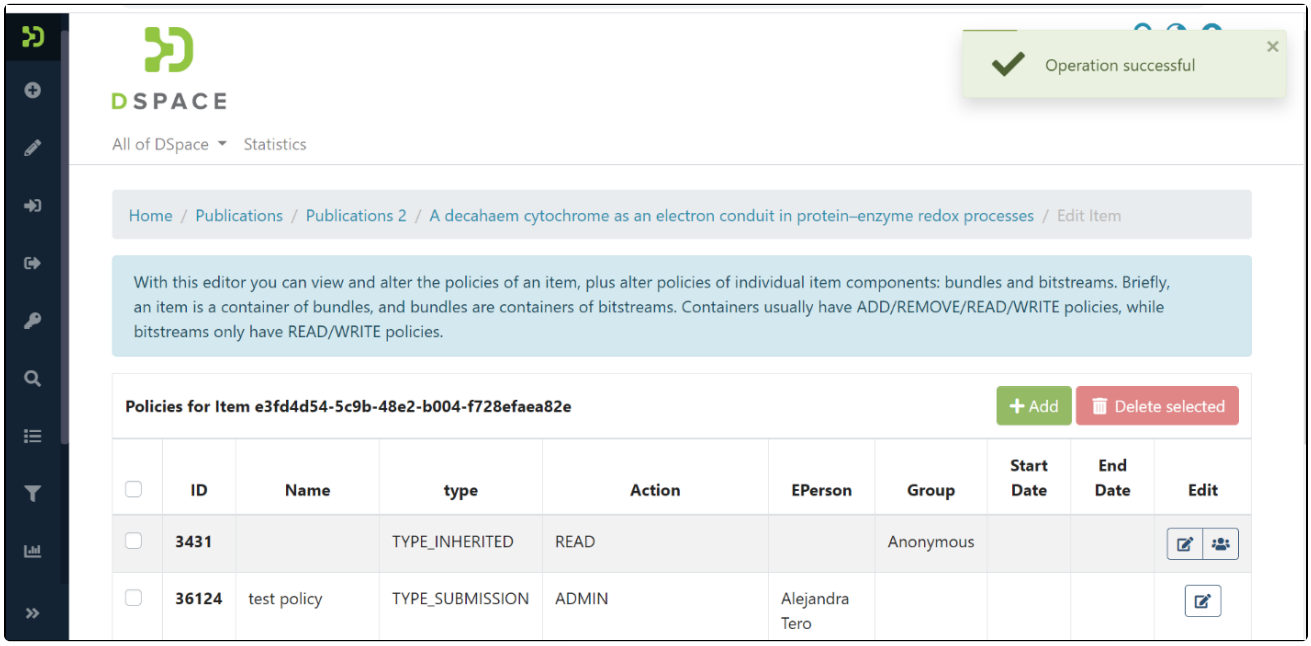
Delete Policy

Step 1: Click on the check box on the left-hand side of each policy, and the "Delete Selected" button will be activated.



Step 2: Click the "Delete selected" button to delete the policy. Please note that the deleted policy is irrecoverable.

You'll see a success prompt upon deletion of the selected policy.



Collection Mapper

- Scope and Use Case
- Audience
- Collection Mapping
    - Map the item to a new collection
    - Manage existing collections the item is mapped to

Scope and Use Case

Items live in one collection but can be mapped to additional collections. The item page will display all of the collections the item lives in or is mapped to. The item will appear in the lists of items in each collection it lives in or is mapped to.

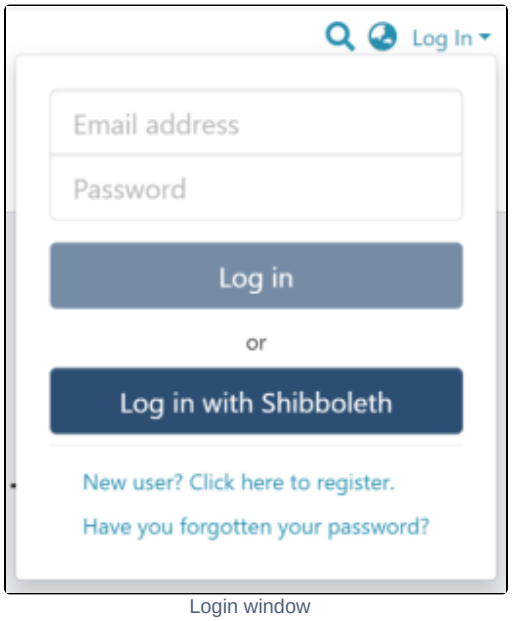
This process describes how to map items to collections and how to manage existing mappings.

Audience

- Repository Administrators
- Community Administrators
- Collection Administrators (to map an item, a user must be a collection administrator for both the collection the item lives in and the collection it is being mapped to)

Collection Mapping

Step 1: Login using your DSpace credentials.



Login window

Step 2: Go to the item you want to edit.

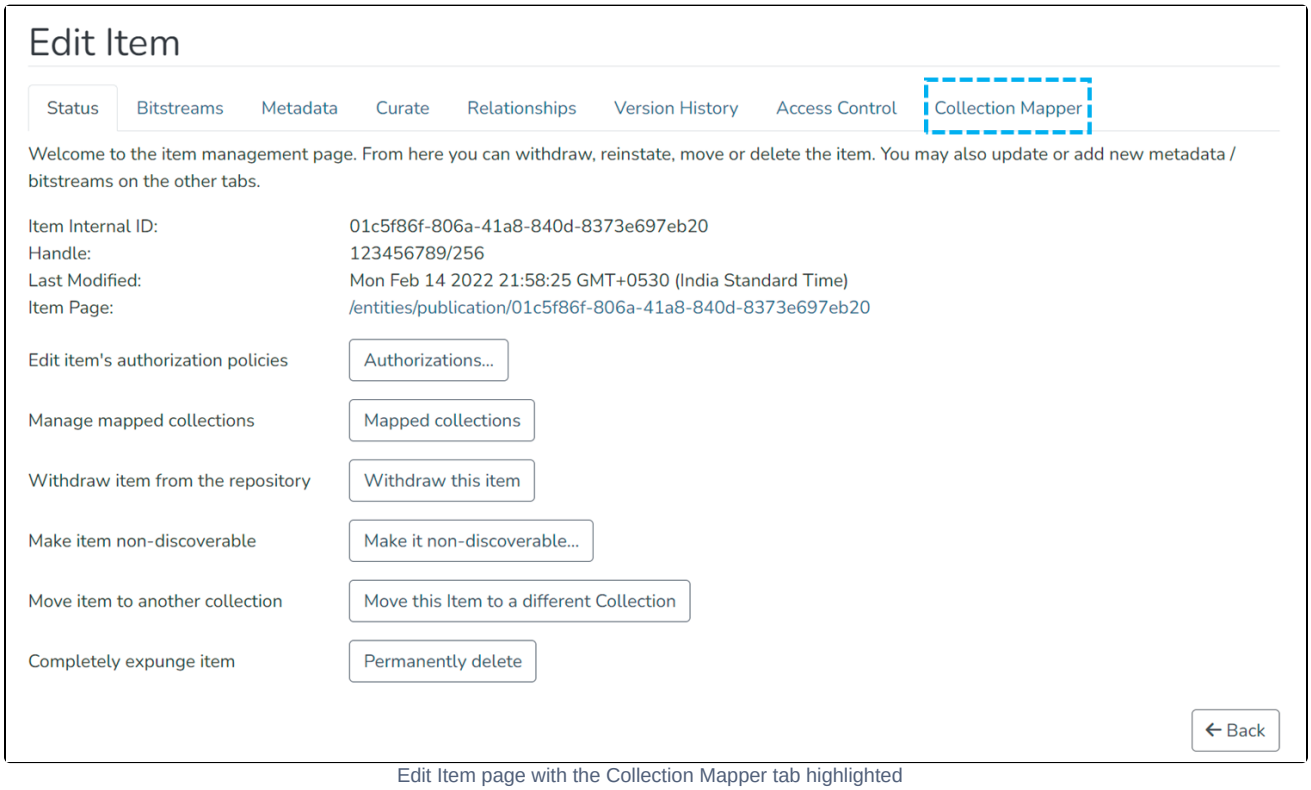
Users can reach an item through multiple methods, including those listed below:

1. Search for an item
2. Browse communities and collections
3. Find an item in the Administration section at Edit > Item
4. Use the existing item URL

Step 3: Click the "Edit" button as highlighted on the screen below. This button will appear to any user who has edit rights on the target item.



Step 4: Click the "Collection Mapper" tab.

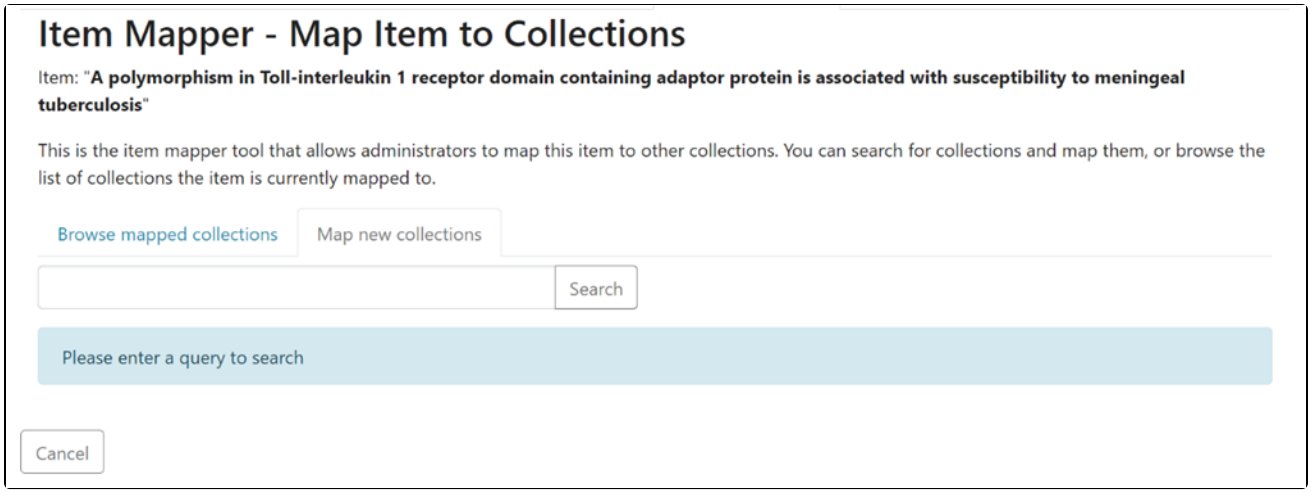


Edit Item page with the Collection Mapper tab highlighted

> *Users can perform multiple functions in the Collection Mapper tab. These are:*
>
> 1. *Map the item to a new collection.*
> 2. *Manage existing collections the item is mapped to.*

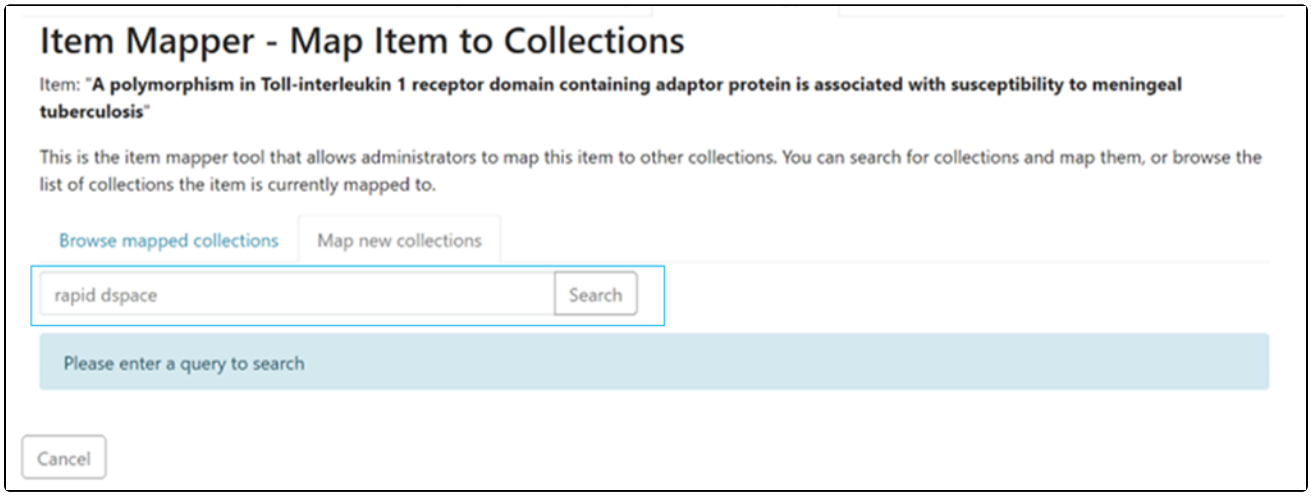
Map the item to a new collection

Step 1: Click on the “Map new collections” tab.



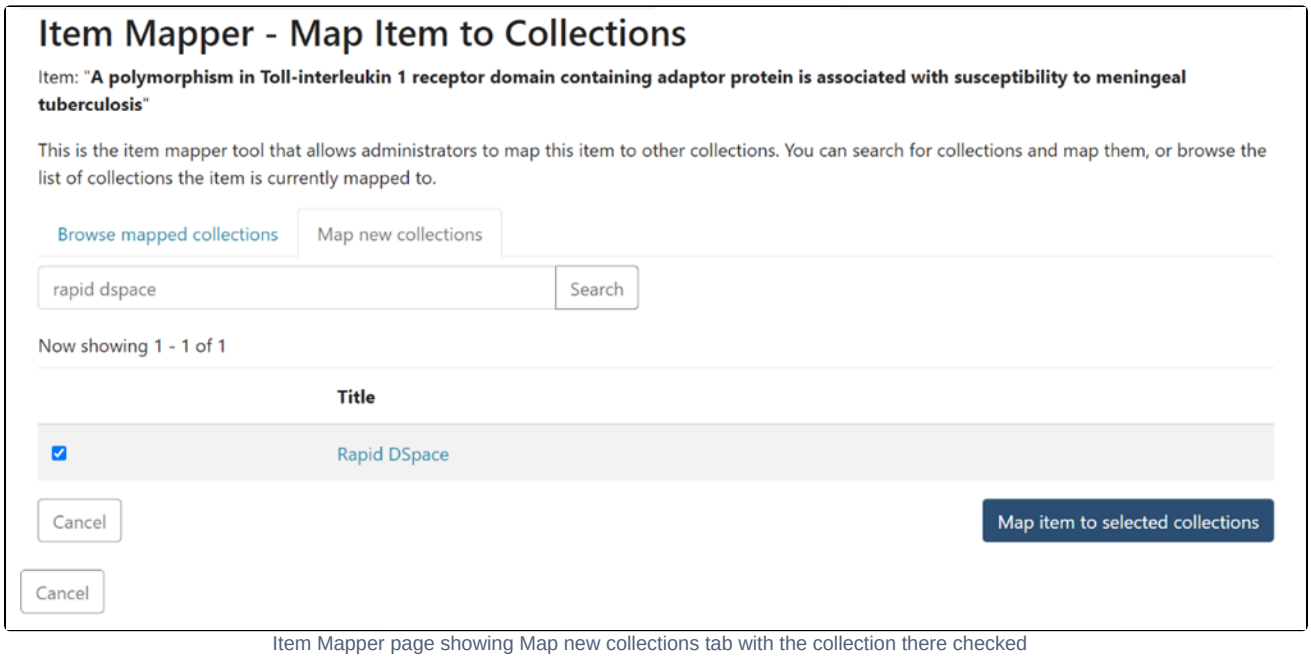
Item Mapper page showing Map new collections section

Step 2: Enter the name of the collection you want to map this Item to and click on the “Search” button.



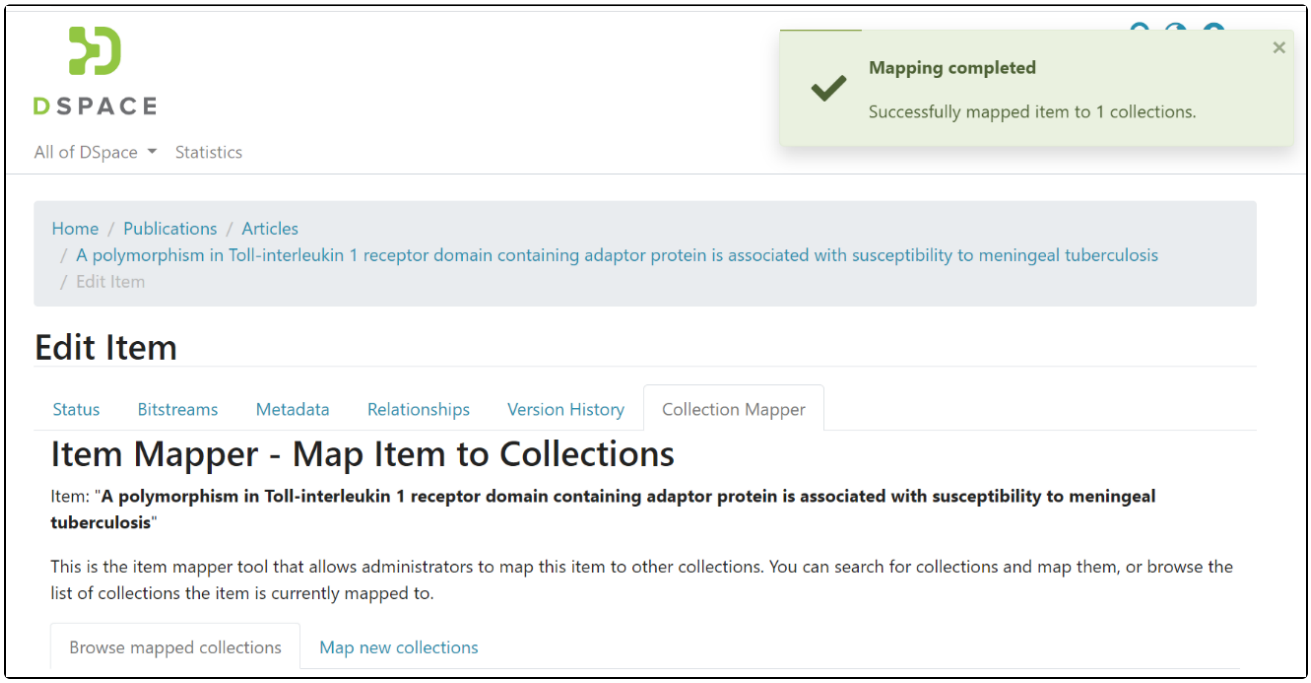
Item Mapper page with text in the Map new collections searchbar

Step 3: Click the checkbox on the left of the target collection to select it. Click the “Map item to selected collections” button to complete mapping, or use the “Cancel” button to cancel the operation.



Item Mapper page showing Map new collections tab with the collection there checked

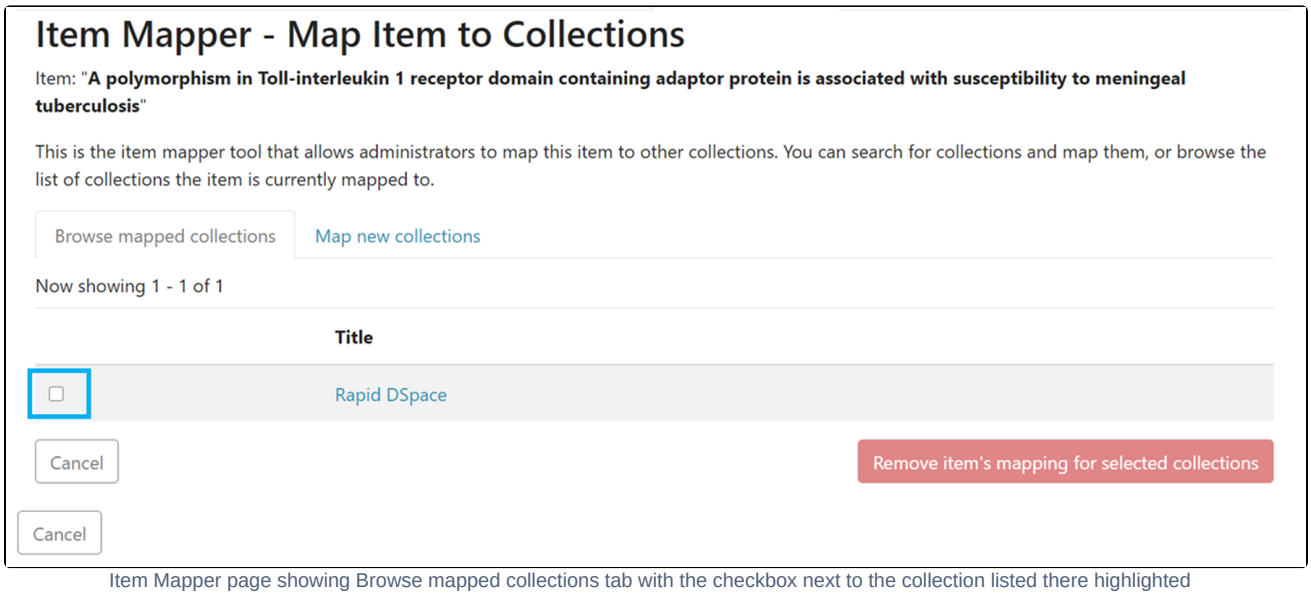
A success prompt confirming the collection mapping will appear. The selected collection will appear under the “Browse mapped collections” tab.



Banner saying Mapping completed

Manage existing collections the item is mapped to

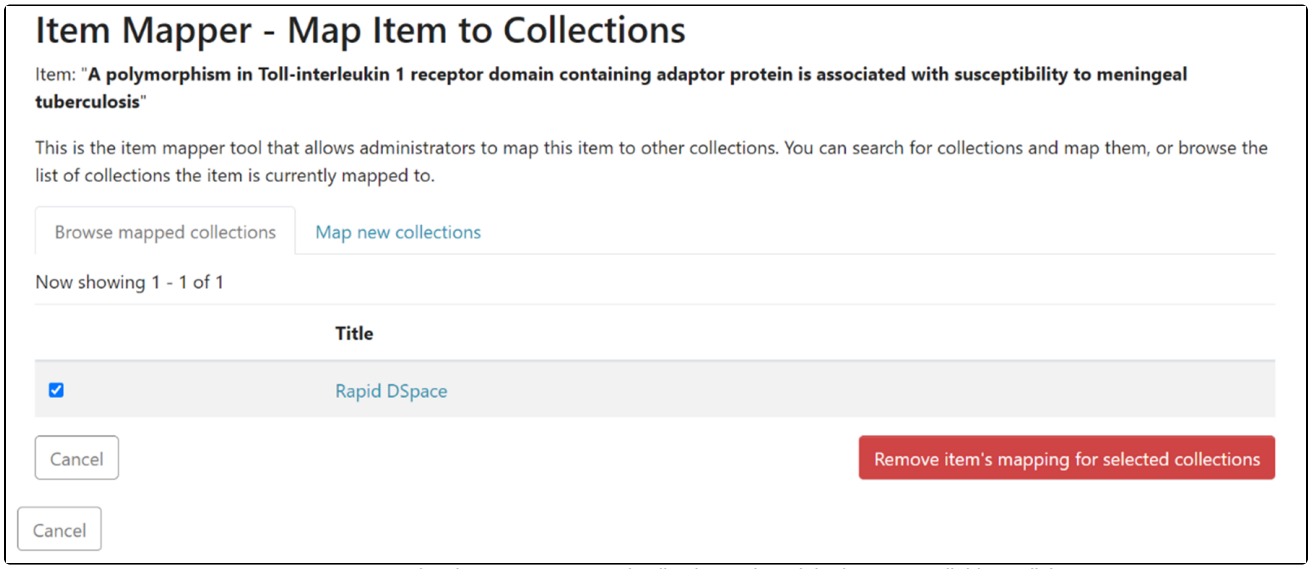
Step 1: On the “Browse mapped collections” tab, to delete an existing collection the item is mapped to, click the checkbox on the left of the collection name to select it.



Item Mapper page showing Browse mapped collections tab with the checkbox next to the collection listed there highlighted

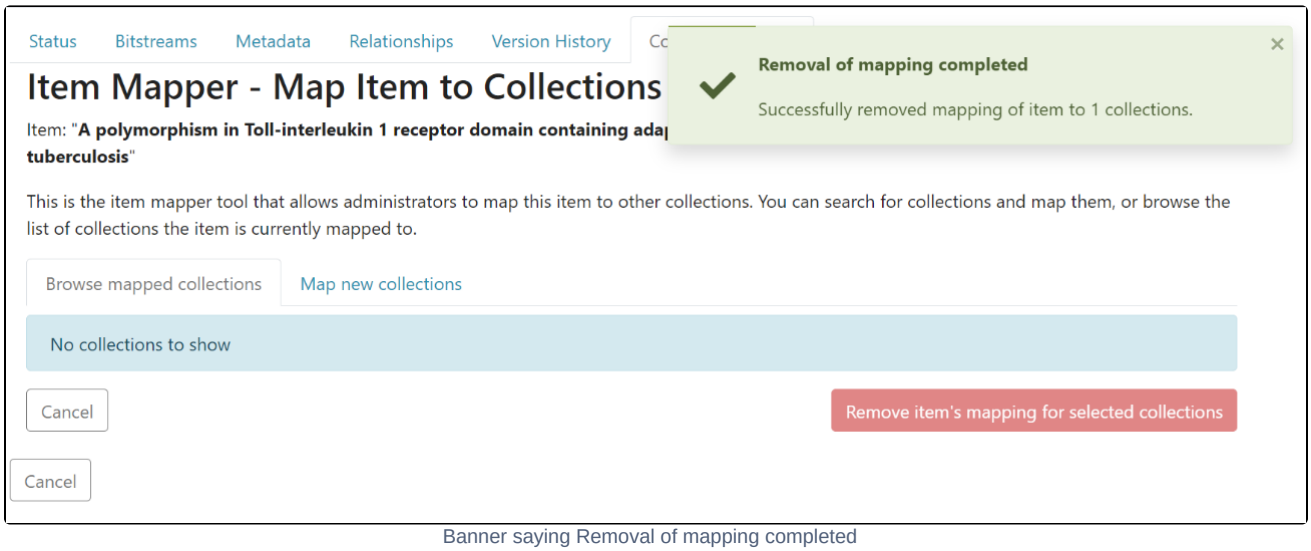
The "Remove item's mapping for selected collections" button will activate upon selecting the collection.

Step 2: Click on “Remove item’s mapping for selected collections” to unmap selected collection(s) or click “Cancel” to cancel the operation.



Item Mapper page showing Browse mapped collections tab and the buttons available to click

A prompt confirming the successful unmapping of the collection will appear. The selected collection will disappear from the list in the “Browse mapped collections” tab.



Banner saying Removal of mapping completed

Edited and approved by DCAT Documentation Working Group, December 2024.

Edit Bitstream

- Scope and Use Case
- Audience
- Edit Bitstream Process
  - Add a new bundle containing a new bitstream
- Edit a Bitstream or Bundle in an item

Scope and Use Case

Bitstreams are files attached to items in DSpace. After an item is submitted, the bitstream(s) attached to it can be edited. New bitstreams can be added and existing bitstreams can be edited or deleted.

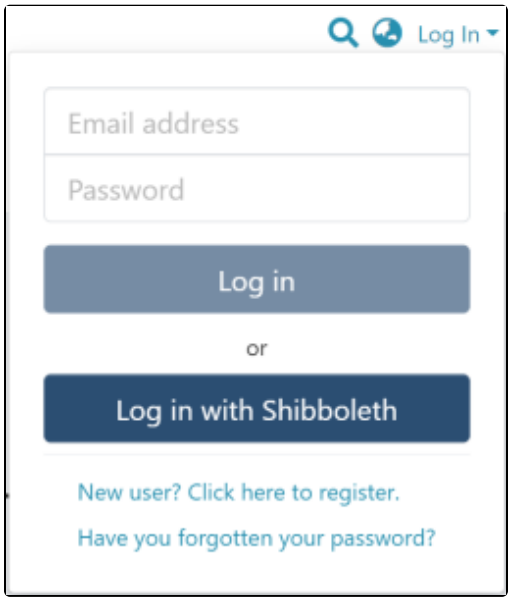
Bitstreams are added in bundles (e.g., Original, which contains the original file(s); License, which contains any license associated with the file(s); Text, which contains any file(s) with extracted text for searching; Thumbnail, which contains any thumbnail image(s) of the original file(s)). New bundles can be added and the bitstreams in existing bundles can be edited or deleted.

Audience

- Repository Administrators
- Community Administrators
- Collection Administrators

Edit Bitstream Process

Step 1: Login using your DSpace credentials.



Login window

Step 2: Go to the item you want to edit.

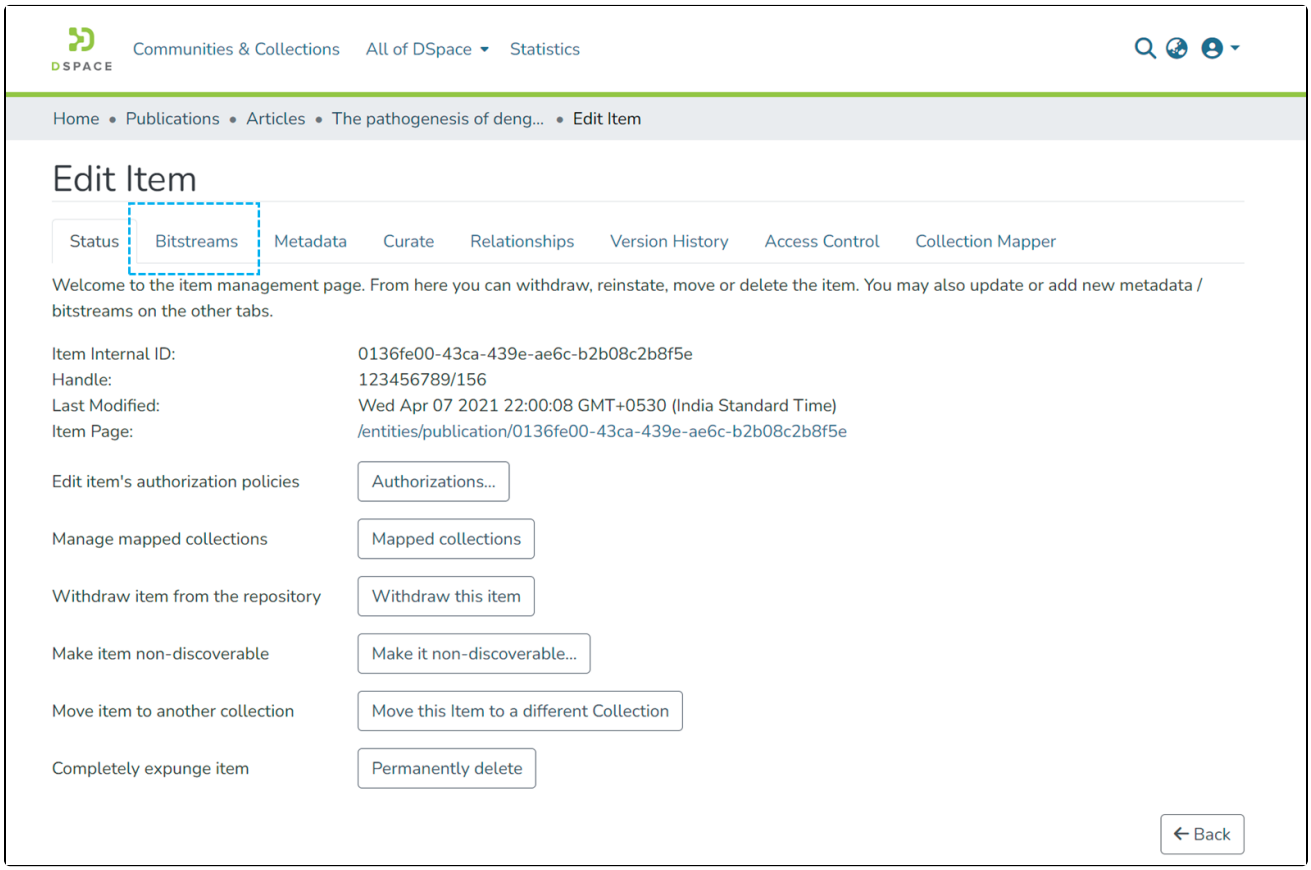
Users can reach an item through multiple methods, including those listed below:

1. Search for an item
2. Browse communities and collections
3. Find an item in the Administration section at Edit > Item
4. Use the existing item URL

Step 3: Click on the “Edit” button as highlighted on the screen below. This button will appear to any user who has edit rights on the target item.



Step 4: Click on the “Bitstreams” tab to edit the bitstreams.



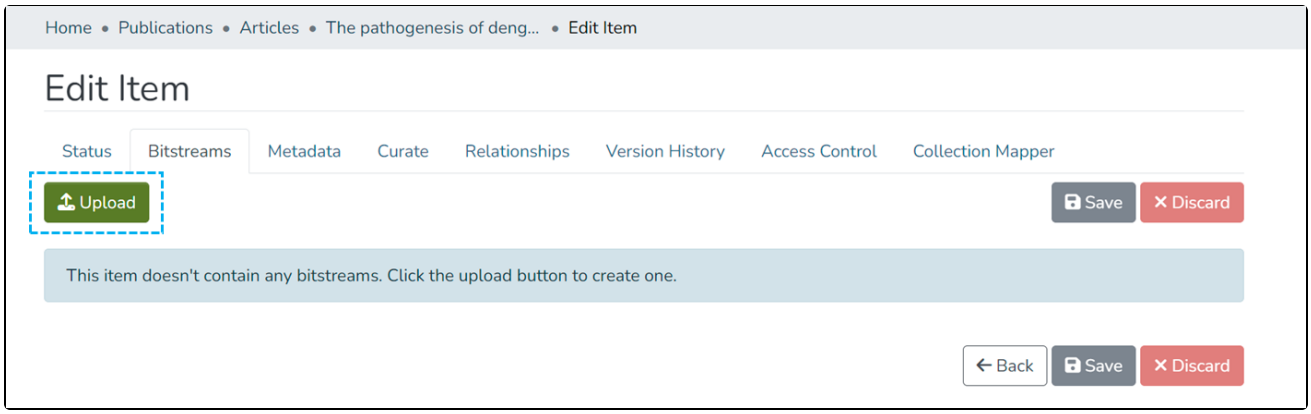
Edit Item page with the Bitstreams tab highlighted

From the Bitstreams tab you can:

1. Add a new bundle containing a new bitstream.
2. Add a new bitstream to an existing bundle.
3. Edit or delete an existing bitstream in an existing bundle.

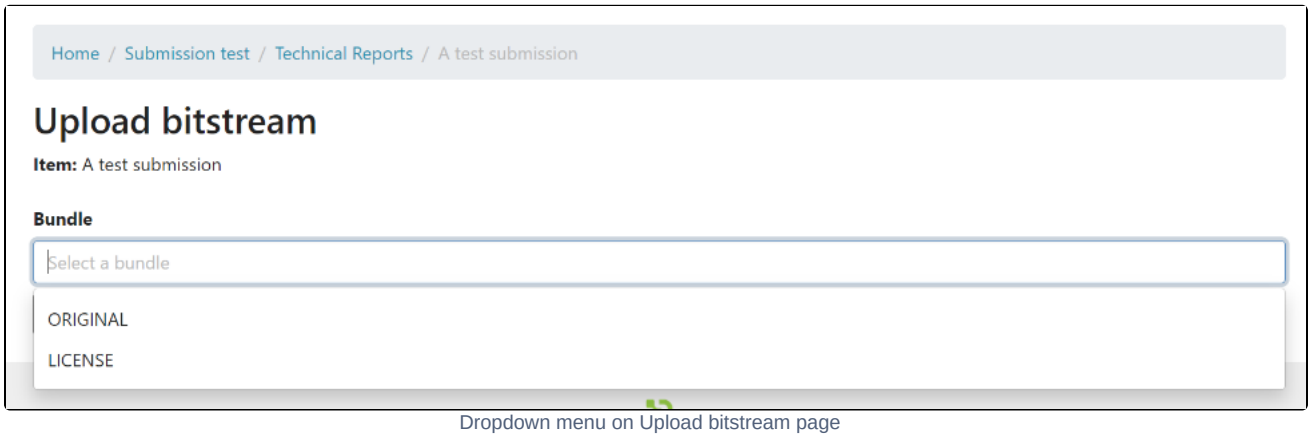
Add a new bundle containing a new bitstream

Step 1: Click on the “Upload” button to upload a bitstream in existing bundles or create a new bundle to add a bitstream.



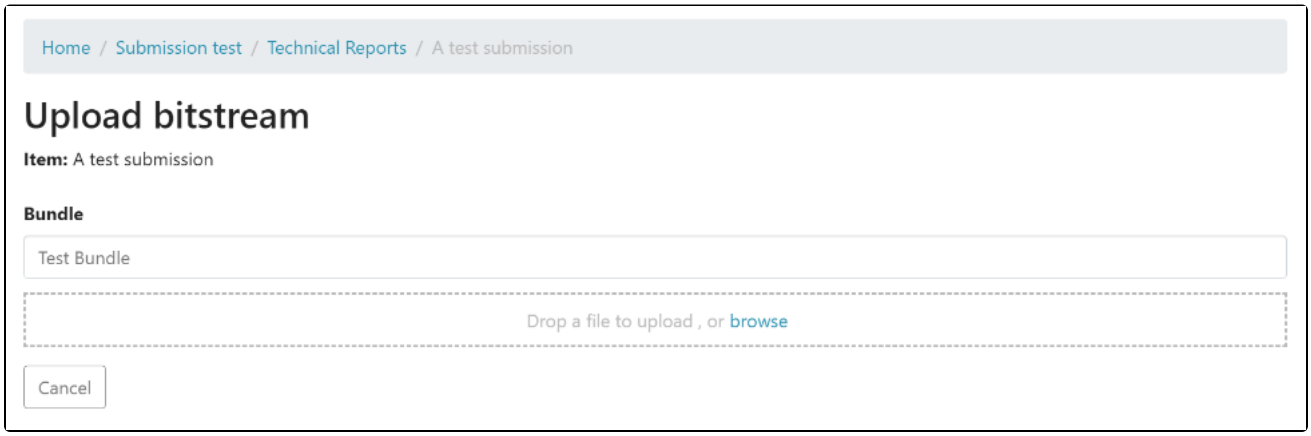
Edit Item page, Bitstreams tab, with Upload button highlighted

Step 2: Enter the bundle name or select an existing name from the dropdown menu.



Dropdown menu on Upload bitstream page

Step 3: Drag and drop the bitstream(s) you want to attach to the bundle, or click on the “browse” link appearing in the file upload section to locate the bitstream(s).

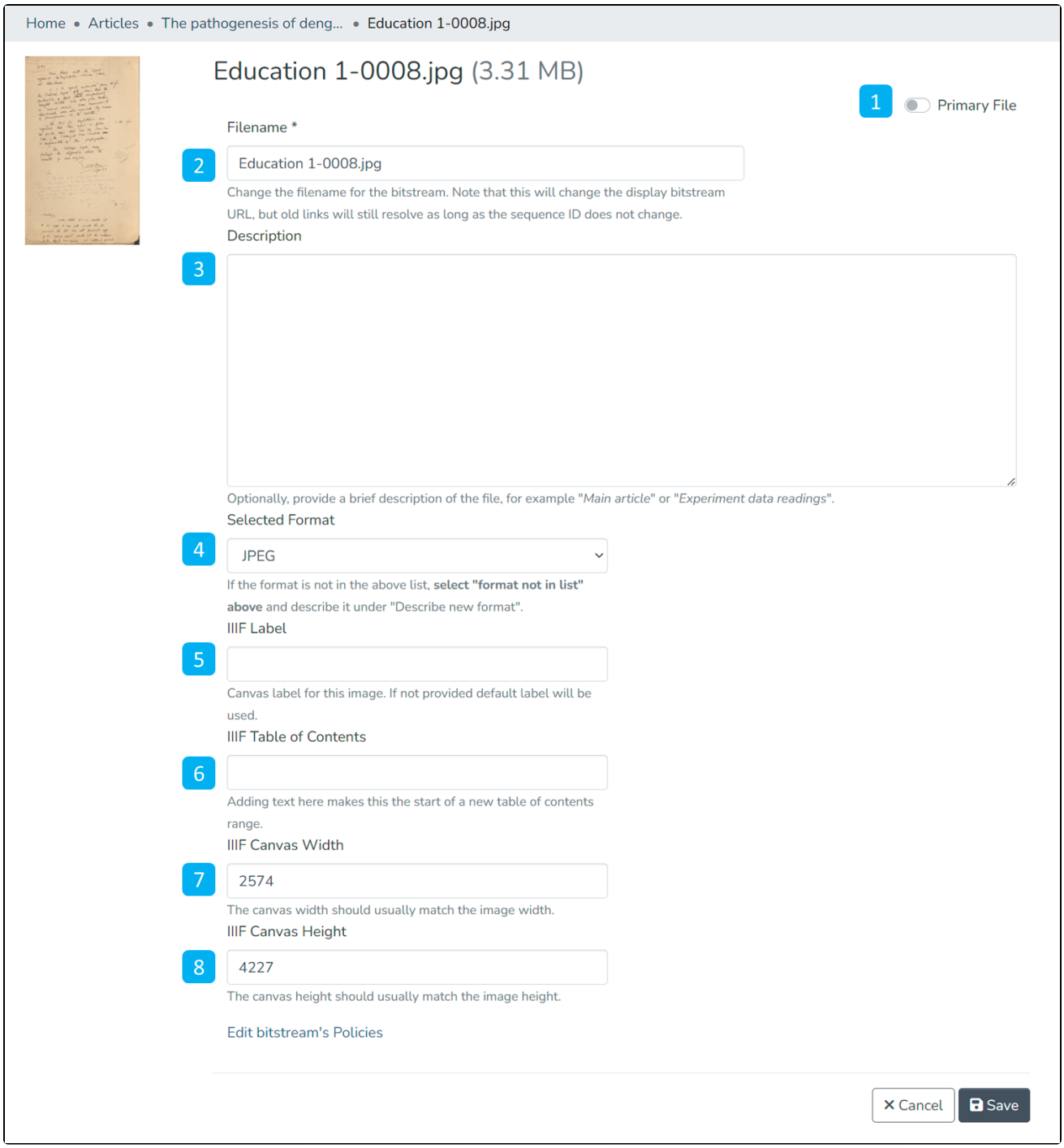


Upload bitstream page with file upload section

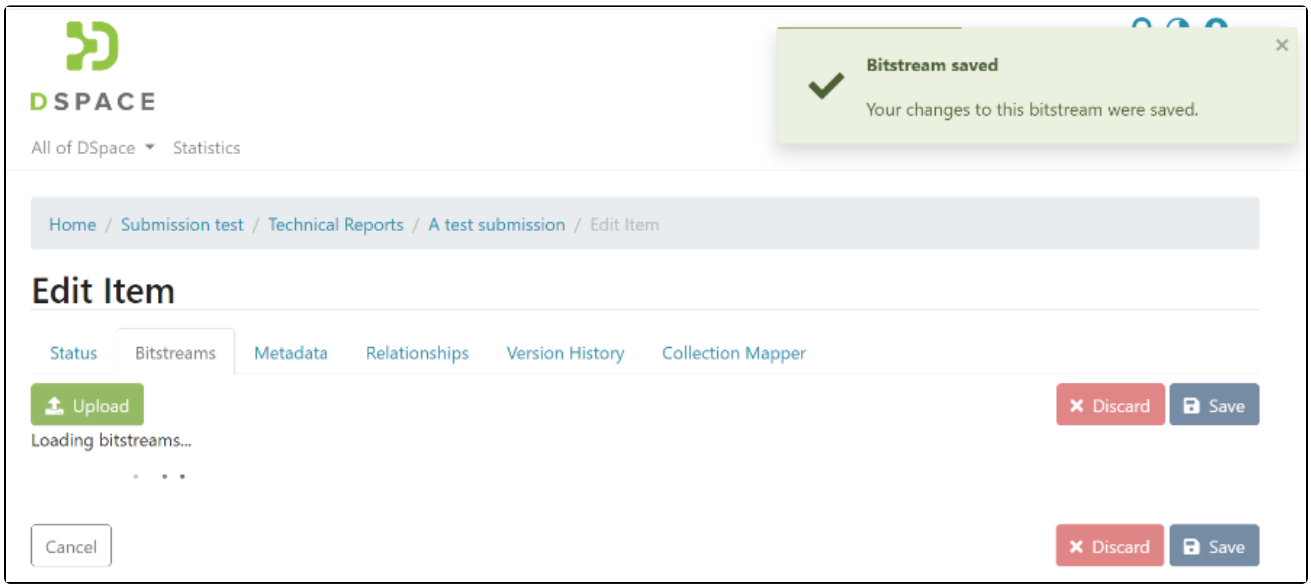
Step 4: After a successful bitstream upload, add more details about the bitstream on the next screen. Click the “Save” button at the bottom of the page to save details, or click the “Cancel” button to discard updates.

Below is the description of various fields appearing on this page

1. Primary file: Click the button to set the bitstream as the primary file for that bundle. If another bitstream was the primary file beforehand, it will be replaced with this one. The thumbnail for the primary file of the ORIGINAL bundle will be used as the main thumbnail for the item.
2. Filename: The default value appearing in this field is the bitstream's filename. However, users can replace it with the value of their choice.
3. Description: Users can add a description of the attachment in this field.
4. Embargo until date: Users can select a future date to restrict public access to the attachment. Additionally, users can grant access to a specific set of users by selecting user groups.
5. Selected format: If the file extension of the uploaded attachment exists in the DSpace's bitstream registry, the user will see the registered value for extension in this field. Users can change the value by selecting another one using the dropdown.

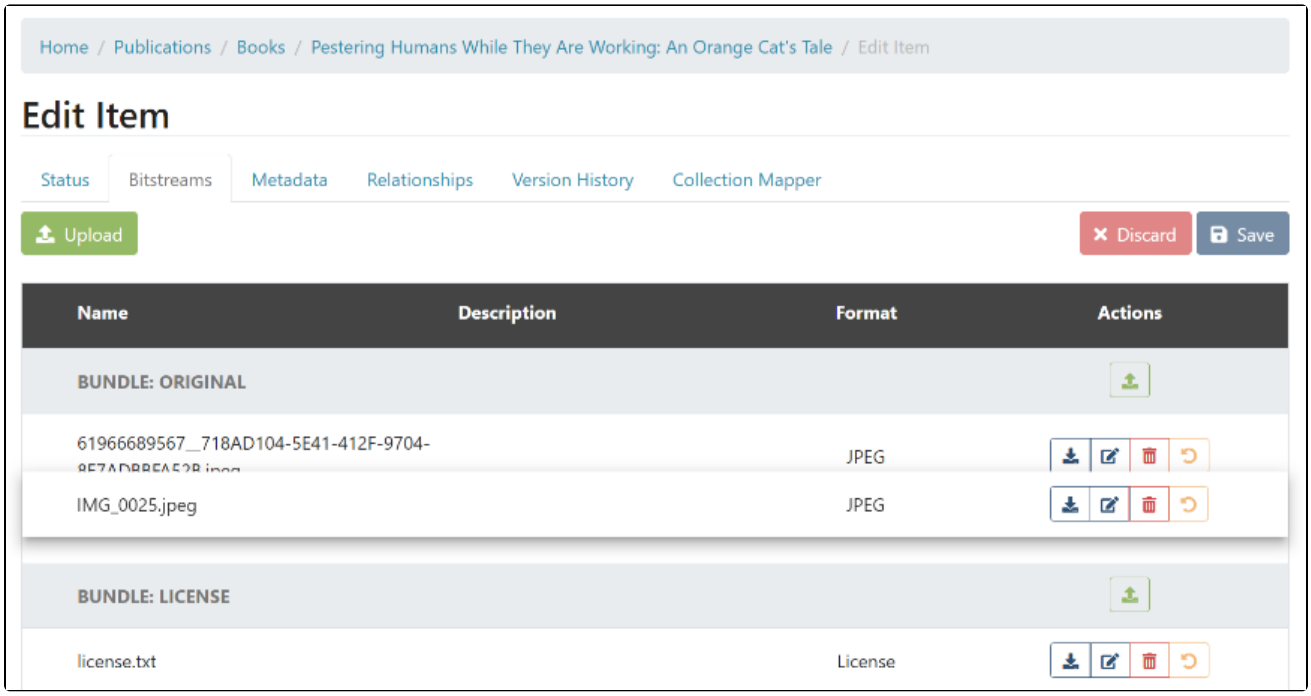


After clicking the "Save" button, the user will be redirected to the bitstream tab. A prompt confirming success or failure will appear.



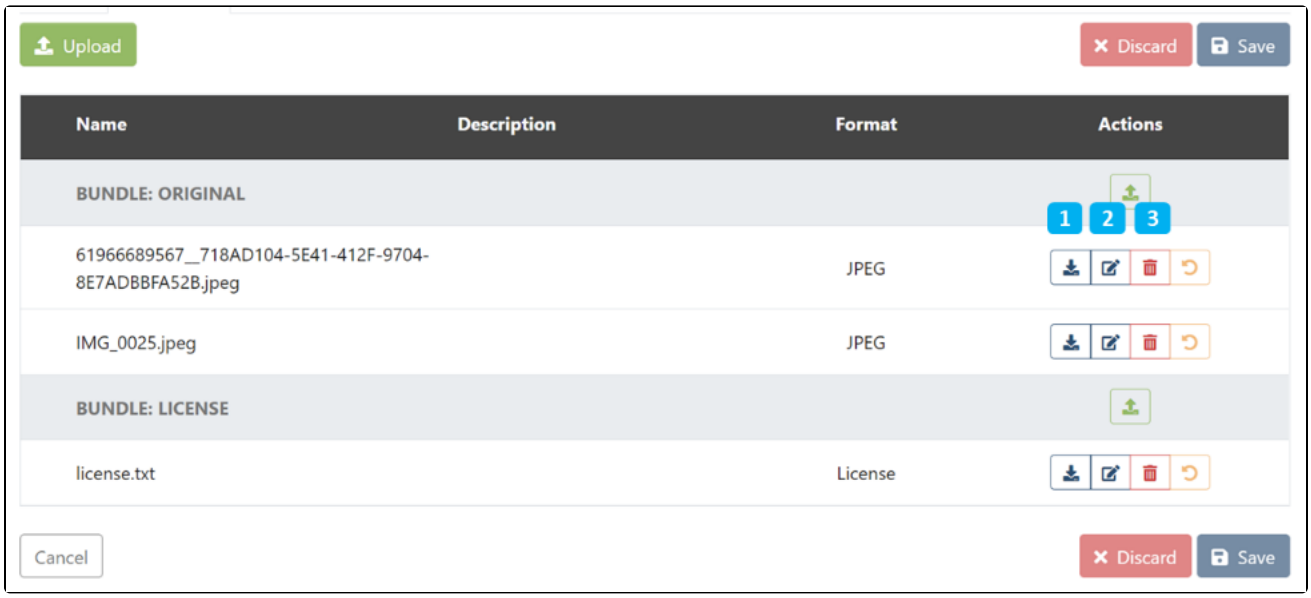
Edit a Bitstream or Bundle in an item

Step 1: Click on a bitstream and drag it above or below another bitstream(s) in the bundle to change the bitstream's sequence.



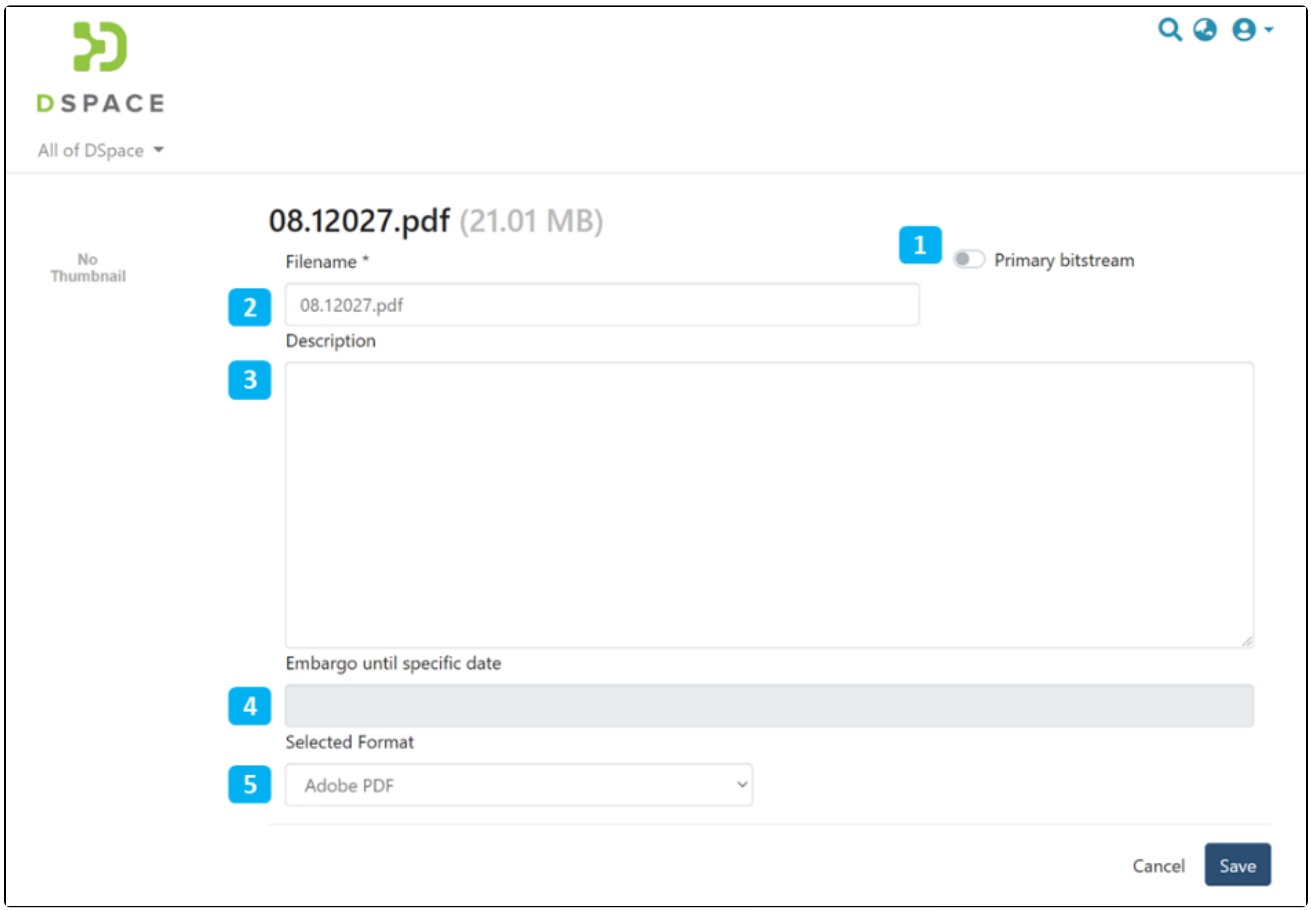
Step 2: Apart from adjusting the bitstream's sequence in a bundle, below are other options available:

1. Download Bitstream: Click the button to download the attachment on the local device.
2. Edit Bitstream: The "Edit bitstream" button is used to edit details. More details are given in the following steps.
3. Delete Bitstream: Click on the Delete Bitstream button to delete Bitstream from the bundle.

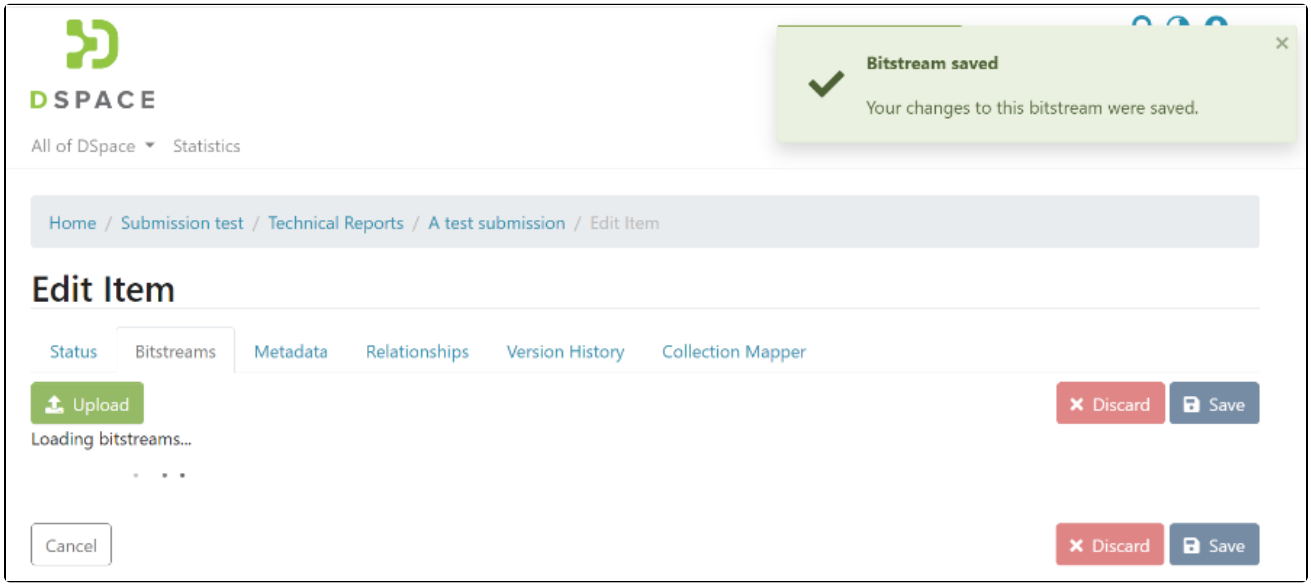


Step 3: Click the "Edit" button on the screen above to edit the bitstream details. Below is the description of various fields appearing on this form

1. Primary bitstream: Click on this button to set the bitstream as the primary bitstream for that bundle. If another bitstream was the primary bitstream beforehand, it will be replaced with this one. The thumbnail for the primary bitstream of the ORIGINAL bundle will be used as the main thumbnail for the item.
2. Filename: The default value appearing in this field is the attachment's filename. Users can replace it with the value of their choice.
3. Description: Users add a description of the attachment in this field.
4. Embargo until date: Users can select a future date to restrict public access to the attachment. Additionally, users can grant access to a specific set of users by selecting user groups.
5. Selected format: If the file extension of the uploaded attachment exists in the DSpace's bitstream registry, the user will see the registered value for extension in this field. Users can change the value by selecting another one using the dropdown.

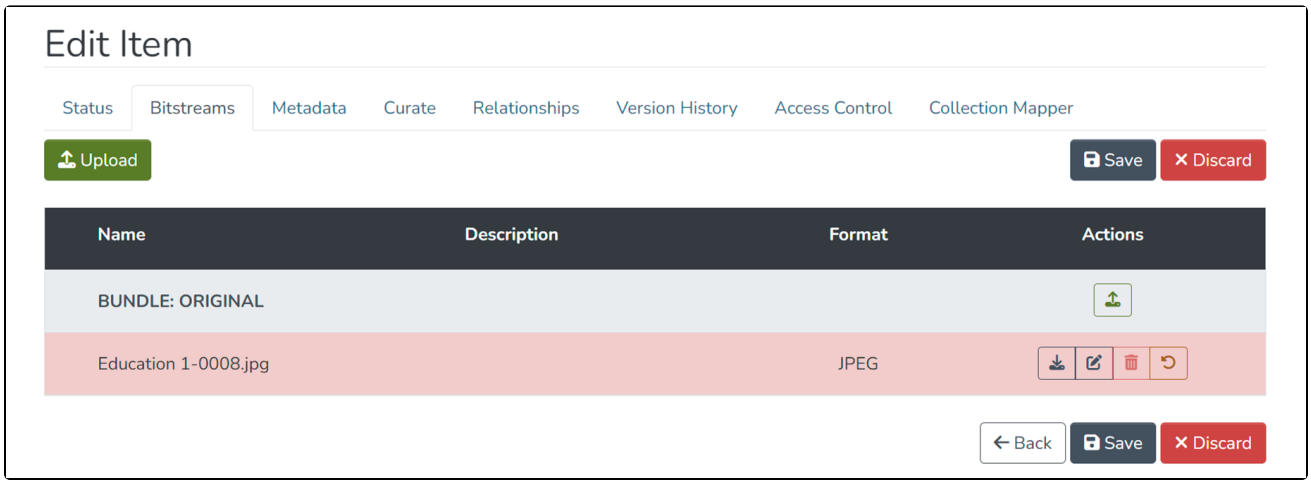


After clicking the “Save” button, the user will be redirected to the bitstream tab. A prompt confirming success or failure will appear.



Step 4: Use the "Delete" button to delete a bitstream. The attachment you want to delete will be highlighted in the red background for confirmation.

Click the “Save” button appearing below the bitstreams list to continue with the deletion. Otherwise, click the “Discard” button to cancel the process.

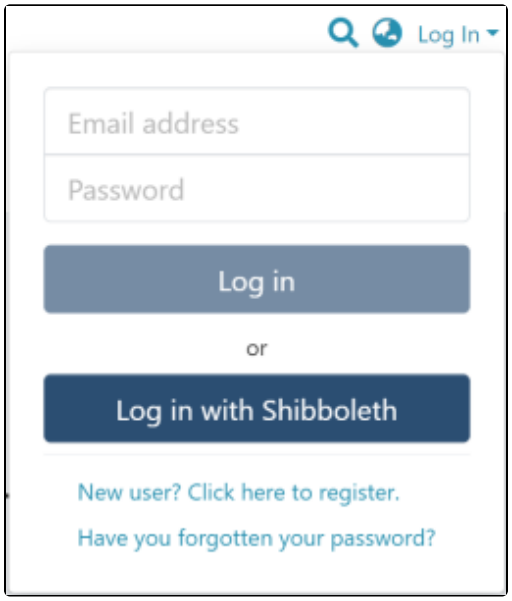

Edit Item
Status
Bitstreams
Metadata
Curate
Relationships
Version History
Access Control
Collection Mapper
Upload
Save
Discard
Name
Description
Format
Actions
BUNDLE: ORIGINAL
Education 1-0008.jpg
JPEG
Back
Save
Discard

Edit Metadata

- Edit Metadata Process
- Add a metadata field
- Edit an existing metadata field
- Delete an existing metadata field
- Add or edit authority-controlled metadata fields
    - Metadata fields controlled by a list of value-pairs
    - Metadata fields controlled by Hierarchical Taxonomies or Controlled Vocabularies
    - Authority-controlled metadata fields with authority keys
        - Editing or removing an authority key
        - Confidence values

Edit Metadata Process

Step 1: Login using your credentials



Step 2: Go to the Item you want to edit

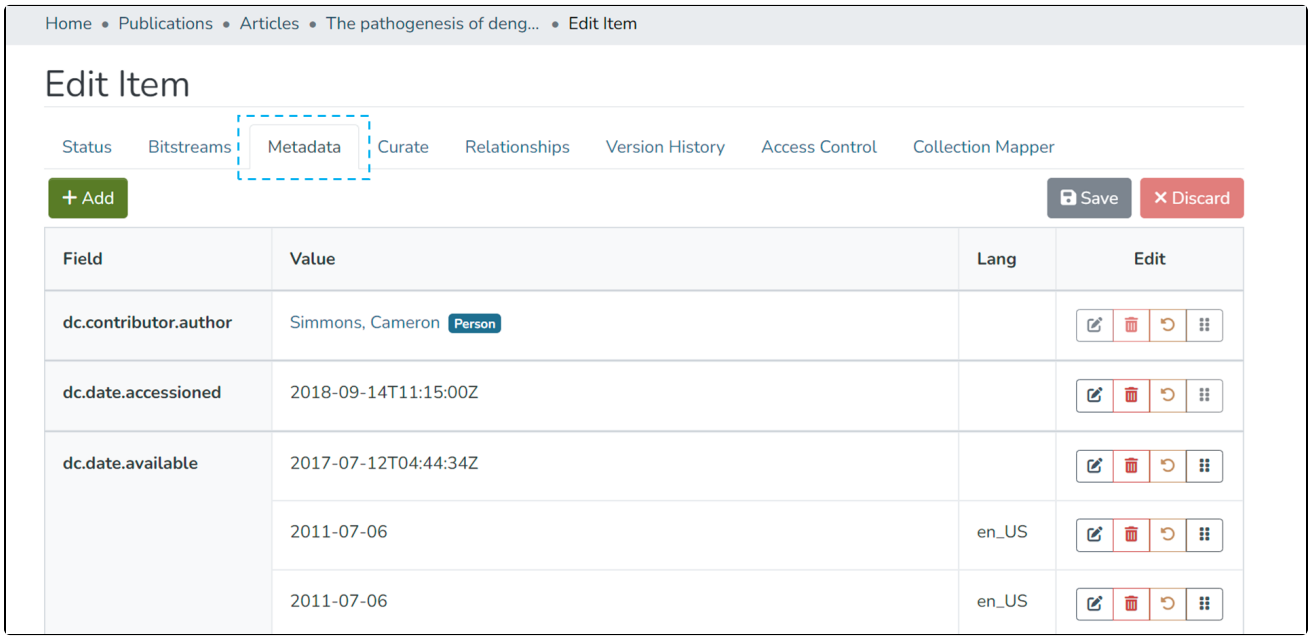
Users can reach an item through multiple methods, as listed below:

1. Search for an item
2. Browse communities and collections
3. Finding an item in the Administration section at Edit > Item

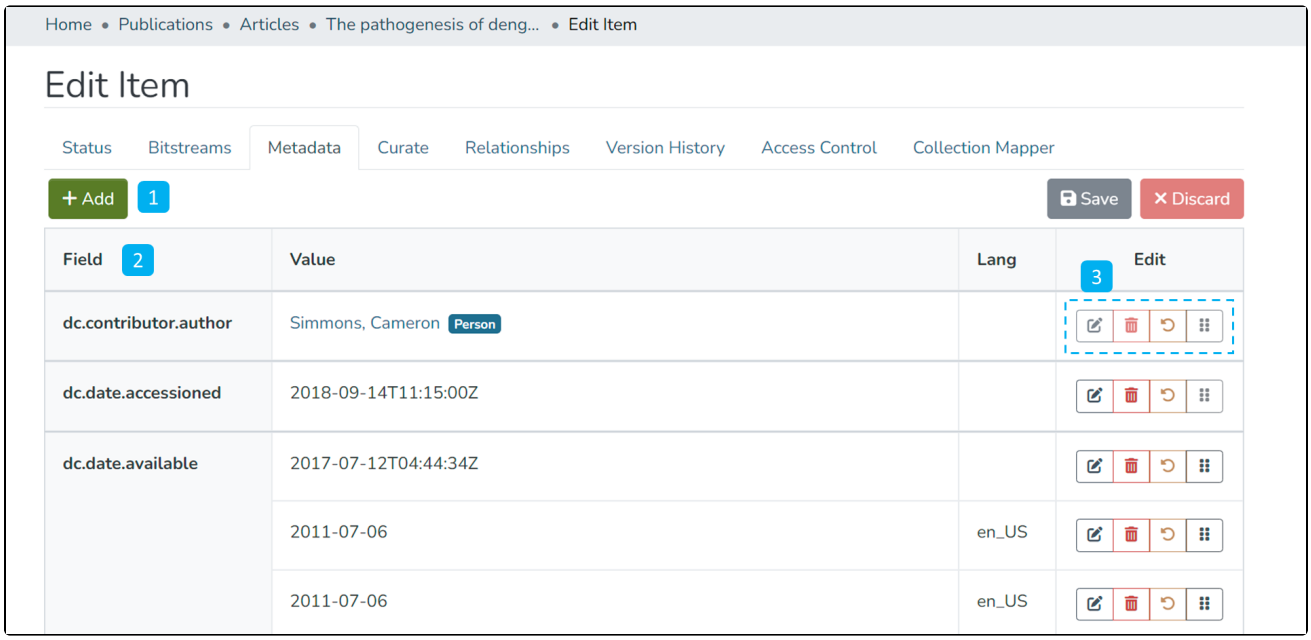
Click on the “Edit” button appearing on the right-hand side of the item title.



Step 3: Click on the “Metadata” tab to edit the metadata.



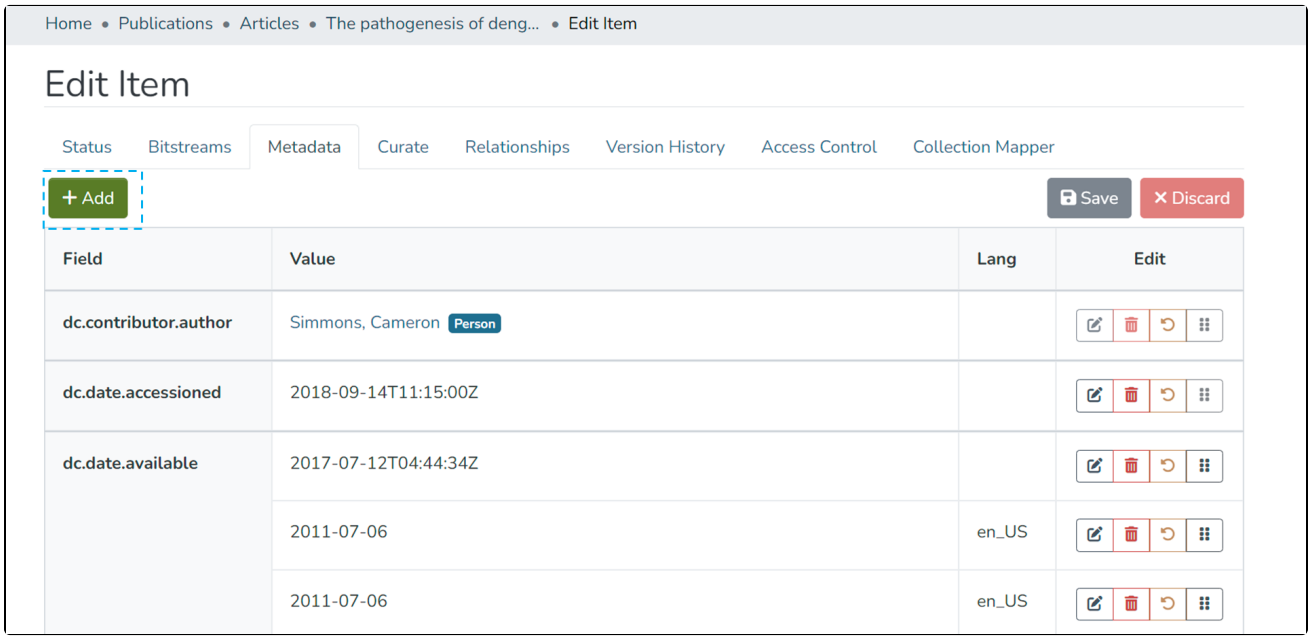
Step 4: Users can perform multiple actions in the Edit Metadata section, which are listed after the screenshot.



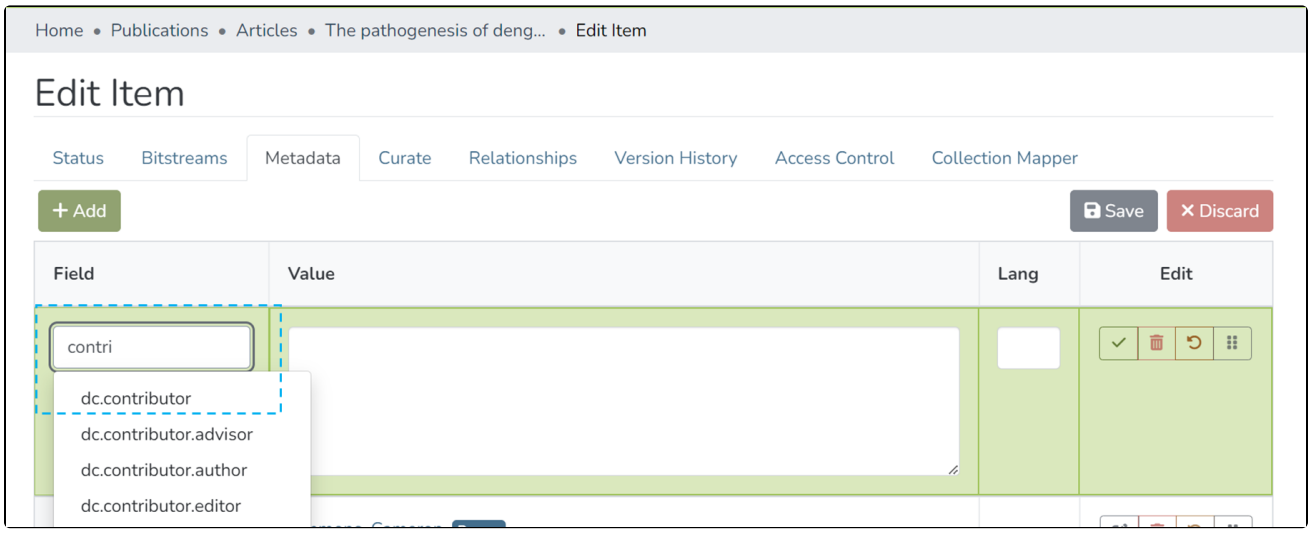
1. **Add:** Button used for adding new metadata elements in the existing Item.
2. **Metadata fields:** The fields column shows the metadata element's value appearing in the "Value" column.
3. **Edit:** This panel contains various options to update the specific metadata field. They are
    a. Edit value – Users click on this button to edit the existing metadata value
    b. Delete metadata field – Click on this button to delete the metadata field from the Item
    c. Undo changes – Click on this button to undo changes made in the metadata field

Add a metadata field

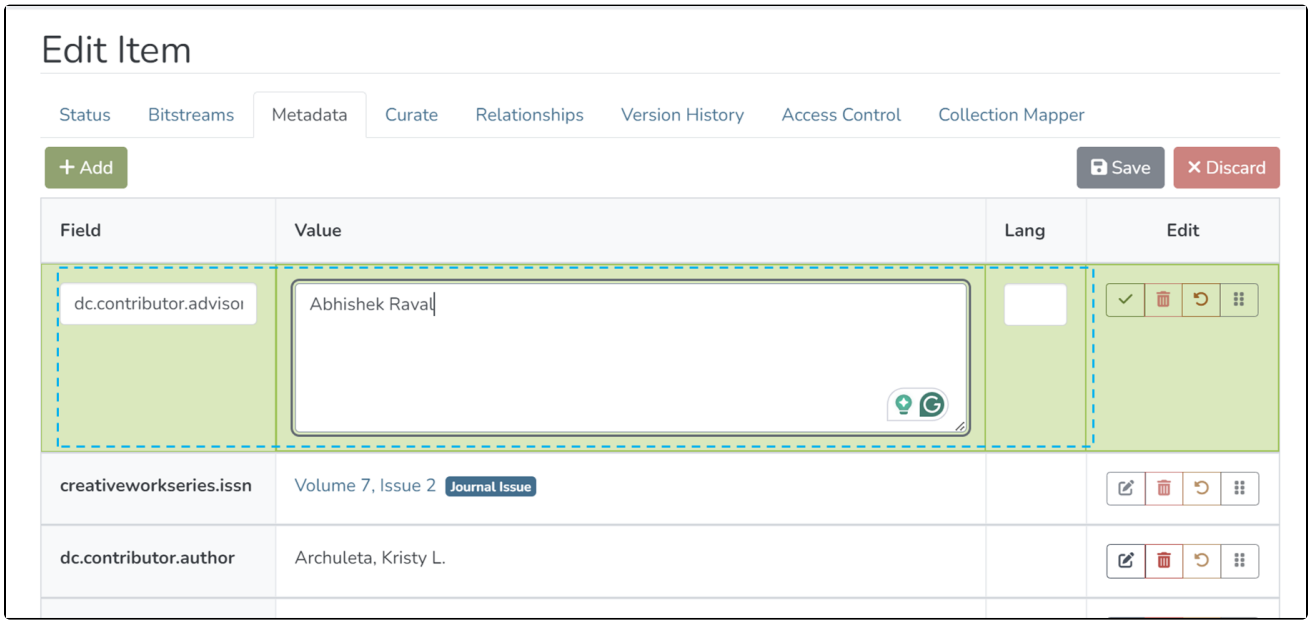
Step 1: Click on the "Add" button to add a metadata field.



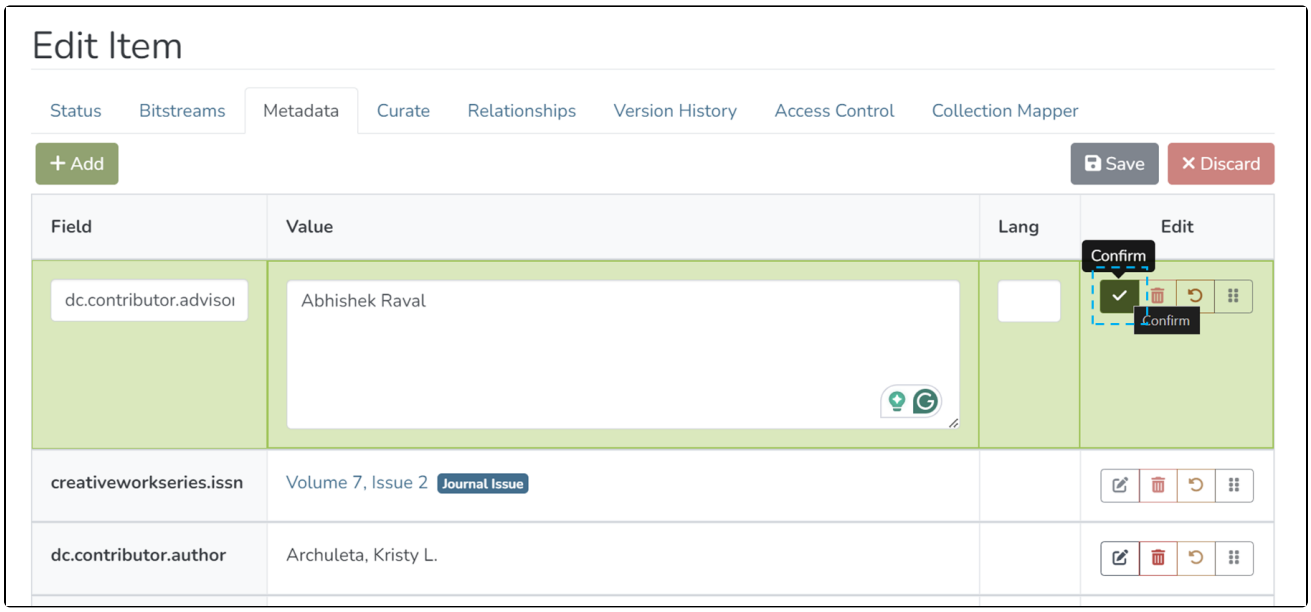
Step 2: Upon typing a few characters of the metadata element, users will notice a drop-down list showing metadata elements matching the entered value. Users can select the appropriate metadata element from the drop-down list.



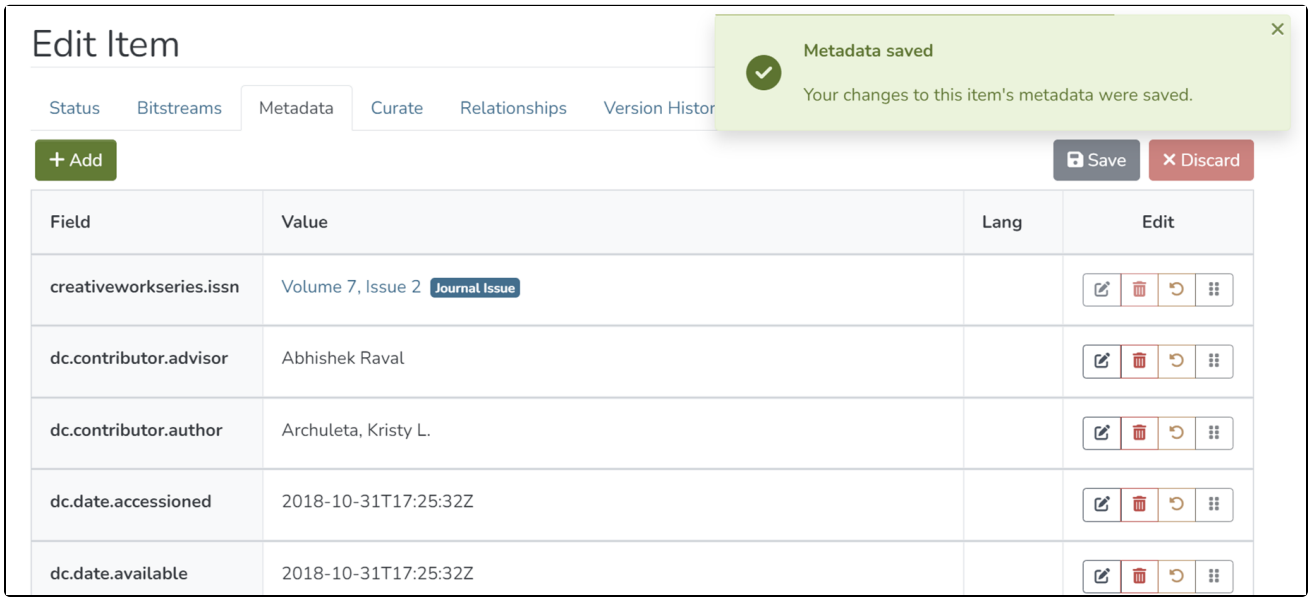
Step 3: After selecting the required metadata element, enter the metadata value in the “Value” field and the ISO code of the language under the Lang field. For example, enter "en" for English.



Step 4: Click on the “Complete” button as highlighted below to update the field.

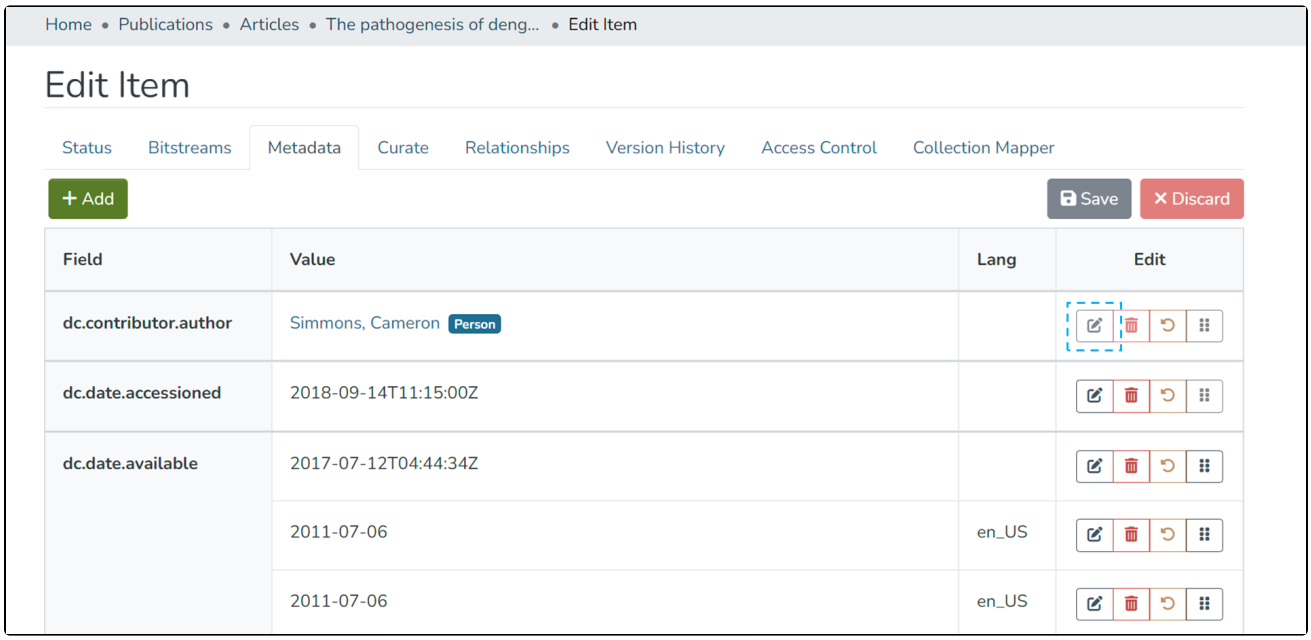


Step 5: Click on the “Save” button to continue saving changes or “Discard” to cancel changes made in the metadata fields. A success prompt confirming Metadata updates will appear as shown below.

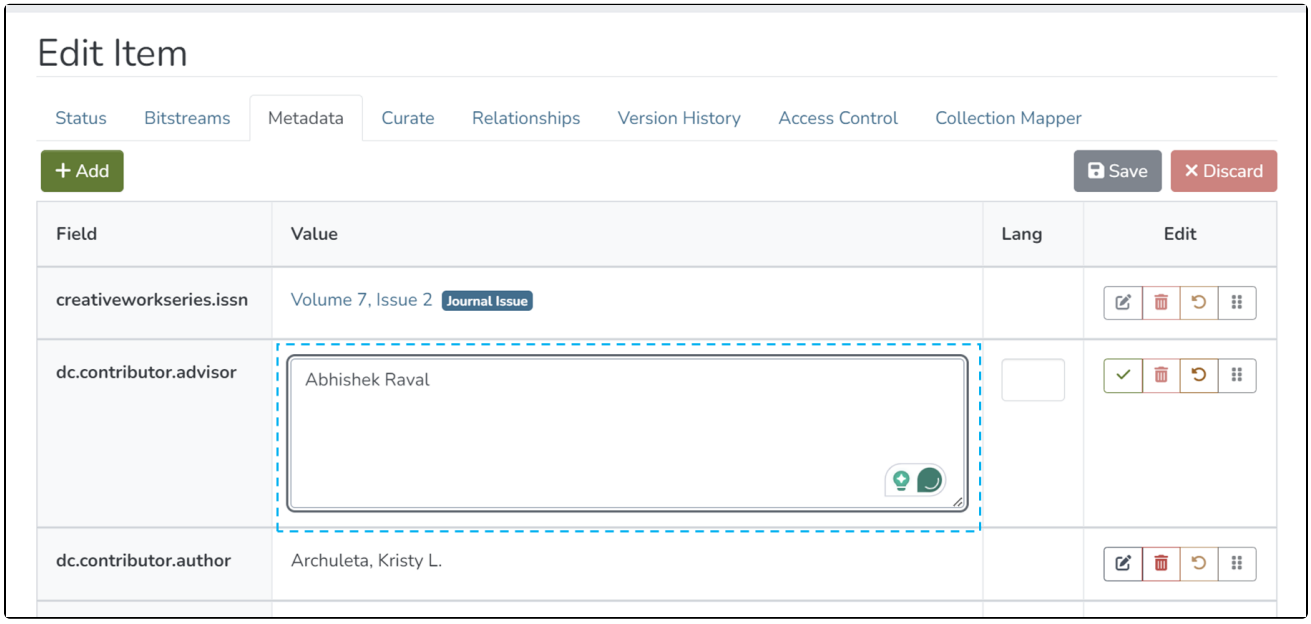


Edit an existing metadata field

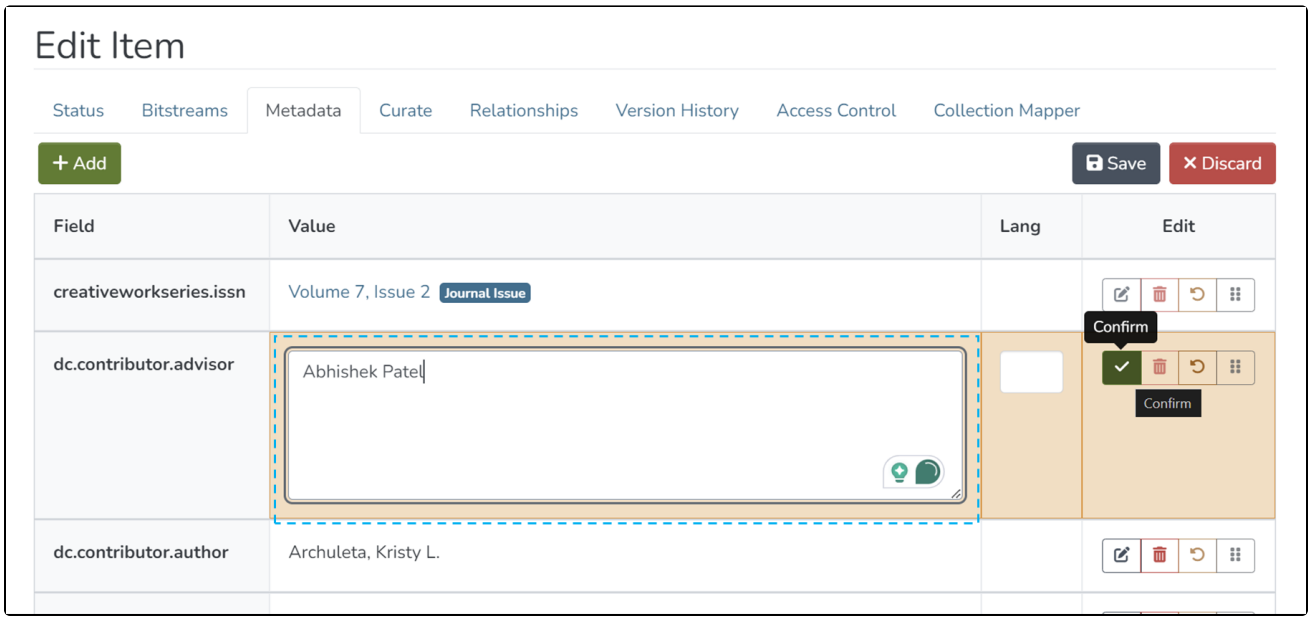
Step 1: Click on the “Edit” button to edit an existing metadata field.



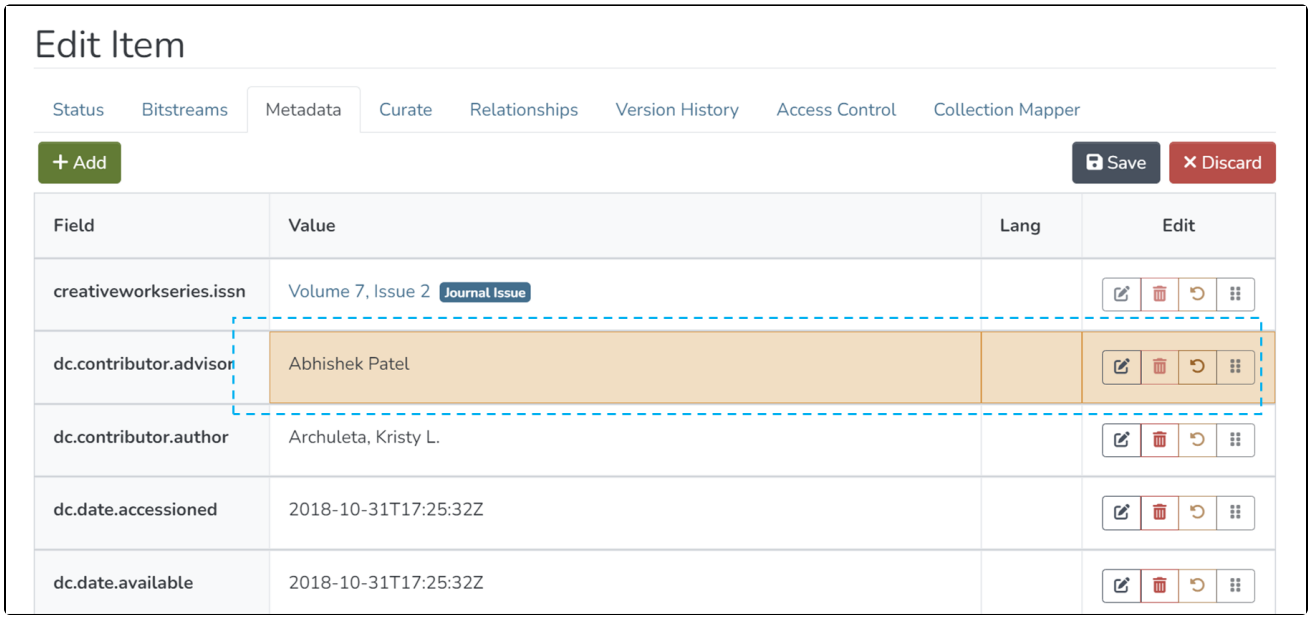
Step 2: The metadata field becomes editable after clicking the “Edit” button to edit the metadata field element, value, and language.



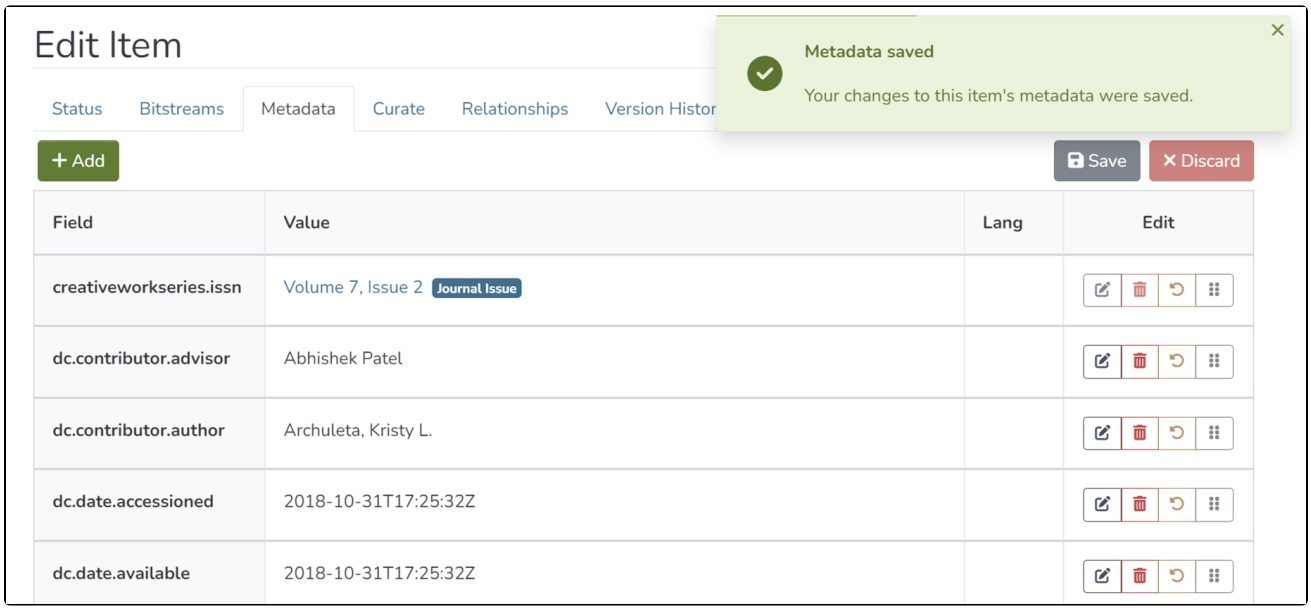
Step 3: Click on the “Complete” button as highlighted in the screenshot below to finish the update.



The metadata field will highlight the successful addition of the metadata, as shown in the screenshot below. Click on the undo button if you want to undo the addition of the metadata field.

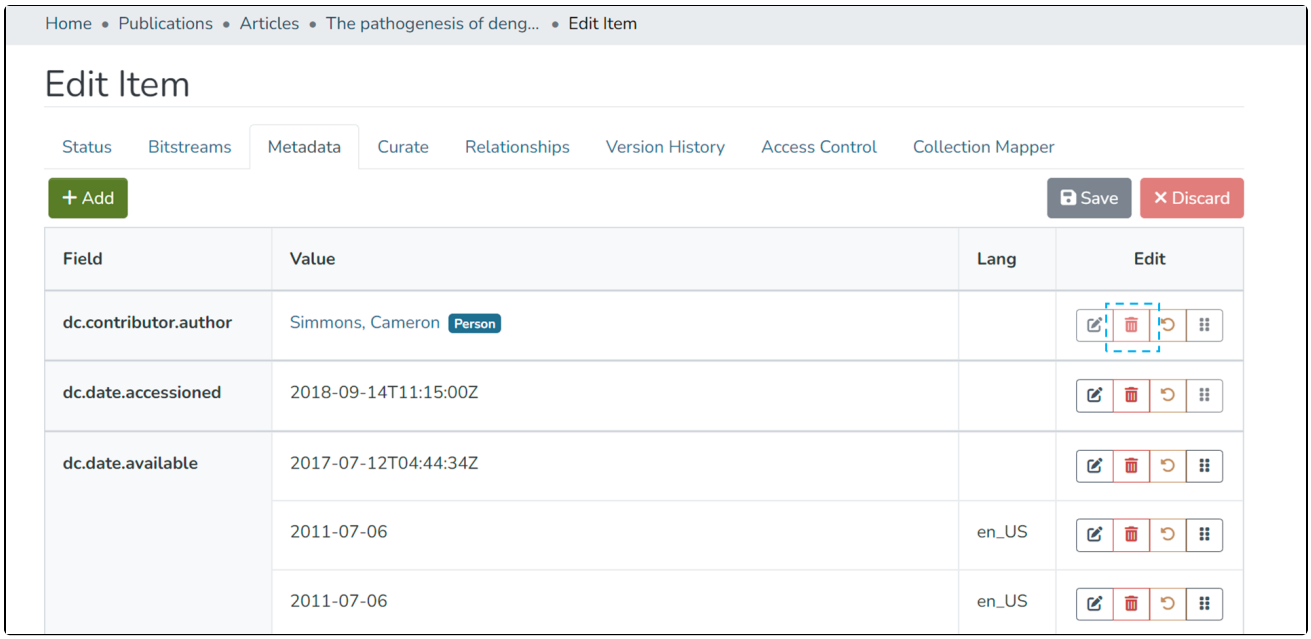


Step 4: Click on the “Save” button to continue saving changes or “Discard” to cancel changes made in the metadata fields. A success prompt confirming Metadata updates will appear as shown below.

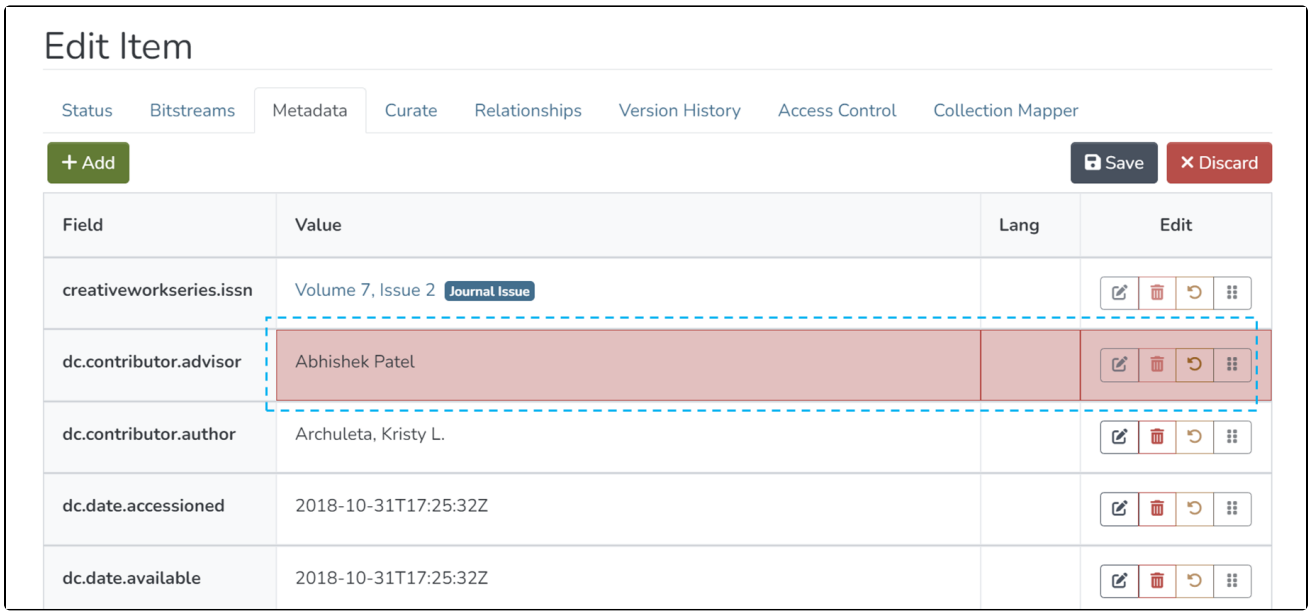


Delete an existing metadata field

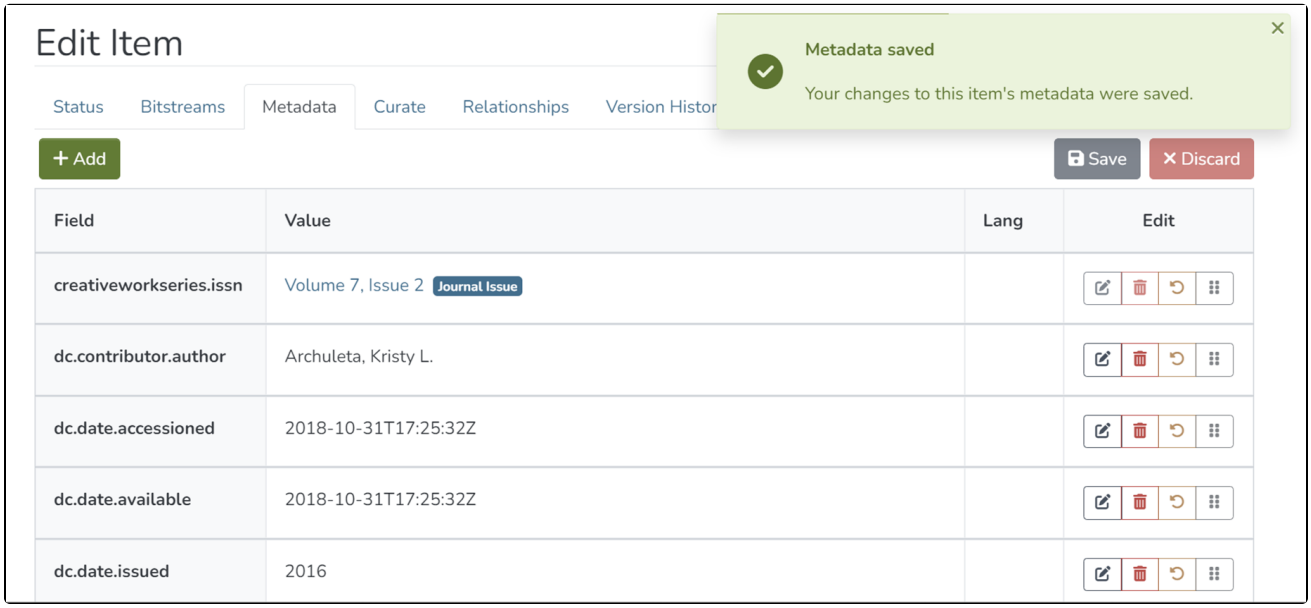
Step 1: Click on the “Delete” button to initiate the metadata field deletion.



Step 2: The deleted metadata field will be highlighted in red.



Step 3: Click on the “Save” button to continue saving changes or “Discard” to cancel changes made in the metadata fields. A success prompt confirming Metadata updates will appear as shown below.



Add or edit authority-controlled metadata fields

Some metadata fields may be configured to use controlled authority values. When adding or editing metadata in fields configured with such authorities, the metadata input field adjusts to enable the selection of these controlled values.

Three different presentations are offered for editing/selecting authority-controlled metadata:

- **Dropdown field**: for metadata fields controlled by a list of value-pairs defined in submission-form.xml.
- **Pop-up tree selector:** for metadata fields controlled by Hierarchical Taxonomies or Controlled Vocabularies.
- **Autocomplete field**: for authority-controlled metadata fields with authority keys.

A metadata field could be configured to use controlled authority values in two ways:

- Firstly, if the metadata field is globally configured to work with a ChoiceAuthority.
- Secondly, if the metadata is configured to use any of these controlled fields in the submission form defined for the owning collection of the item being edited.

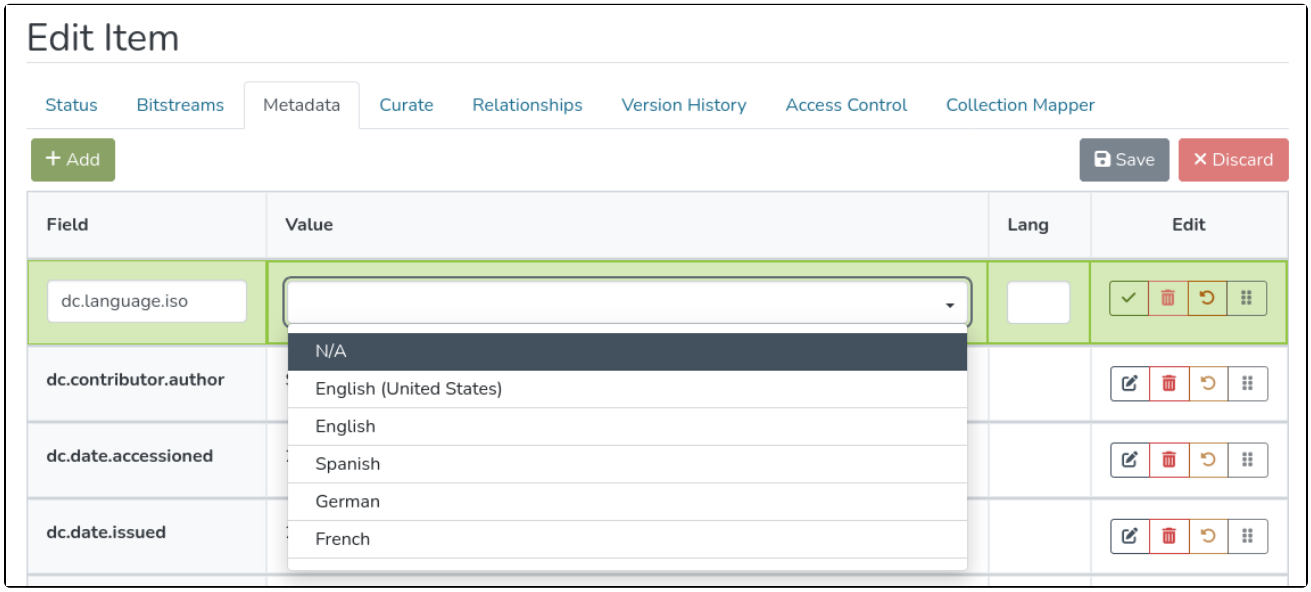
If both options are defined, the first option - the configuration defined for the ChoiceAuthority - will be used to obtain the controlled values.

See Authority Control of Metadata Values for more information.

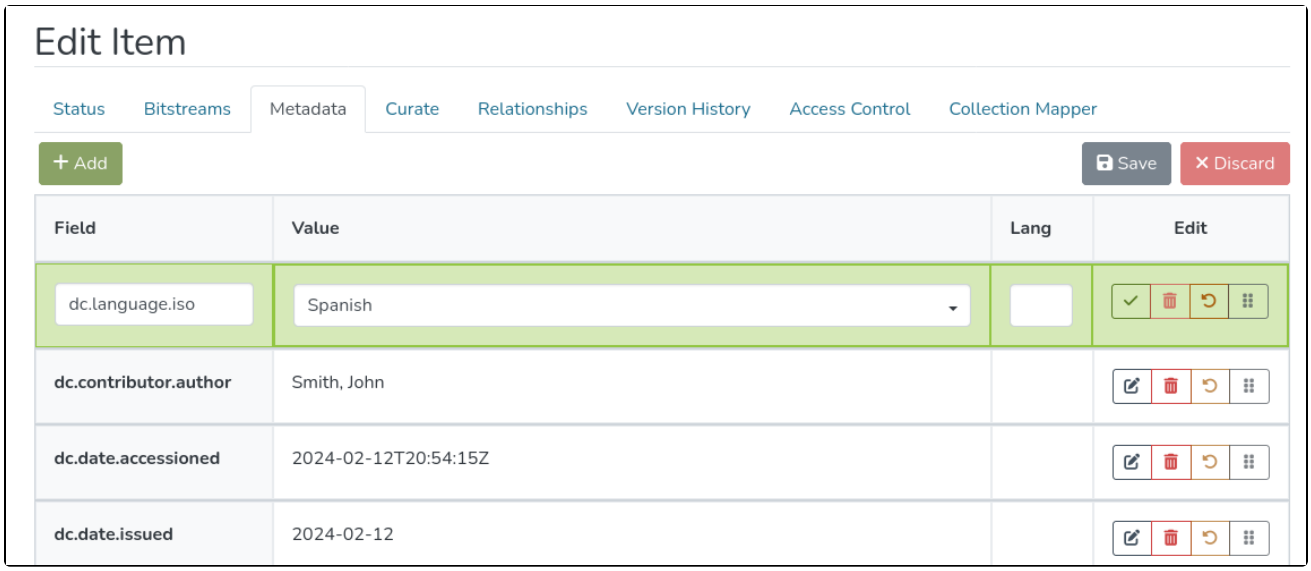
The following sections show how these fields are presented when adding new metadata. These fields will also be displayed similarly when editing an authority-controlled metadata field.

Metadata fields controlled by a list of value-pairs

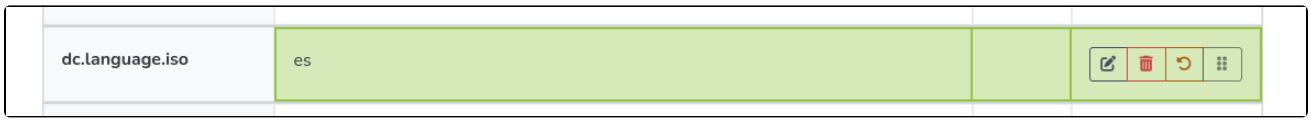
When metadata controlled by a list of value pairs is created, a dropdown is displayed to select the value:



After selecting a value and confirming it:



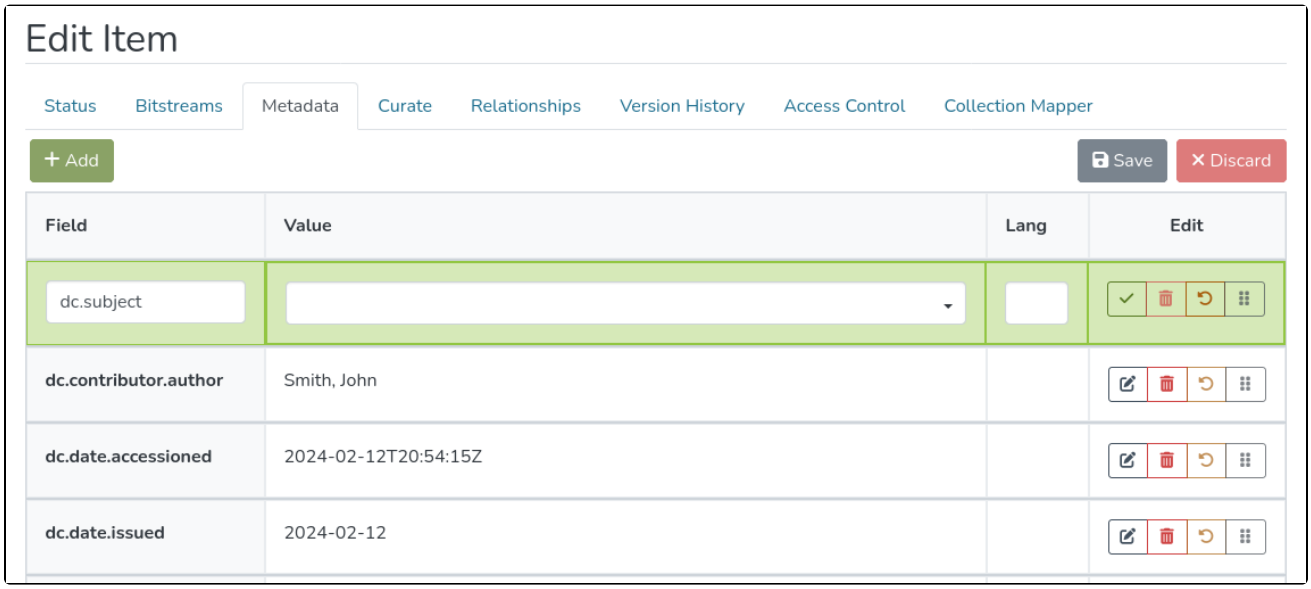
The new metadata is added. Note that once the change is confirmed, it will display the value that is internally stored, whereas during editing, it shows the displayed value configured in the list of pairs.



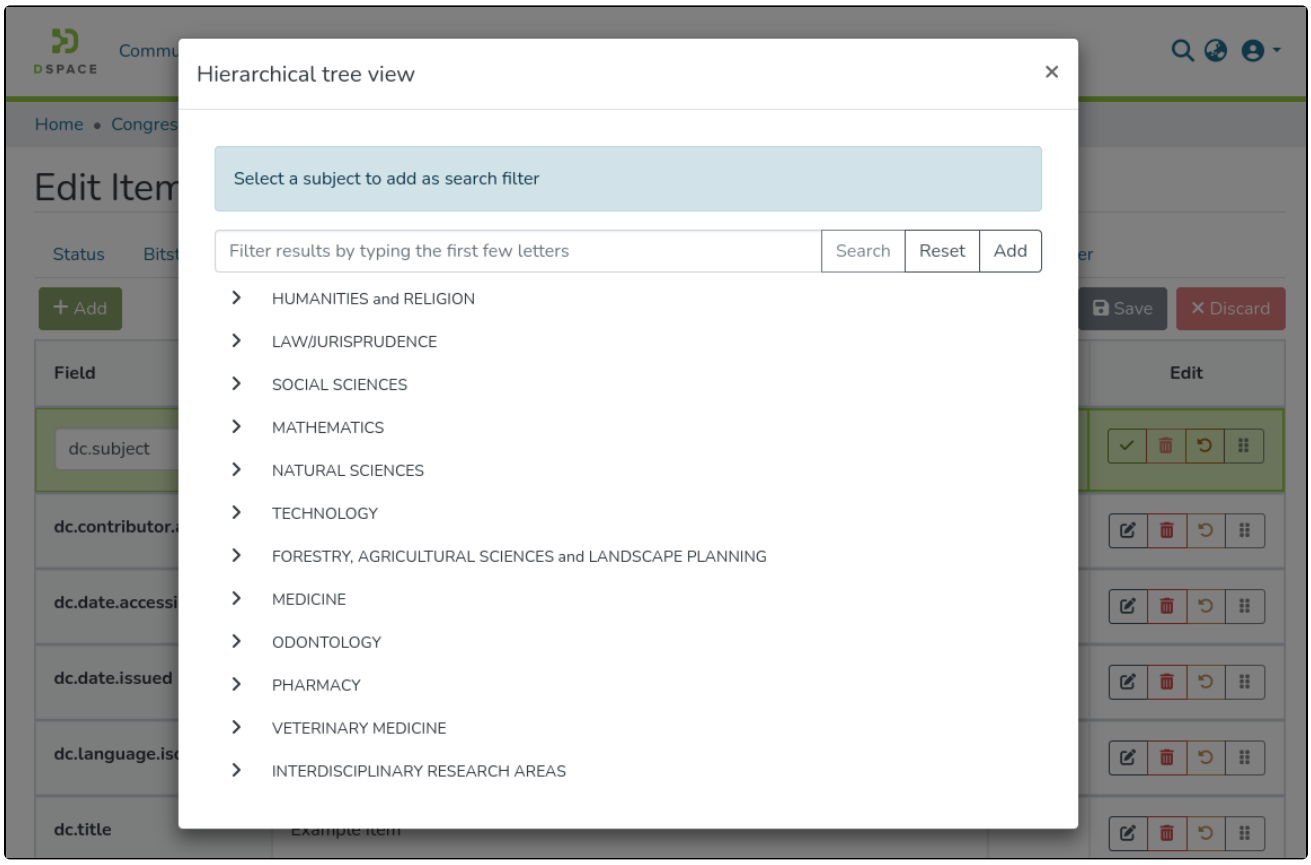
As with other fields, click on the “Save” button to continue saving changes or “Discard” to cancel changes made in the metadata fields.

Metadata fields controlled by Hierarchical Taxonomies or Controlled Vocabularies

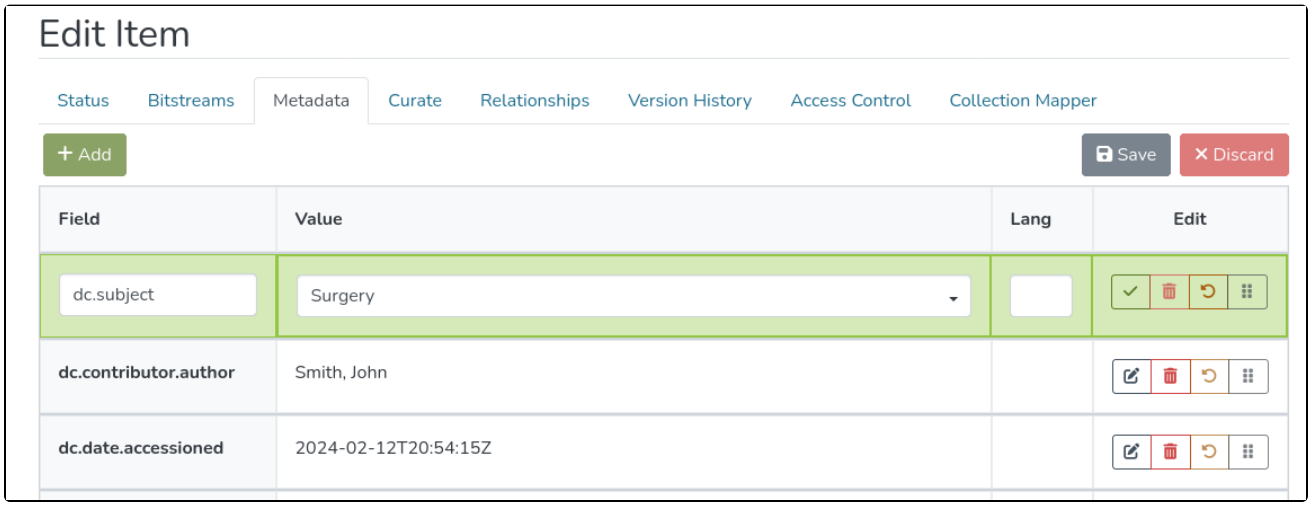
When a metadata controlled by a taxonomy list or controlled vocabulary is created, a field is displayed as follows:



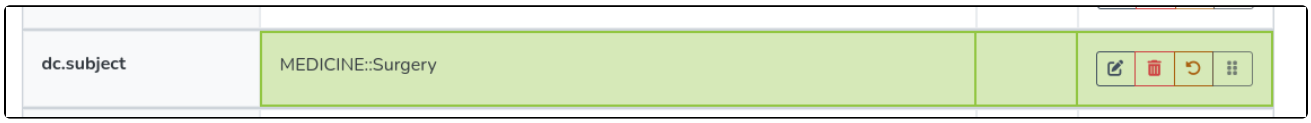
Clicking on the field opens a pop-up to select the controlled value:



Once selected, it will be displayed in the field:



After confirming the value, the field is displayed again with the value that is internally stored once the changes are saved.

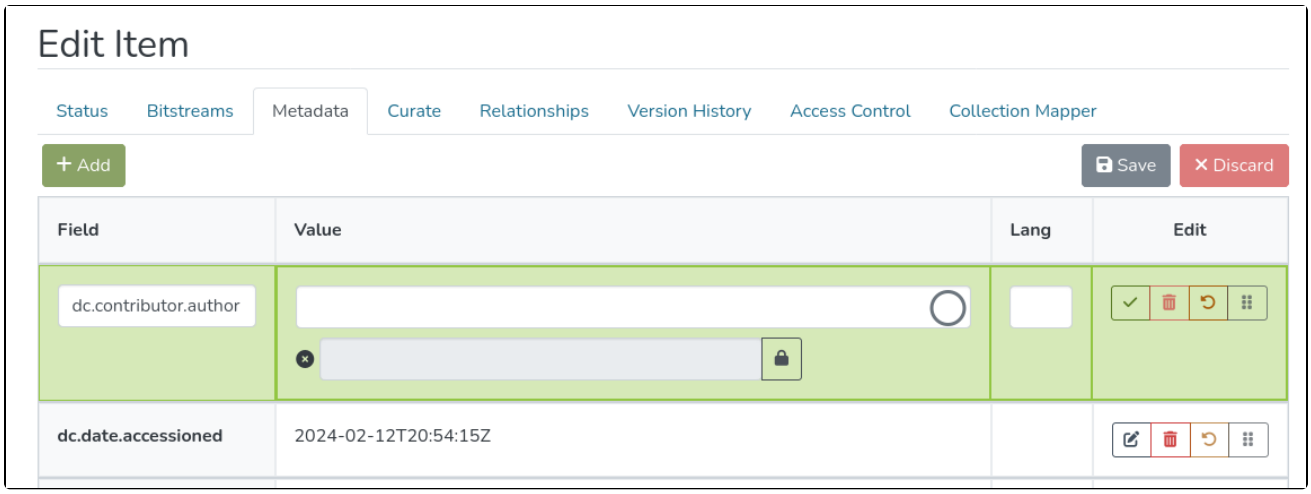


As with other fields, click on the “Save” button to continue saving changes or “Discard” to cancel changes made in the metadata fields.

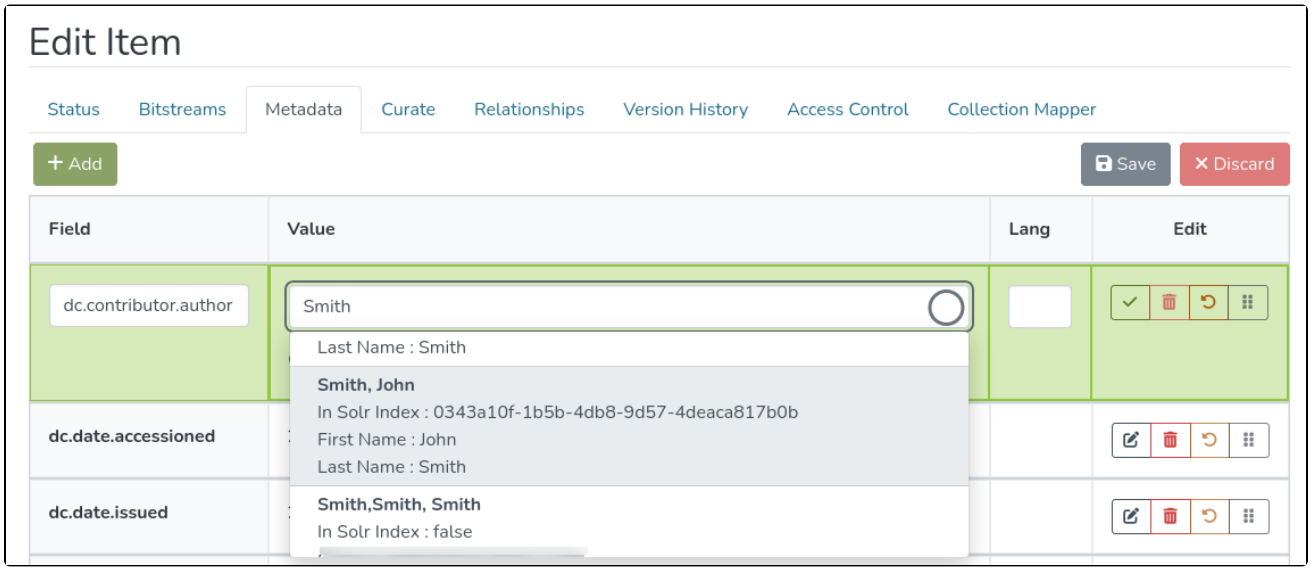
Authority-controlled metadata fields with authority keys

When creating metadata controlled by an authority with authority keys, a group of fields with the following elements will be displayed as follows:

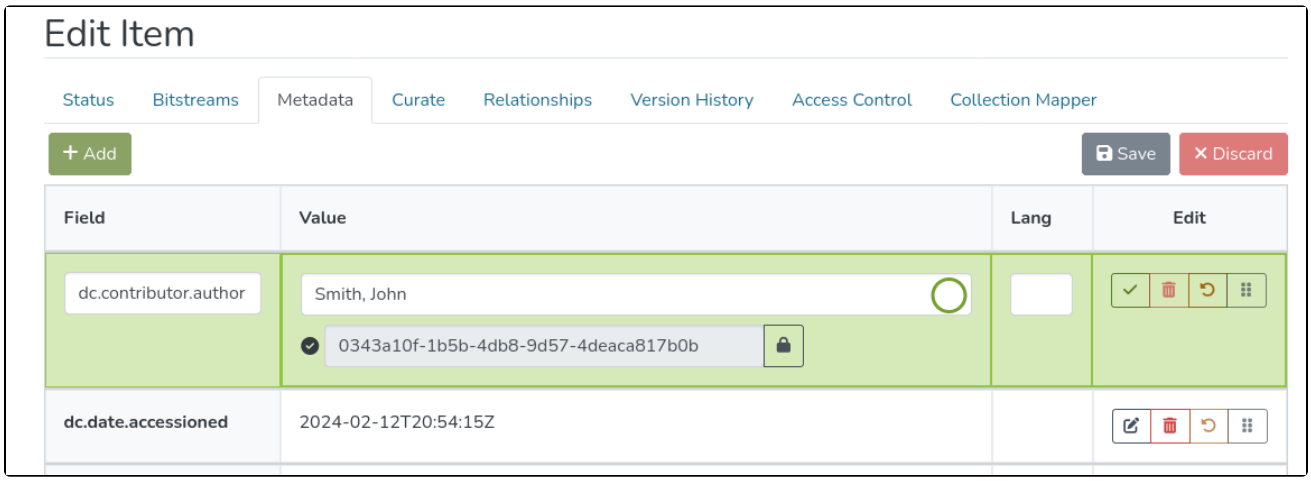
- A main input field with the metadata value
- A secondary field for the authority key
- An attached icon displays the associated confidence level with the key



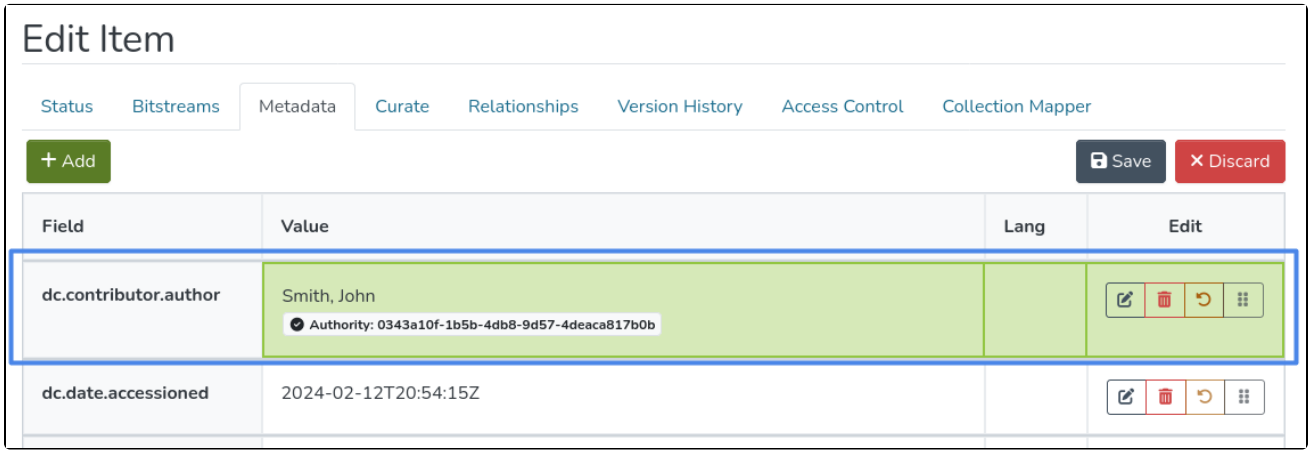
Entering text in the main field will suggest options to select controlled values as shown in the image:



When selecting one of the values, the selected value along with the associated key will be displayed in the corresponding fields, and the confidence value will display the icon to confirm that it is a value accepted by a user:



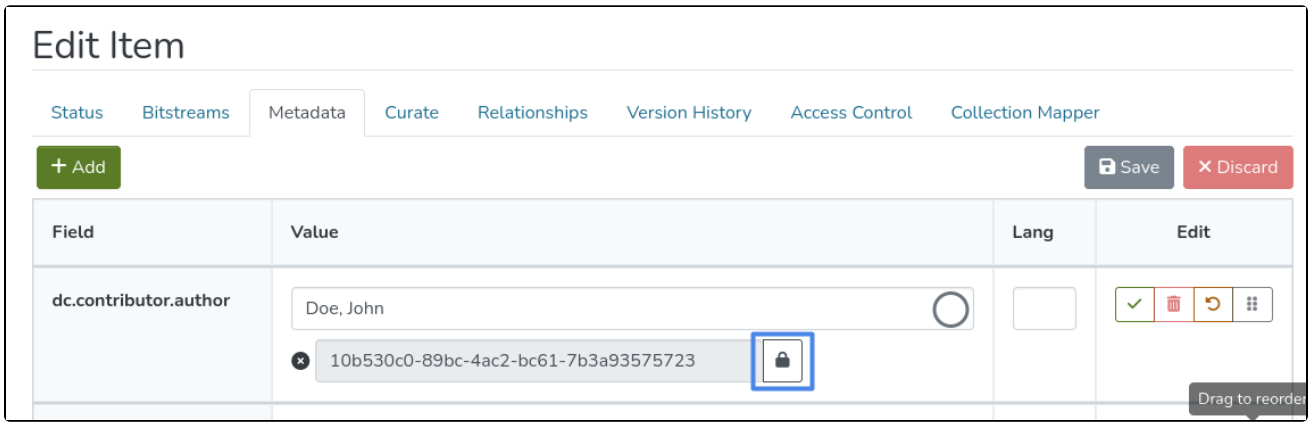
Once the change is confirmed, the associated authority key will be displayed along with the icon showing the confidence value:



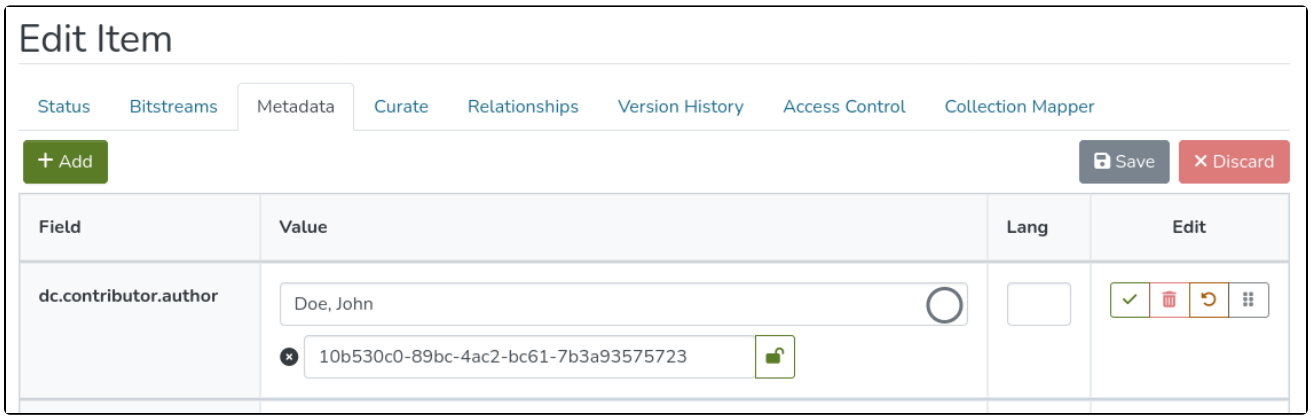
Fields that have an associated authority key with a minimum confidence value will be displayed in the metadata list with the corresponding authority and confidence icon.

Editing or removing an authority key

It is possible to edit the authority key directly. To do so, first unlock the field for editing by clicking the button with the lock icon:



Edit the field with the desired changes:



To remove the authority key, empty the field. The field will be displayed without the key before saving:



To change the authority key, edit the value. The key value will be modified, and the associated confidence value will be set as accepted by the user:



Confidence values

The following table shows the possible confidence values and the icons used by default:

Icon	Confidence	Confidence value (stored in metadatavalue)	Description	Displayed when not editing
	ACCEPTED	600	This authority value has been confirmed as accurate by an interactive user or authoritative policy	Yes
	UNCERTAIN	500	Authority value is singular and valid but has not been seen and accepted by a human, so its provenance is uncertain	Yes
	AMBIGUOUS	400	There are multiple matching authority values of equal validity	Yes
	NOTFOUND	300	There are no matching answers from the authority	Yes

	FAILED	200	The authority encountered an internal failure in trying to match the value	Yes
	REJECTED	100	The authority recommends this submission be rejected	Yes
	NOVALUE	0	No reasonable confidence value is available	No
	UNSET	-1	No confidence value has been set (default value in the DB table)	No

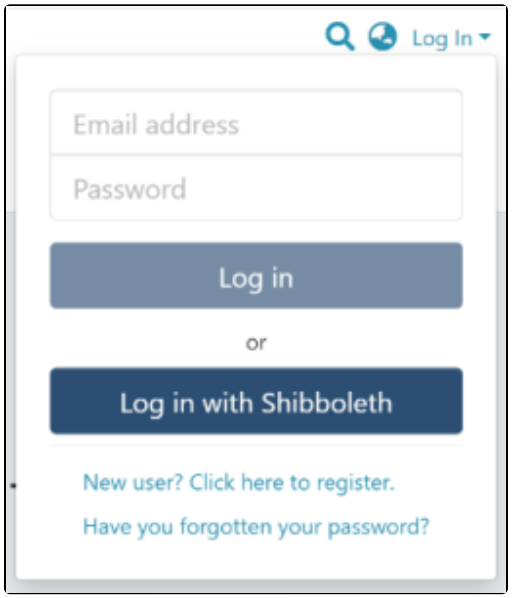
Edit Relationship

- Relationship Management
- Add Relationships with other items
- Delete a Relationship

Relationship Management

Entity relationships help authorized users to link two or more items by defining relationships among them. A few good examples are the Relationship between an article & Author, a Journal and Journal Article, an Organization Unit and Individuals in the unit, etc.

Step 1: Login using your credentials



Step 2: Go to the Item you want to edit

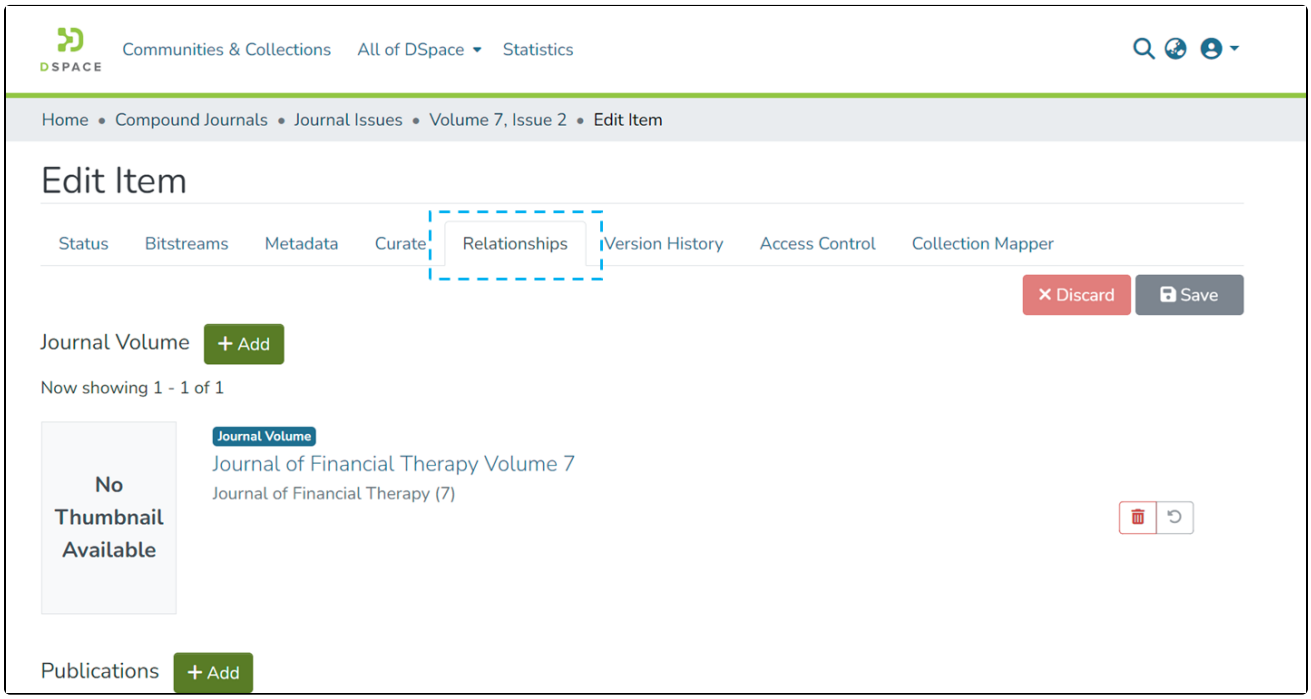
Users can reach an item through multiple methods, which are listed below

1. Finding an item using search functionalities of the DSpace
2. Reaching an item by browsing communities and collections
3. Finding an item in the Administration section at Edit > Item

Click on the “Edit” button appearing towards the right-hand side of the item title.

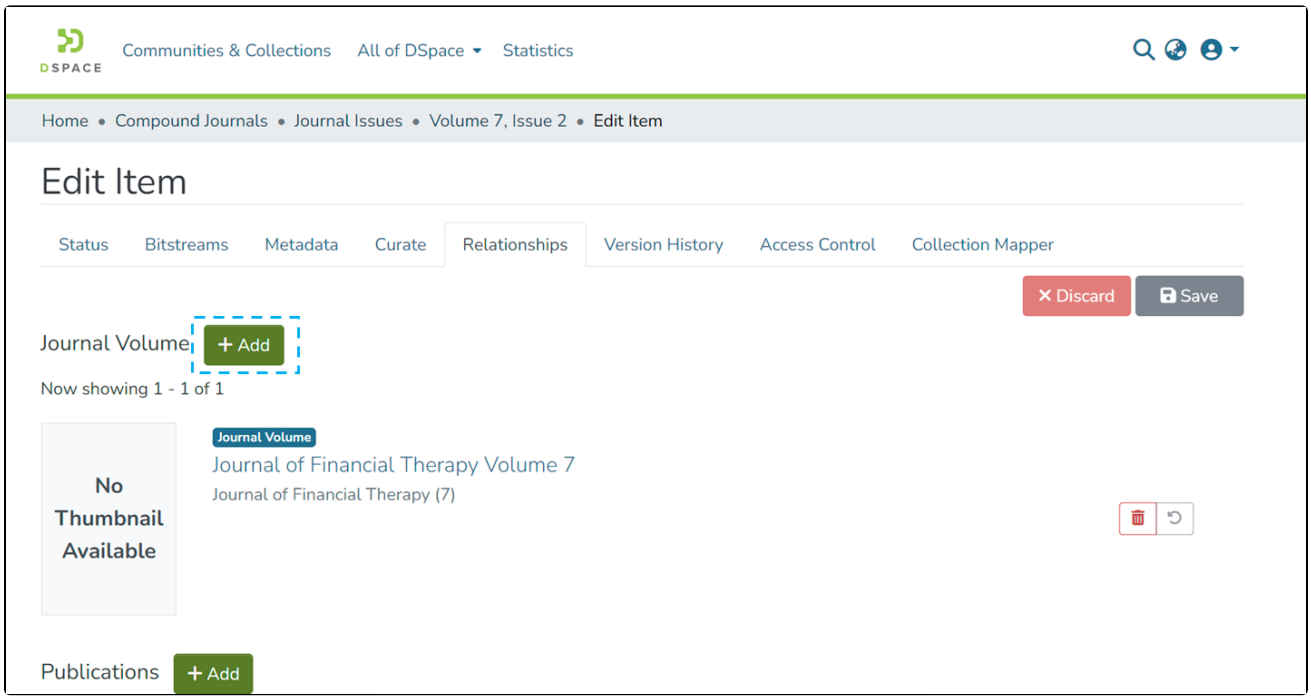


Step 3: Click on the “Relationships” tab.

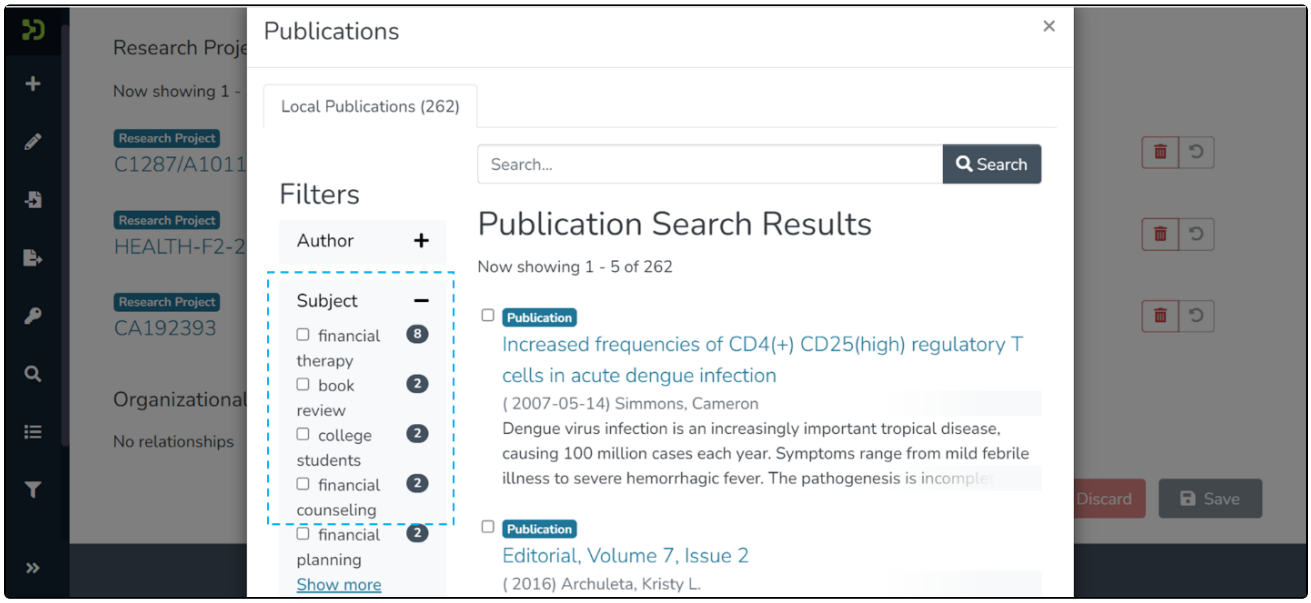


Add Relationships with other items

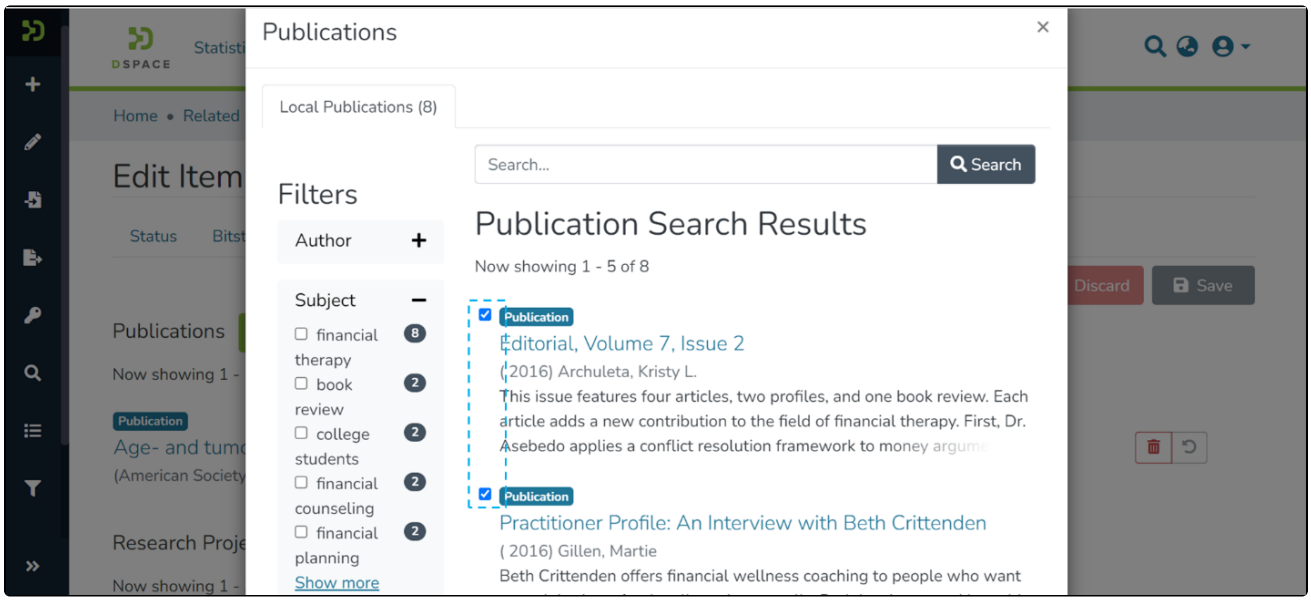
Step 1: Click on the “Add” button next to the relationship type to map Item (s) under each relationship type. For example, click on the “Add” button next to Publications to add an item as a related publication.



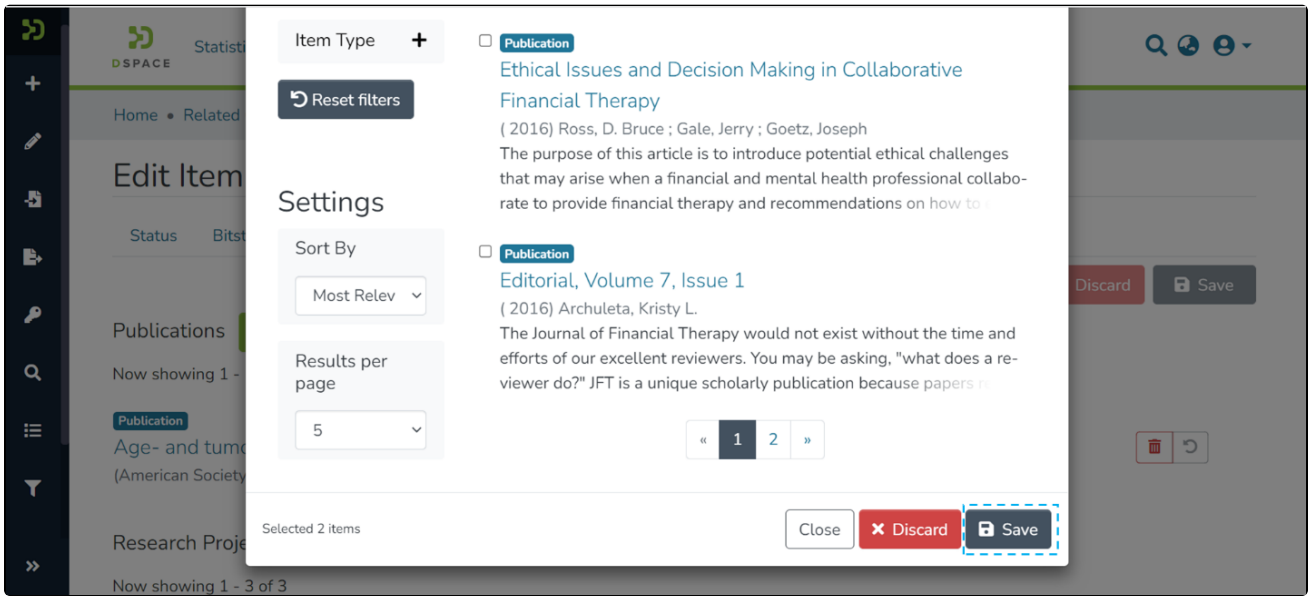
Step 2: Identify target items for addition using Filters and Search functions. Please note that the process is identical to the advanced search process.



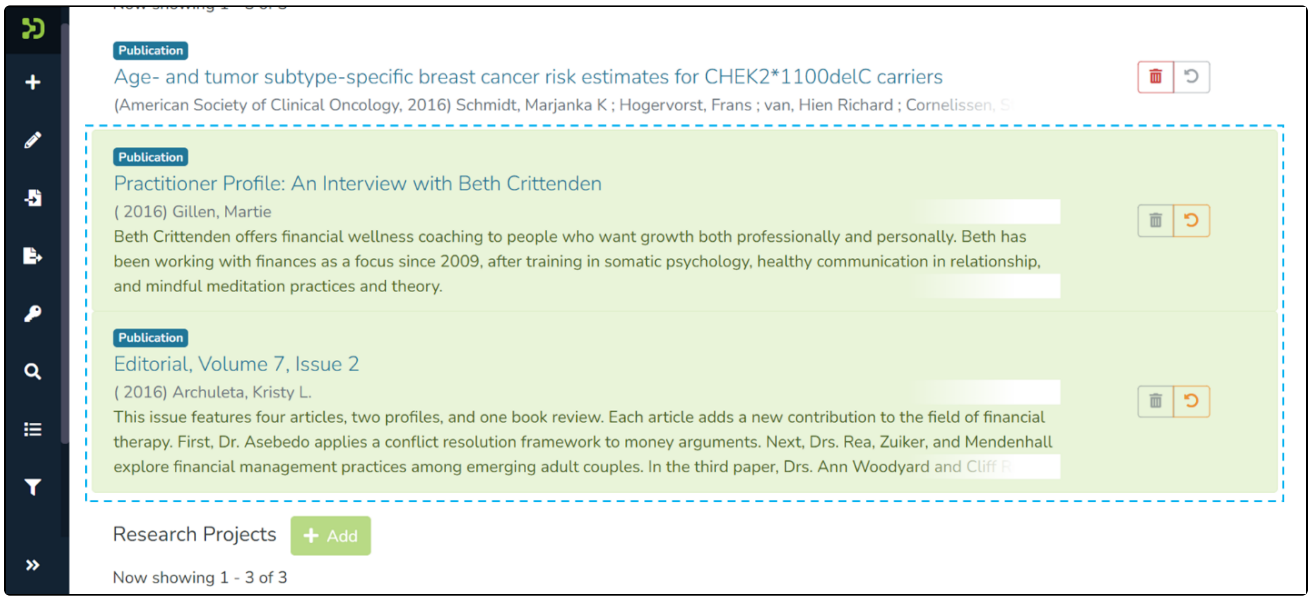
Step 3: Click checkboxes appearing on the left-hand side of each Item to select them for the Relationship.



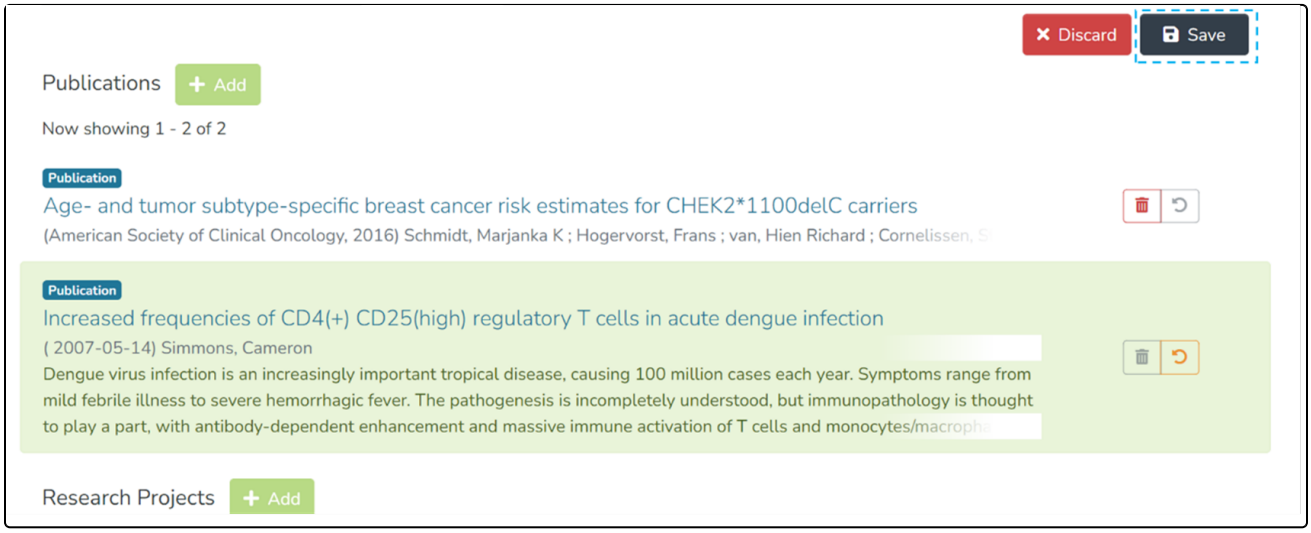
Step 4: Scroll down to the bottom of the screen and click on the “Save” button to complete the process.



The application will redirect users to the source item’s relationship tab, highlighting newly added items for easy identification.



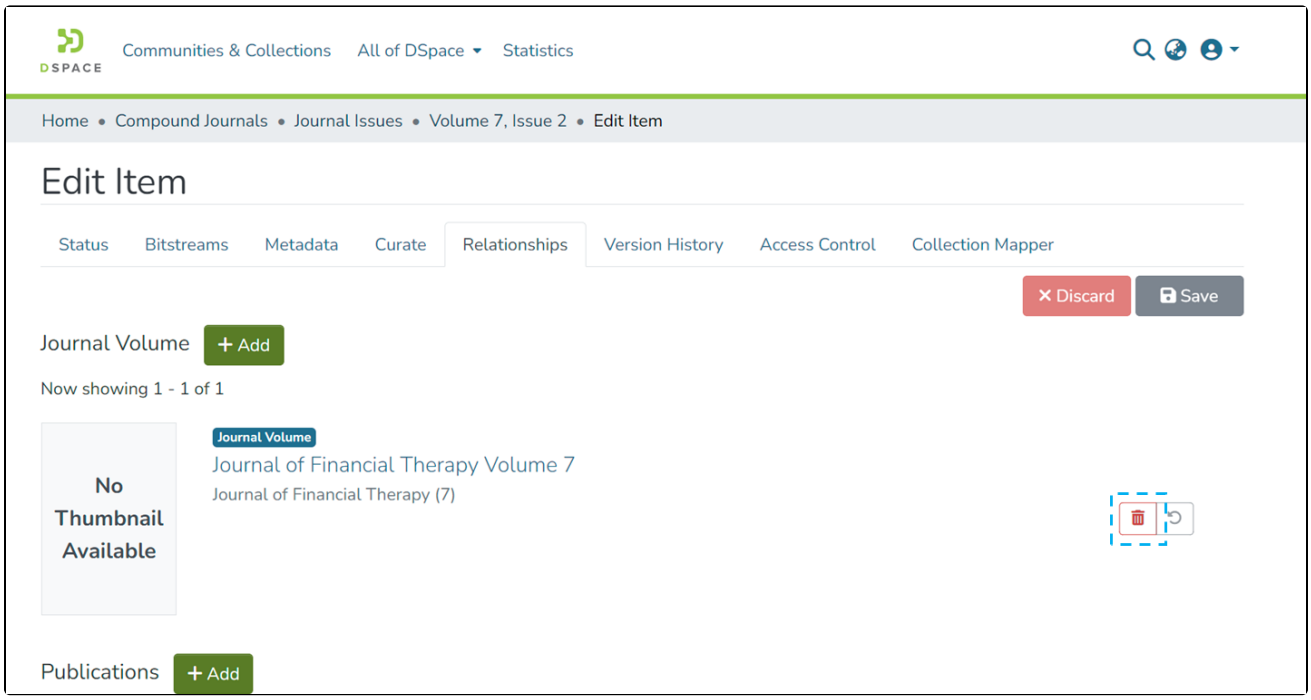
Step 5: Click on the “Save” button to confirm the item addition under the selected relationship type or click “Discard” to undo the addition.



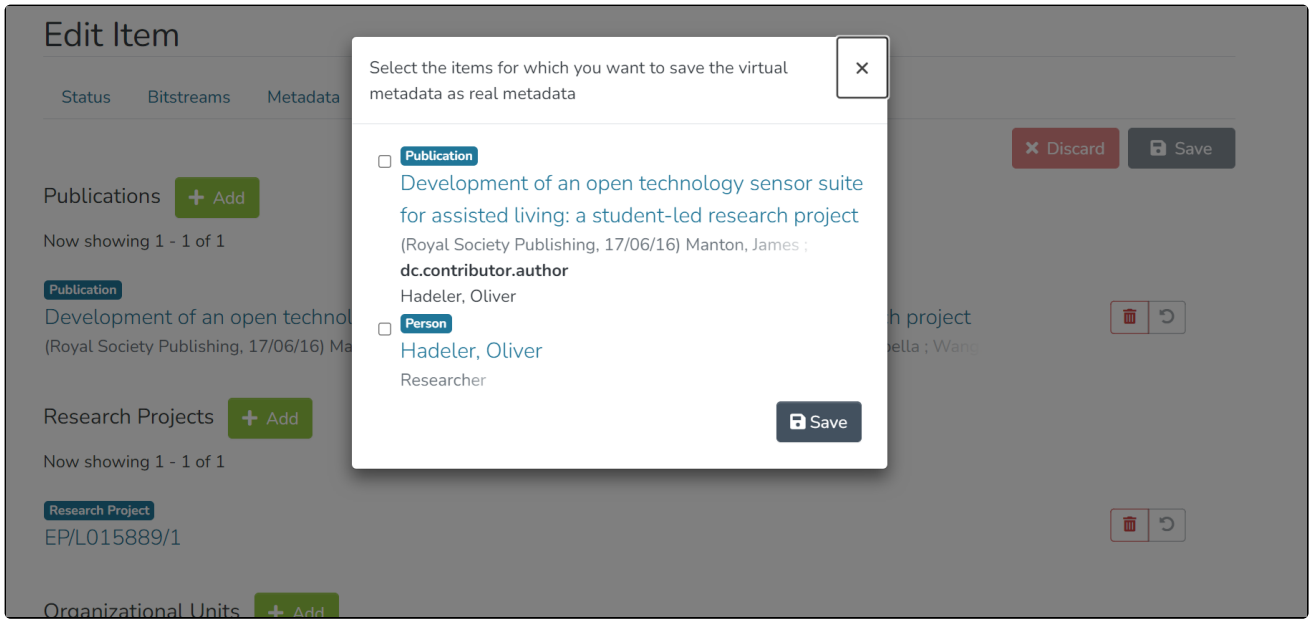
Users will see a prompt confirming the successful mapping of items under the selected relationship type.

Delete a Relationship

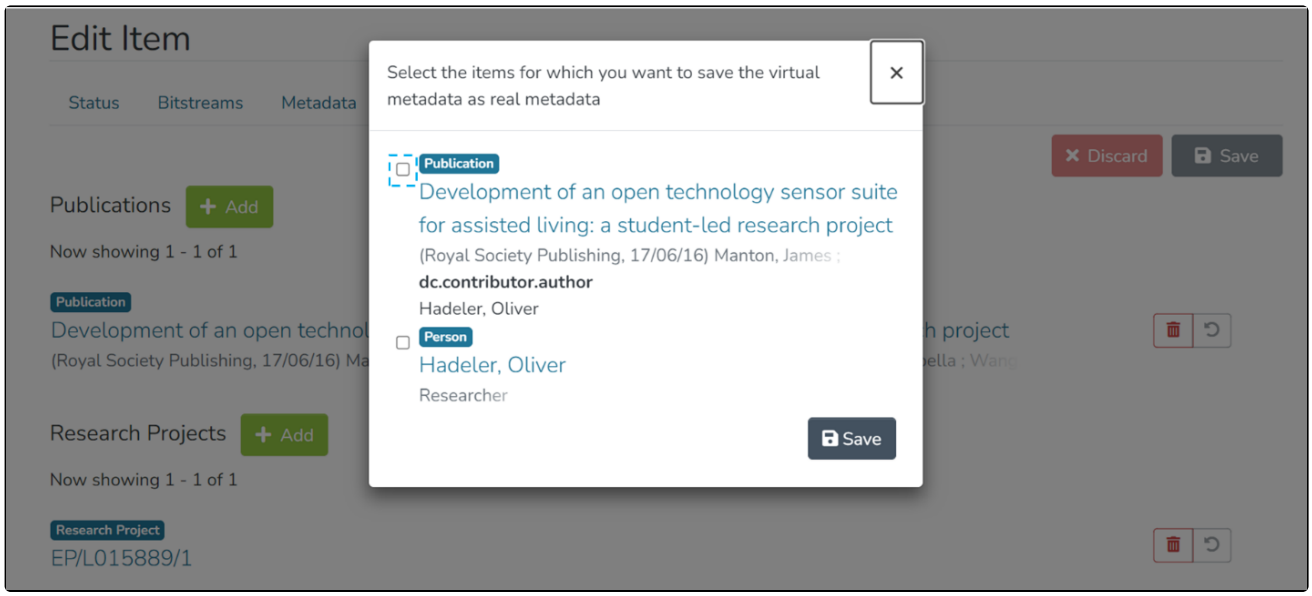
Step 1: Click on the “delete” button appearing a bitstream and drag it above or below another bitstream(s) in the bundle to change the bitstream’s sequence.



Step 2: A prompt showing items concerning the selected Item will appear in the prompt. Click the checkbox appearing next to items. Please refer screenshots below for a better understanding.

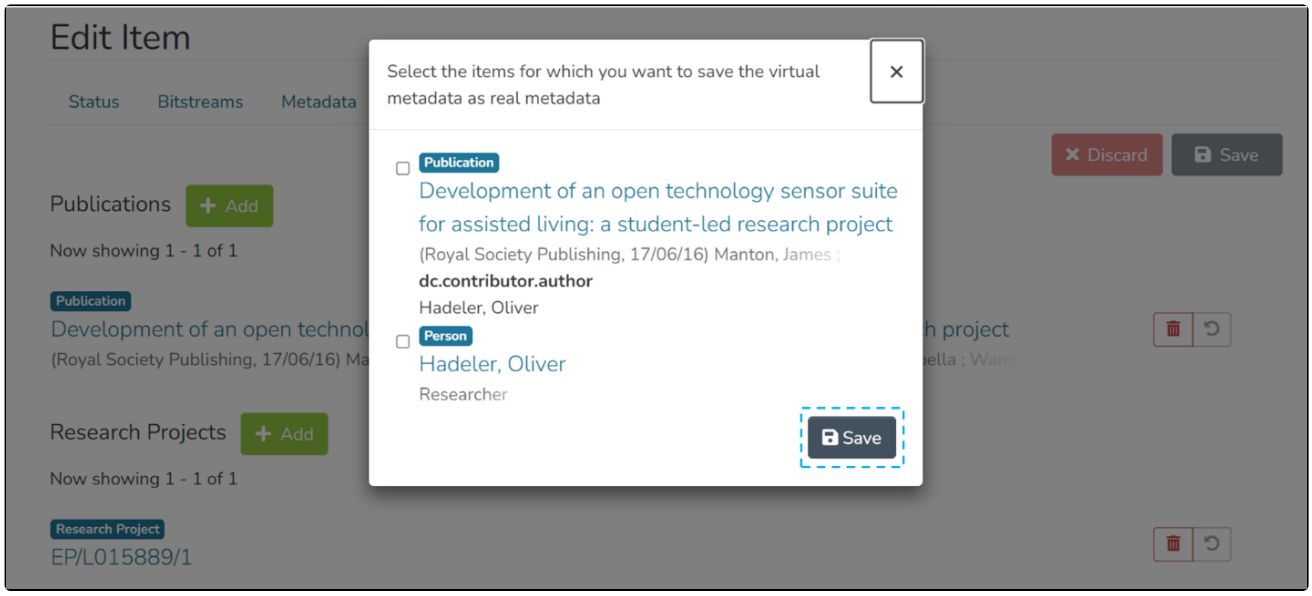


List of items concerning the selected Item.



Click on the checkbox highlighted before the Item

Step 3: Click the “Save” button to complete the selection process and go back to the Relationship tab of the source item.



Step 4: Click on the “Save” button to confirm the deletion under the selected relationship type or click “Discard” to undo the action. Users will see a prompt confirming the successful deletion of items under the selected relationship type.

Make an item discoverable

- Scope and Use Case
- Audience
- Make an item discoverable

Scope and Use Case

By default, all items are *discoverable*, meaning they will appear in search/browse/OAI results. If an item has been made non-discoverable (see Make an item non-discoverable), it can be made discoverable again by following the process below.

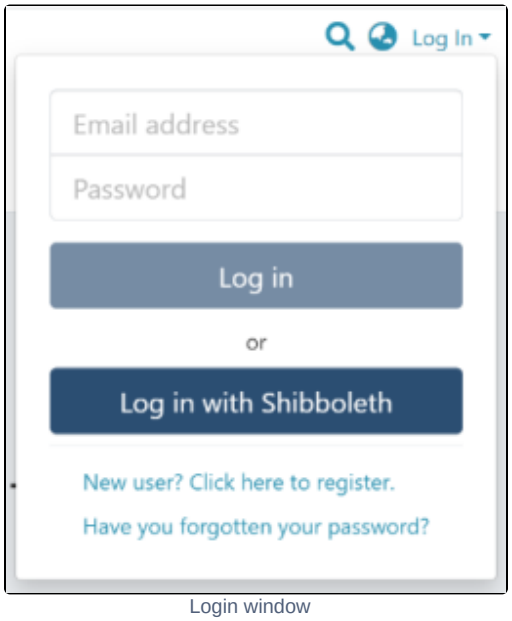
Once discoverable, it will appear in search, browse, and OAI results.

Audience

- Repository Administrators
- Community Administrators
- Collection Administrators

Make an item discoverable

Step 1: Login using your DSpace credentials.



Login window

Step 2: Go to the item you want to edit.

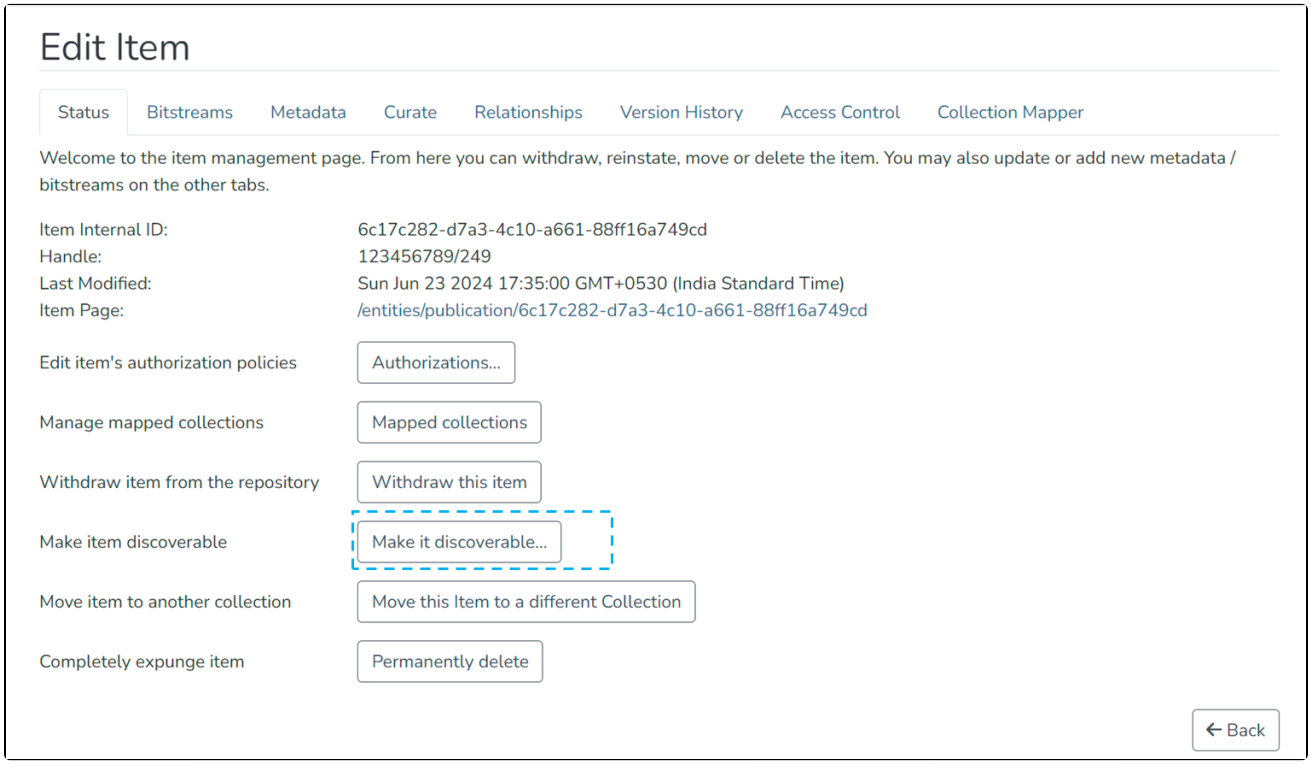
Users can reach an item through multiple methods, including those listed below:

1. Search for an item
2. Browse communities and collections
3. Find an item in the Administration section at Edit > Item
4. Use the existing item URL

Step 3: Click on the “Edit” button as highlighted on the screen below. This button will appear to any user who has edit rights on the target item.



Step 4: Click on the “Make it discoverable...” button under the “Status” tab.



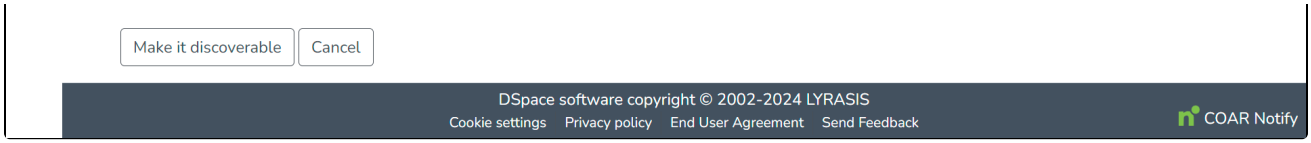
Edit Item page with the Make it discoverable button highlighted

Step 5: Click on the “Make it discoverable” button to make the the item discoverable or click the “Cancel” button to cancel the operation.

Make item discoverable: 123456789/249

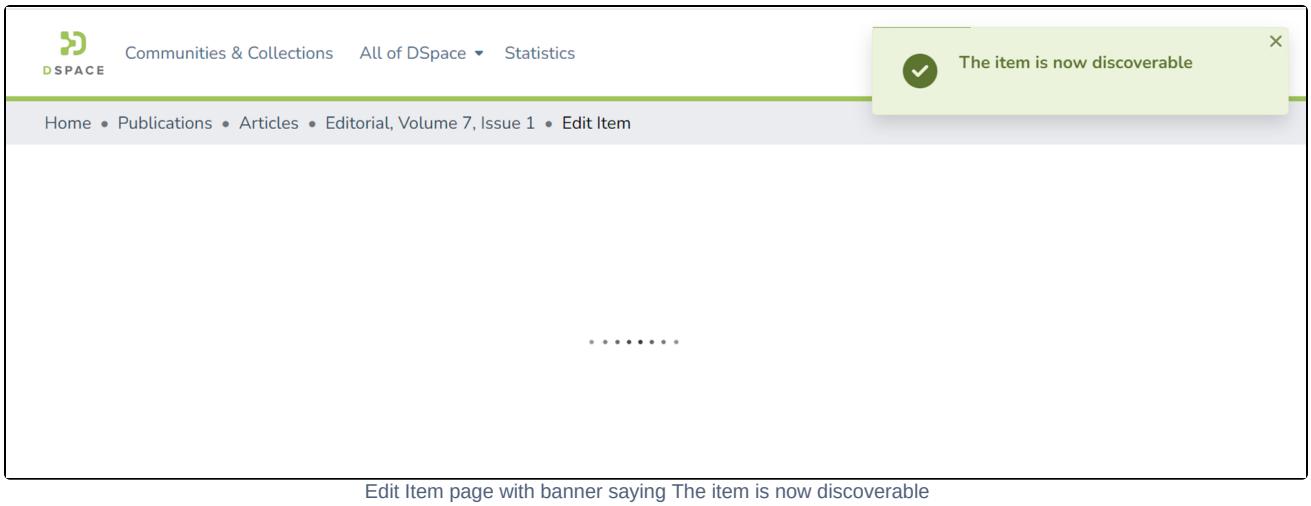
Are you sure this item should be made discoverable in the archive?

Field	Value	Language
creativeworkseries.issn	1945-7774	
dc.contributor.author	Archuleta, Kristy L.	
dc.contributor.author	Wood, James	
dc.date.accessioned	2018-10-26T16:11:52Z	
dc.date.available	2018-10-26T16:11:52Z	
dc.date.issued	2016	
dc.description.abstract	The Journal of Financial Therapy would not exist without the time and efforts of our excellent reviewers. You may be asking, "what does a reviewer do?" JFT is a unique scholarly publication because papers require the rigor of academic standards, but also must be translatable to non-researchers. It is not uncommon for researchers and practitioners to fail to communicate effectively with one another because the two groups speak what seems like different languages. Therefore, it is the goal of JFT to publish quality scholarly research and to emphasize the practicality of the research.	en_US
dc.description.provenance	Submitted by at mire (atmirenv@gmail.com) on 2018-10-26T16:11:52Z No. of bitstreams: 1 Editorial Vol. 7 Iss. 1.pdf: 788711 bytes, checksum: a74c74b893a3dabb51428aff831ac1d9 (MD5)	en
dc.description.provenance	Made available in DSpace on 2018-10-26T16:11:52Z (GMT). No. of bitstreams: 1 Editorial Vol. 7 Iss. 1.pdf: 788711 bytes, checksum: a74c74b893a3dabb51428aff831ac1d9 (MD5) Previous issue date: 2016	en
dc.identifier.citation	Archuleta, K. L. (2016). Editorial, Volume 7, Issue 1. Journal of Financial Therapy, 7 (1) 1. https://doi.org/10.4148/1944-9771.1114	en_US
dc.identifier.uri	https://demo7.dspace.org/handle/123456789/249	
dc.language.iso	en_US	en_US
dc.subject	editorial	en_US
dc.subject	financial therapy	en_US
dc.subject	Journal of Financial Therapy	en_US
dc.subject	peer review	en_US
dc.title	Editorial, Volume 7, Issue 1	en_US
dc.type	Article	en_US
dspace.entity.type	Publication	
journal.title	Journal of Financial Therapy	
journalvolume.identifier.name	Journal of Financial Therapy Volume 7	
relation.isAuthorOfPublication	0800c9b2-0aa6-46dd-8146-ec595f3c11bf	
relation.isAuthorOfPublication.latestForDiscovery	0800c9b2-0aa6-46dd-8146-ec595f3c11bf	
relation.isJournalIssueOfPublication	77877343-3f75-4c33-9492-6ed7c98ed84e	
relation.isJournalIssueOfPublication.latestForDiscovery	77877343-3f75-4c33-9492-6ed7c98ed84e	
relation.isJournalOfPublication	a23eae5a-7857-4ef9-8e52-989436ad2955	



Make item discoverable page

Step 6: You will see a success prompt confirming the item is discoverable, as shown below.



Edit Item page with banner saying The item is now discoverable

The item will now appear in search and browse results.

Edited and approved by DCAT Documentation Working Group, December 2024.

Make an item non-discoverable

- Scope and Use Case
- Audience
- Make an item non-discoverable

Scope and Use Case

A non-discoverable (or "private") item is one that is simply **hidden** from all search/browse/OAI results, and is therefore only accessible via direct link (or bookmark). By default, all items are *discoverable*, meaning they will appear in search/browse/OAI results.

Non-discoverable items *may or may not be access restricted.* It is possible for an item to be anonymously visible, but non-discoverable, so that you can only access the item if you are given a link to it.

Expected use cases:

- Provide a light rights awareness feature where discovery is not enabled for search and/or browse
- Hide “special items” such as repository presentations, guides or support materials
- Hide an old version of an item in cases where real versioning is not appropriate or liked

Non-discoverable items are **not** guaranteed to be hidden from search engines.

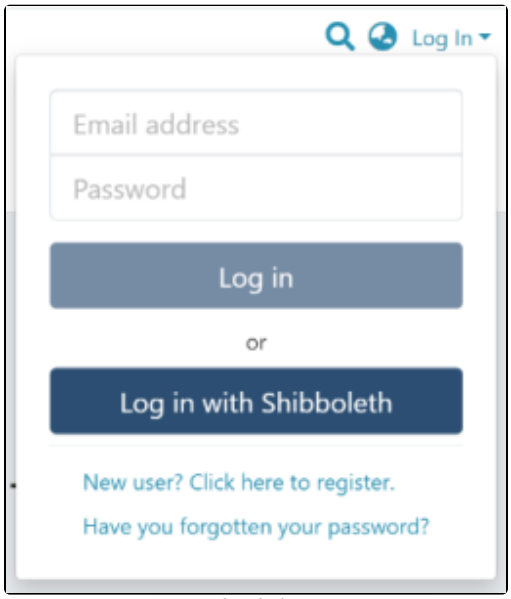
For more details about non-discoverable items, see DSpace Item State Definitions

Audience

- Repository Administrators
- Community Administrators
- Collection Administrators

Make an item non-discoverable

Step 1: Login using your DSpace credentials.



Login window

Step 2: Go to the item you want to edit.

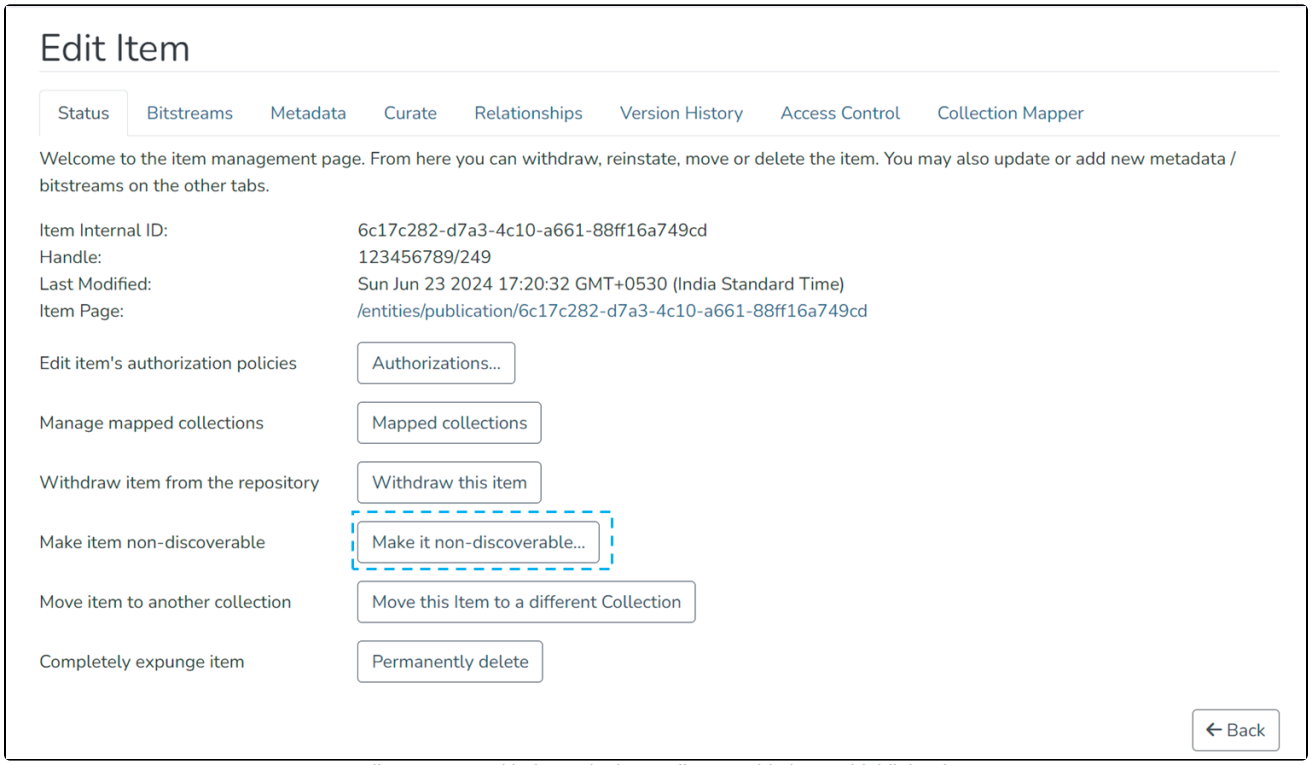
Users can reach an item through multiple methods, including those listed below:

1. Search for an item
2. Browse communities and collections
3. Find an item in the Administration section at Edit > Item
4. Use the existing item URL

Step 3: Click on the “Edit” button as highlighted on the screen below. This button will appear to any user who has edit rights on the target item.



Step 4: Click on the “Make it non-discoverable...” button under the “Status” tab.



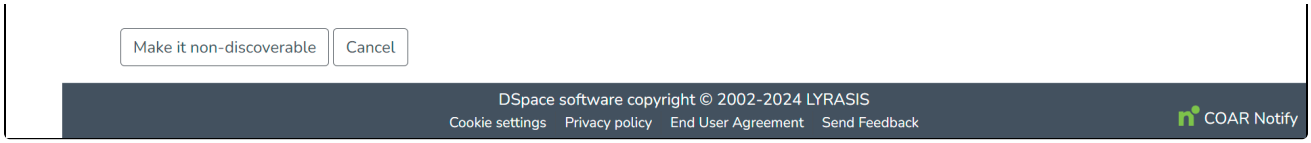
Edit Item page with the Make it non-discoverable button highlighted

Step 5: Click on the “Make it non-discoverable” button to make the item non-discoverable or click on the “Cancel” button to cancel the operation.

Make item non-discoverable: 123456789/249

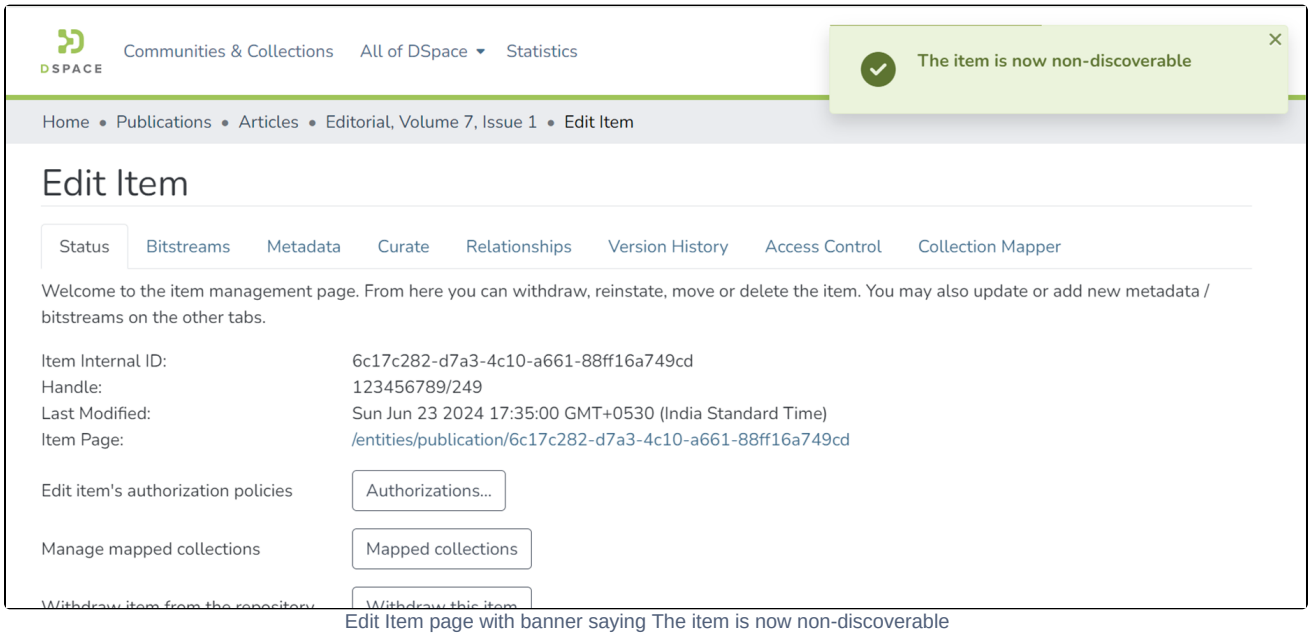
Are you sure this item should be made non-discoverable in the archive?

Field	Value	Language
creativeworkseries.issn	1945-7774	
dc.contributor.author	Archuleta, Kristy L.	
dc.contributor.author	Wood, James	
dc.date.accessioned	2018-10-26T16:11:52Z	
dc.date.available	2018-10-26T16:11:52Z	
dc.date.issued	2016	
dc.description.abstract	The Journal of Financial Therapy would not exist without the time and efforts of our excellent reviewers. You may be asking, "what does a reviewer do?" JFT is a unique scholarly publication because papers require the rigor of academic standards, but also must be translatable to non-researchers. It is not uncommon for researchers and practitioners to fail to communicate effectively with one another because the two groups speak what seems like different languages. Therefore, it is the goal of JFT to publish quality scholarly research and to emphasize the practicality of the research.	en_US
dc.description.provenance	Submitted by at mire (atmirenv@gmail.com) on 2018-10-26T16:11:52Z No. of bitstreams: 1 Editorial Vol. 7 Iss. 1.pdf: 788711 bytes, checksum: a74c74b893a3dabb51428aff831ac1d9 (MD5)	en
dc.description.provenance	Made available in DSpace on 2018-10-26T16:11:52Z (GMT). No. of bitstreams: 1 Editorial Vol. 7 Iss. 1.pdf: 788711 bytes, checksum: a74c74b893a3dabb51428aff831ac1d9 (MD5) Previous issue date: 2016	en
dc.identifier.citation	Archuleta, K. L. (2016). Editorial, Volume 7, Issue 1. Journal of Financial Therapy, 7 (1) 1. https://doi.org/10.4148/1944-9771.1114	en_US
dc.identifier.uri	https://demo7.dspace.org/handle/123456789/249	
dc.language.iso	en_US	en_US
dc.subject	editorial	en_US
dc.subject	financial therapy	en_US
dc.subject	Journal of Financial Therapy	en_US
dc.subject	peer review	en_US
dc.title	Editorial, Volume 7, Issue 1	en_US
dc.type	Article	en_US
dspace.entity.type	Publication	
journal.title	Journal of Financial Therapy	
journalvolume.identifier.name	Journal of Financial Therapy Volume 7	
relation.isAuthorOfPublication	0800c9b2-0aa6-46dd-8146-ec595f3c11bf	
relation.isAuthorOfPublication.latestForDiscovery	0800c9b2-0aa6-46dd-8146-ec595f3c11bf	
relation.isJournalIssueOfPublication	77877343-3f75-4c33-9492-6ed7c98ed84e	
relation.isJournalIssueOfPublication.latestForDiscovery	77877343-3f75-4c33-9492-6ed7c98ed84e	
relation.isJournalOfPublication	a23eae5a-7857-4ef9-8e52-989436ad2955	



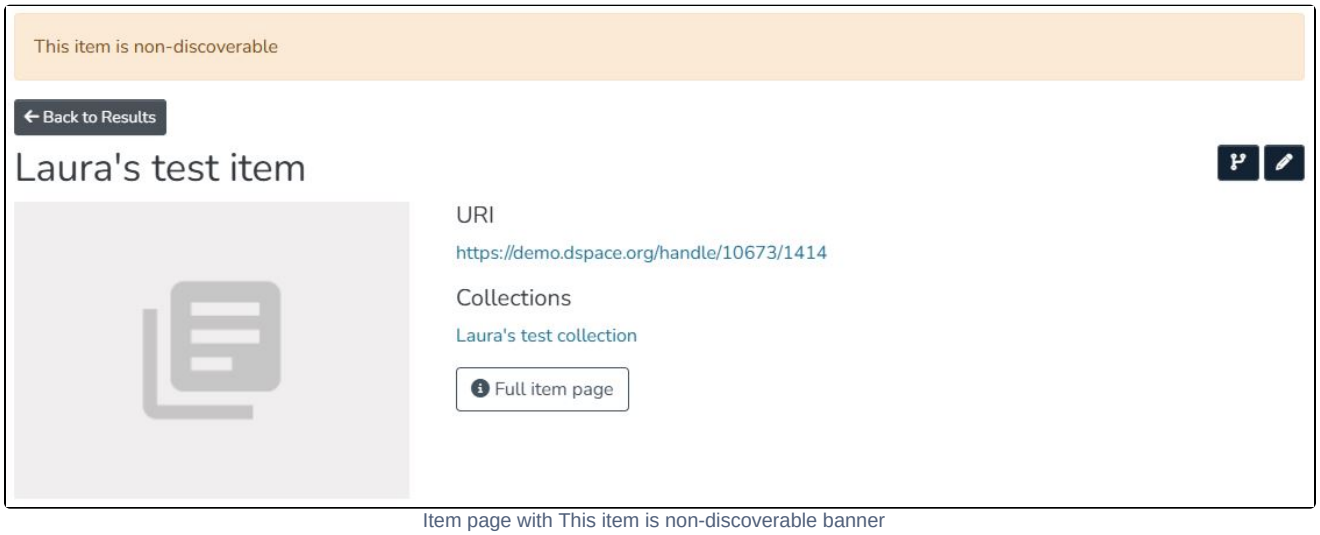
Make item non-discoverable page

Step 6: You will see a success prompt confirming that the item is non-discoverable, as shown below.



Edit Item page with banner saying The item is now non-discoverable

Step 7: The item page now shows a "This item is non-discoverable" banner, and the item does not appear in a list of search or browse results.



Item page with This item is non-discoverable banner

Edited and approved by DCAT Documentation Working Group, December 2024.

Move an Item

- Scope and Use Case
- Audience
- Move an Item

Scope and Use Case

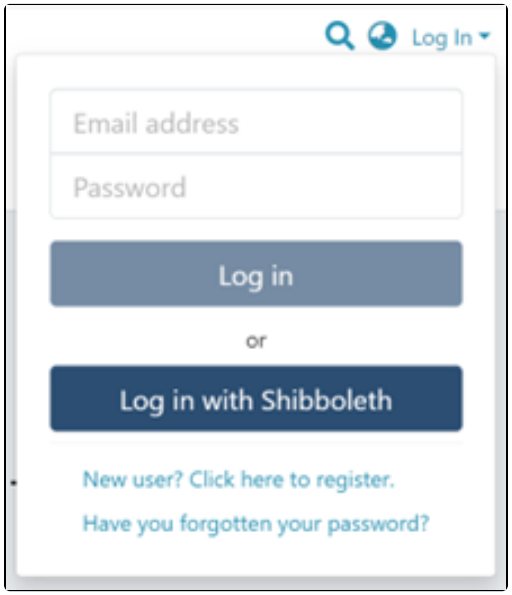
Items can be moved from one collection to another after submission.

Audience

- Repository Administrators
- Community Administrators
- Collection Administrators (to move an item, a user must be a collection administrator for both the collection the item is currently in and the collection it is being moved to)

Move an Item

Step 1: Login using your credentials



Login window

Step 2: Go to the item you want to move.

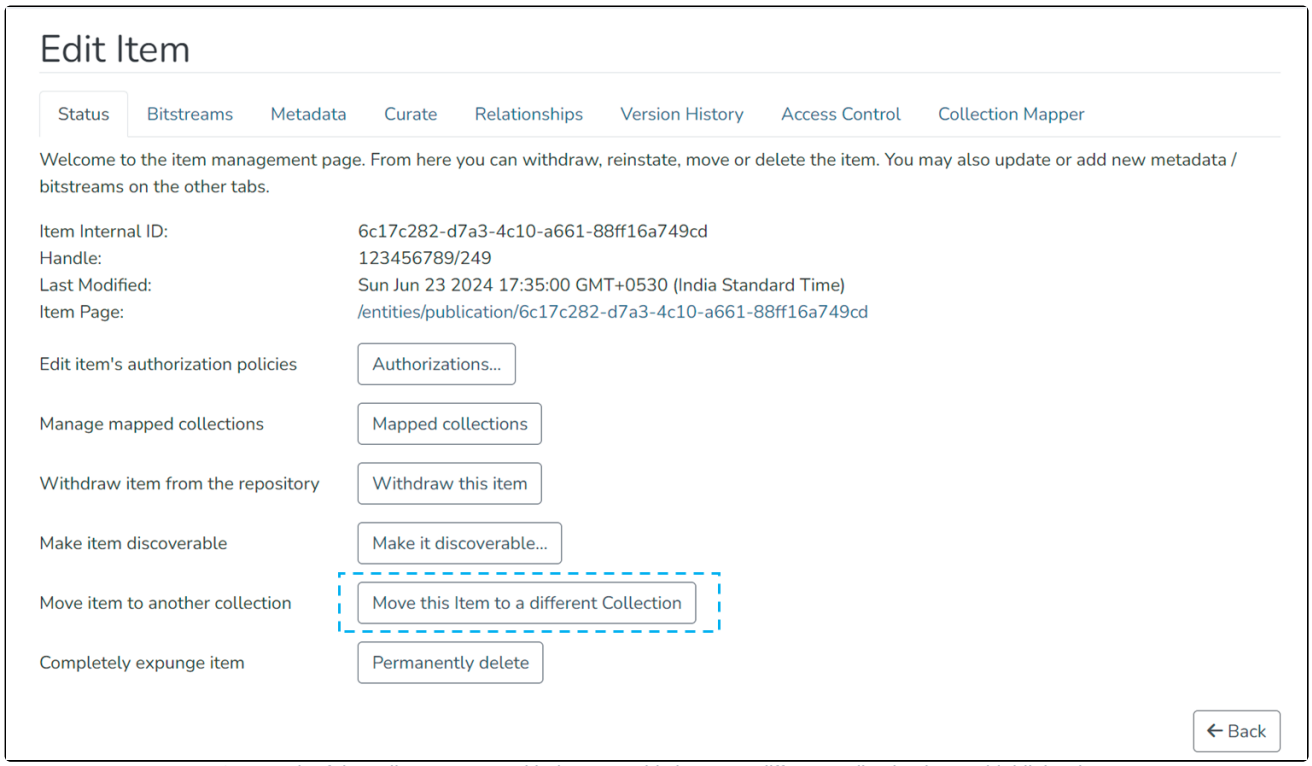
Users can reach an item through multiple methods, including those listed below:

1. Search for an item
2. Browse communities and collections
3. Find an item in the Administration section at Edit > Item
4. Use the existing item URL

Step 3: Click on the “Edit” button, which usually appears on the right side of the item title.



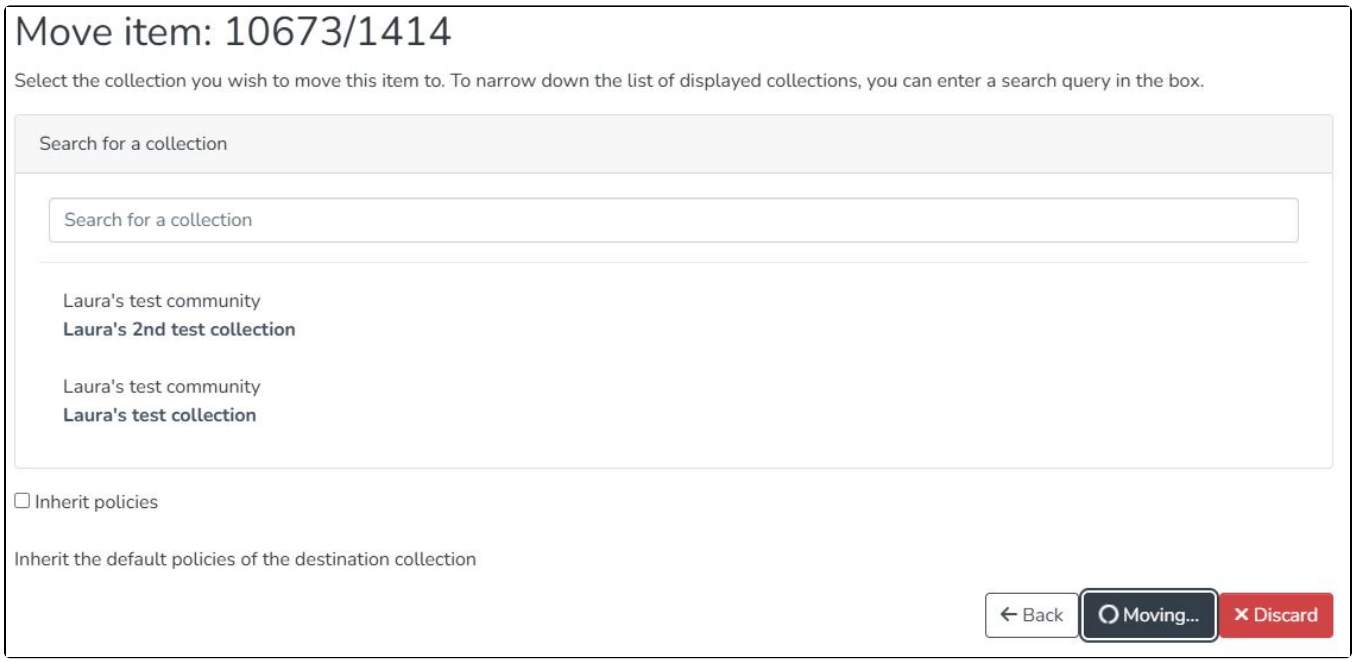
Step 4: Click on the “Status” tab and click the “Move” button.



Status tab of the Edit Item page, with the Move this item to a different collection button highlighted

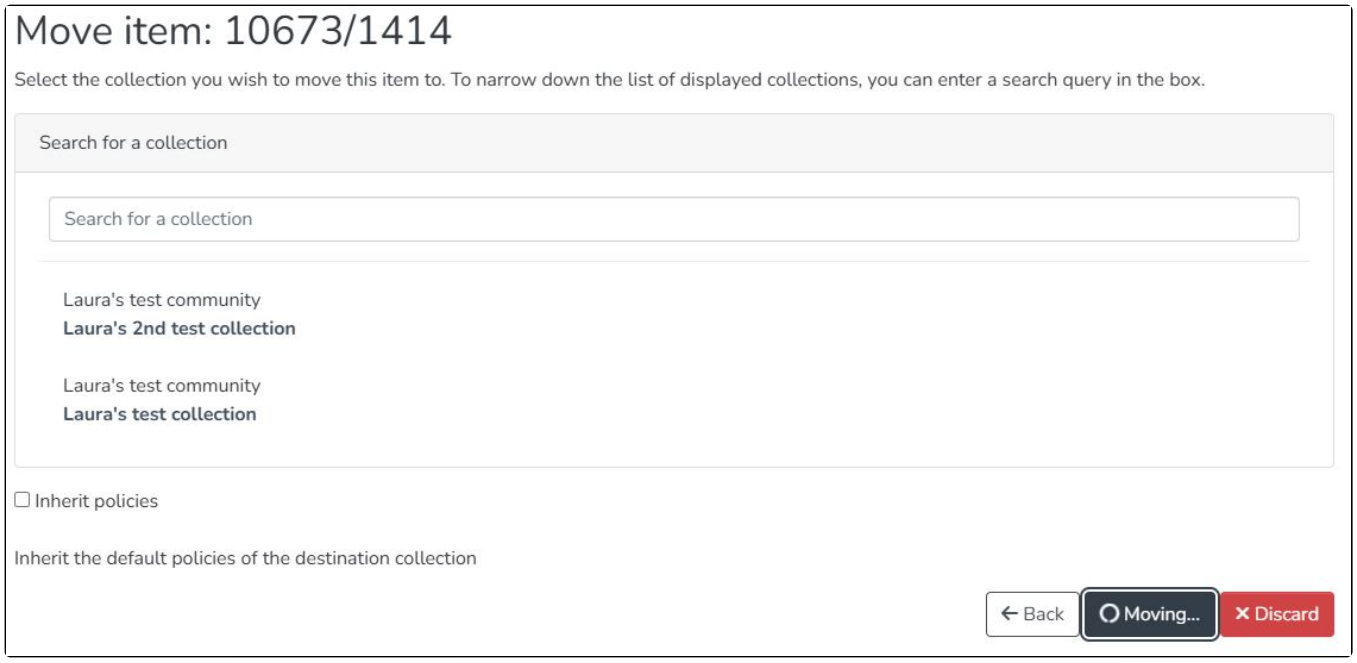
Step 5: Make selections on the Move item page:

1. **Search for a collection.** Enter the destination collection name to move the item to or select the destination collection from the drop-down list. Typing the collection name will narrow the options in the drop-down list.
2. **Inherit policies:** Click on this check box to update the item’s policies according to the destination collection’s policies.
3. **Back:** Click the “Back” button to return to the Item page.
4. **Save:** Click the "Save" button to complete the operation and move the item.
5. **Discard:** Click the “Discard” button to cancel the operation.



Move Item page with options and buttons identified

Step 6: After selecting the destination collection, click on the “Save” button. The item will move to the destination collection upon completing the operation.



Edited and approved by DCAT Documentation Working Group, December 2024.

Versioned Item

- Audience
- Create a version
    - Few important facts
- Access Item's versions
    - Additional information for the version creator

DSpace provides version creation and version management functionality. This functionality enables authorized users to create multiple versions of an item to manage changes in its metadata and attachment while keeping track of differences between two versions. Users can also roll back to the previous version.

Audience

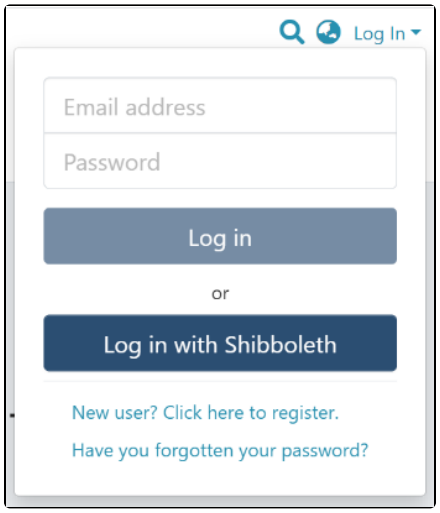
Content Submitters

Community Administrators

System Administrators

Create a version

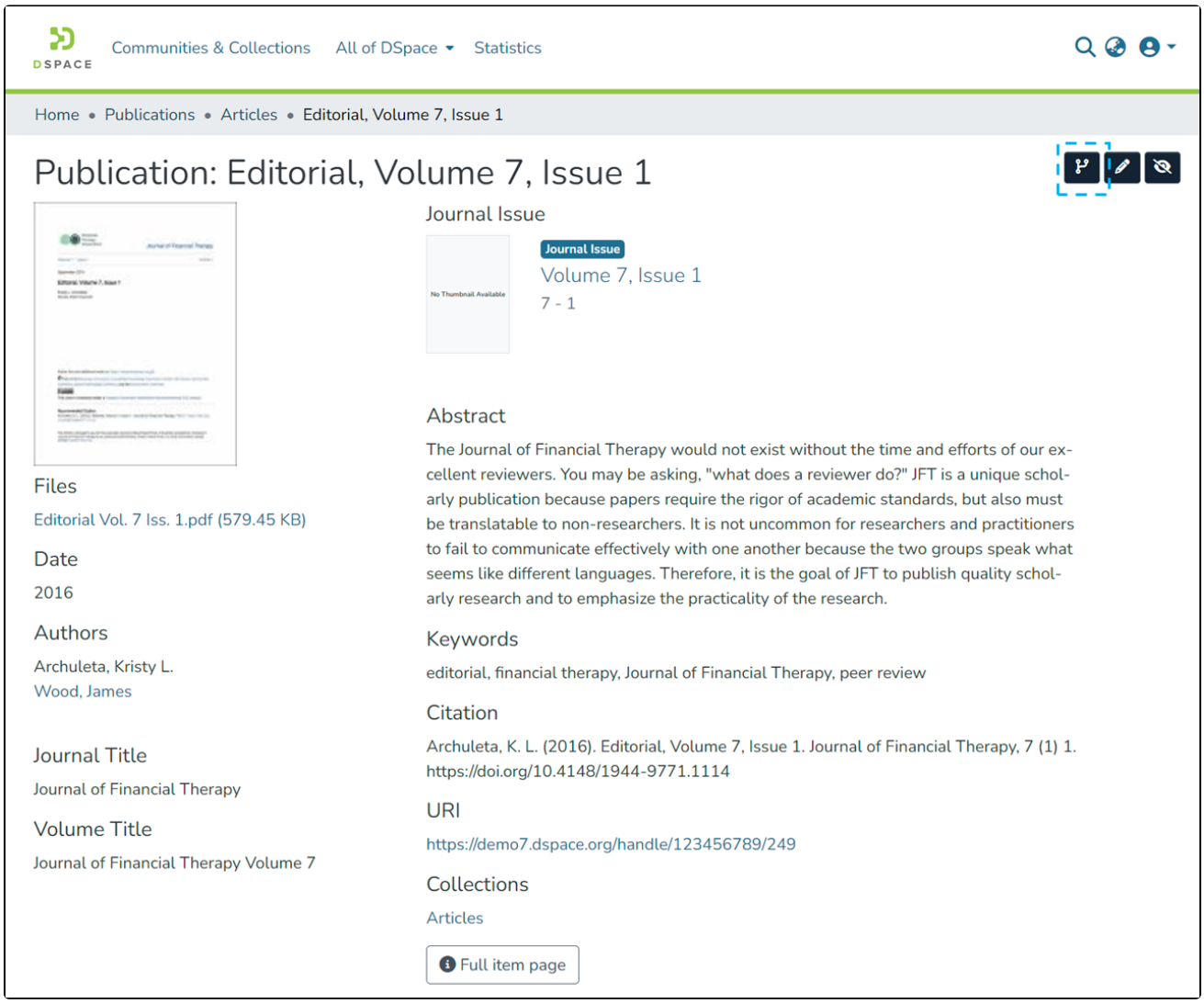
Step 1: Login using the DSpace credentials



Step 2: Users can reach an item to create a version through various methods, which are listed below

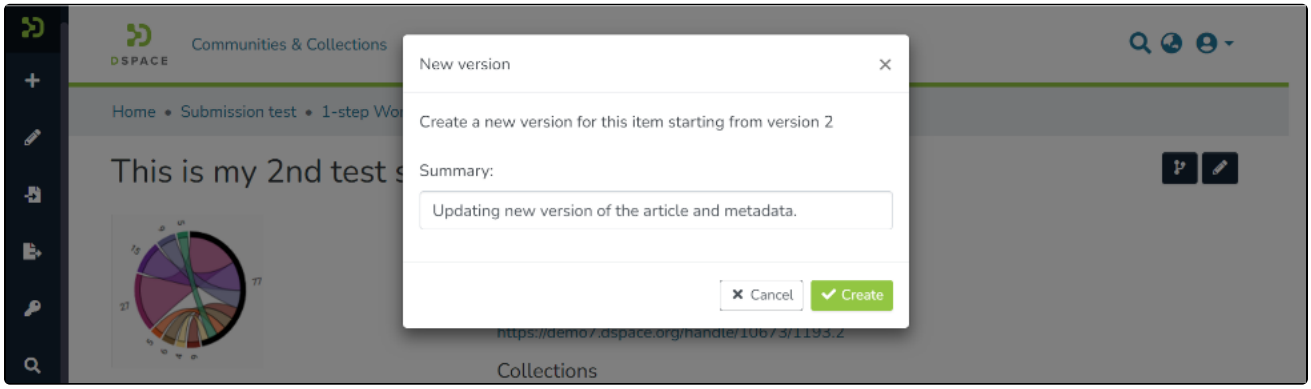
1. Finding an item using search functionalities of the DSpace
2. Reaching an item by browsing communities and collections
3. Finding an item in the Administration section at Edit > Item

Step 3: Users will see the "Create Version" button on the item detail page highlighted below. Click it to create a new version of an item.

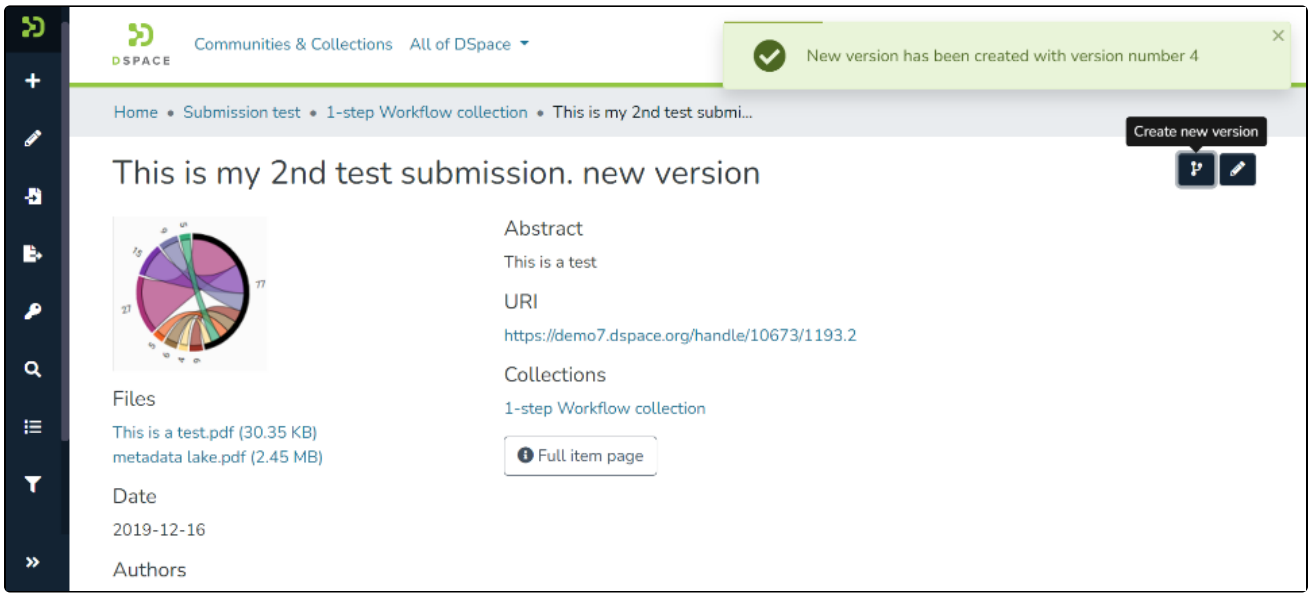


Step 4: After clicking the “Create Version” button, users will see a prompt seeking a summary of the new version. Please enter a summary of changes users will make in the latest version.

Later, this summary plays an essential role in tracking changes made in the version that helps the broader user group and auditors.



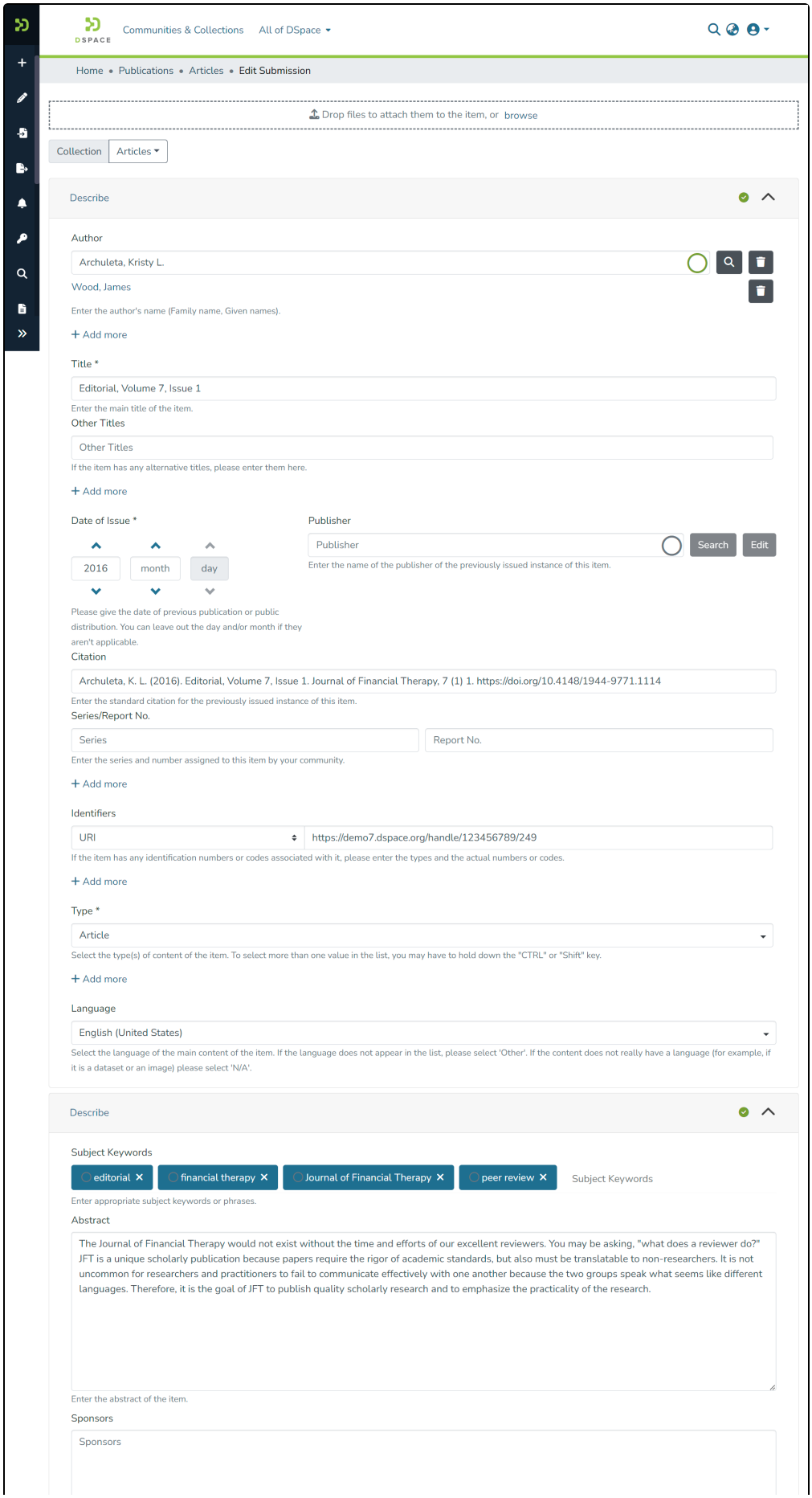
Step 5: Users will see a success prompt confirming a new version creation, as shown below. A page similar to the item submission process will appear with the item’s existing metadata and attachments in an editable mode.

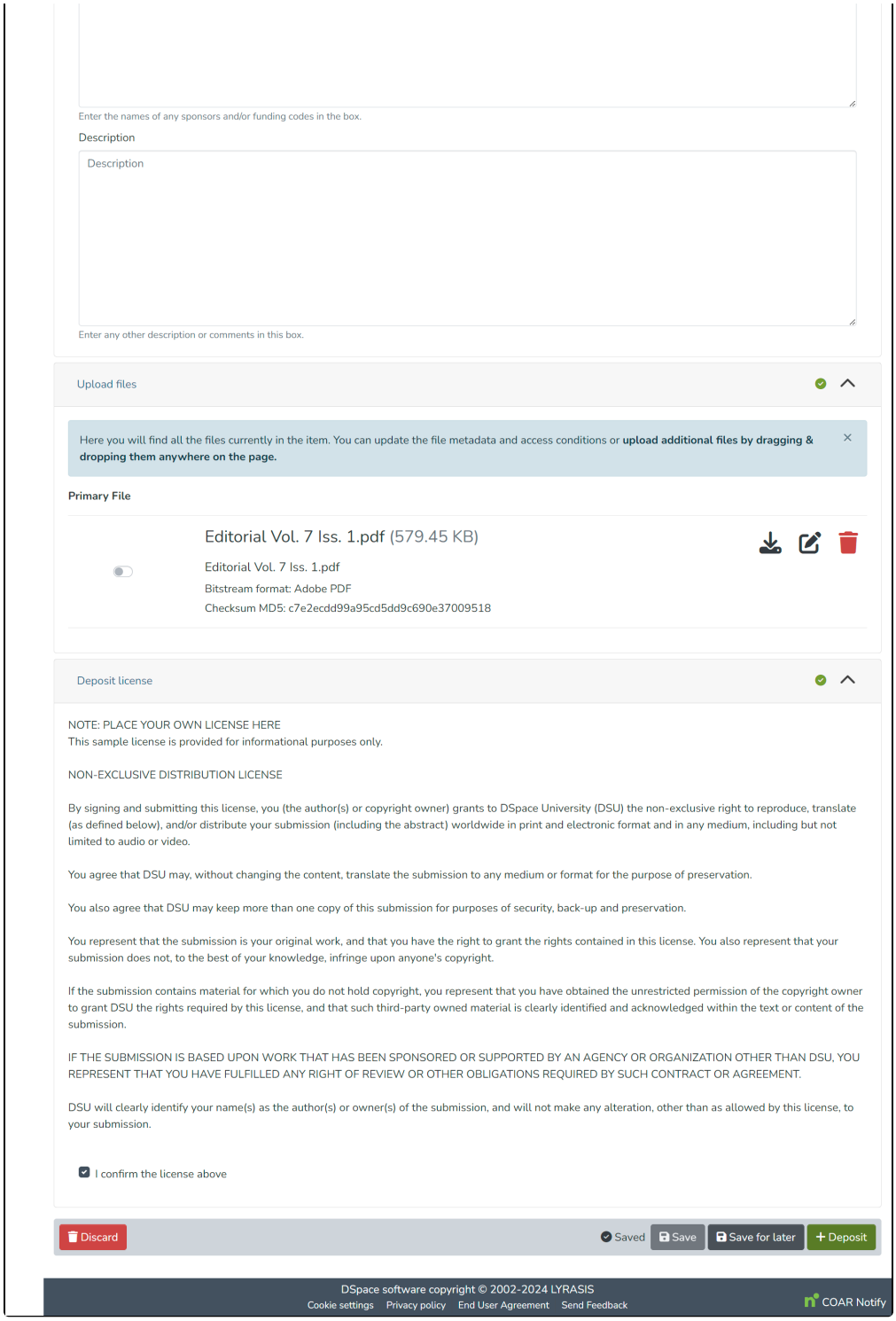


Step 6: Users can update required metadata and attachments on this page the same way they would have done during the item submission process.

Few important facts

- *Users can update and add new metadata during the version update process*
- *Like the metadata, one can also update attachments by updating/removing existing attachments and adding new ones.*
- *It's possible to assign a new collection to the latest version. However, it does not change the storage location of the old version.*
- *Suppose the collection where the latest version needs to be stored has approval workflows assigned. The newest version will be published after necessary approvals.*
- *Users can save the draft version during updates and pick it up from their workspace to complete later.*


Communities & Collections
All of DSpace
Home • Publications • Articles • Edit Submission
Drop files to attach them to the item, or browse
Collection
Articles
Describe
Author
Archuleta, Kristy L.
Wood, James
Enter the author's name (Family name, Given names).
+ Add more
Title *
Editorial, Volume 7, Issue 1
Enter the main title of the item.
Other Titles
Other Titles
If the item has any alternative titles, please enter them here.
+ Add more
Date of Issue *
2016
month
day
Please give the date of previous publication or public distribution. You can leave out the day and/or month if they aren't applicable.
Publisher
Publisher
Search
Edit
Enter the name of the publisher of the previously issued instance of this item.
Citation
Archuleta, K. L. (2016). Editorial, Volume 7, Issue 1. Journal of Financial Therapy, 7 (1) 1. https://doi.org/10.4148/1944-9771.1114
Enter the standard citation for the previously issued instance of this item.
Series/Report No.
Series
Report No.
Enter the series and number assigned to this item by your community.
+ Add more
Identifiers
URI
https://demo7.dspace.org/handle/123456789/249
If the item has any identification numbers or codes associated with it, please enter the types and the actual numbers or codes.
+ Add more
Type *
Article
Select the type(s) of content of the item. To select more than one value in the list, you may have to hold down the "CTRL" or "Shift" key.
+ Add more
Language
English (United States)
Select the language of the main content of the item. If the language does not appear in the list, please select 'Other'. If the content does not really have a language (for example, if it is a dataset or an image) please select 'N/A'.
Describe
Subject Keywords
editorial
financial therapy
Journal of Financial Therapy
peer review
Subject Keywords
Enter appropriate subject keywords or phrases.
Abstract
The Journal of Financial Therapy would not exist without the time and efforts of our excellent reviewers. You may be asking, "what does a reviewer do?" JFT is a unique scholarly publication because papers require the rigor of academic standards, but also must be translatable to non-researchers. It is not uncommon for researchers and practitioners to fail to communicate effectively with one another because the two groups speak what seems like different languages. Therefore, it is the goal of JFT to publish quality scholarly research and to emphasize the practicality of the research.
Enter the abstract of the item.
Sponsors
Sponsors



Step 7: Click on the “Deposit” button to complete the version creation process. Apart from clicking the “Deposit” button, users can perform the following actions during the version creation:

- Discard: Click this button to cancel the version creation process.
- Save: Keep saving the updates intermittently and continue updating version details.
- Save for later: Save the progress and pick up the item from the workspace later to complete the version creation.

As briefed above, if the target collection has the approval workflow assigned to it, then the latest version will appear to users having an approval role for acceptance. However, if no workflow is set to the collection, the new version will be published for public access.

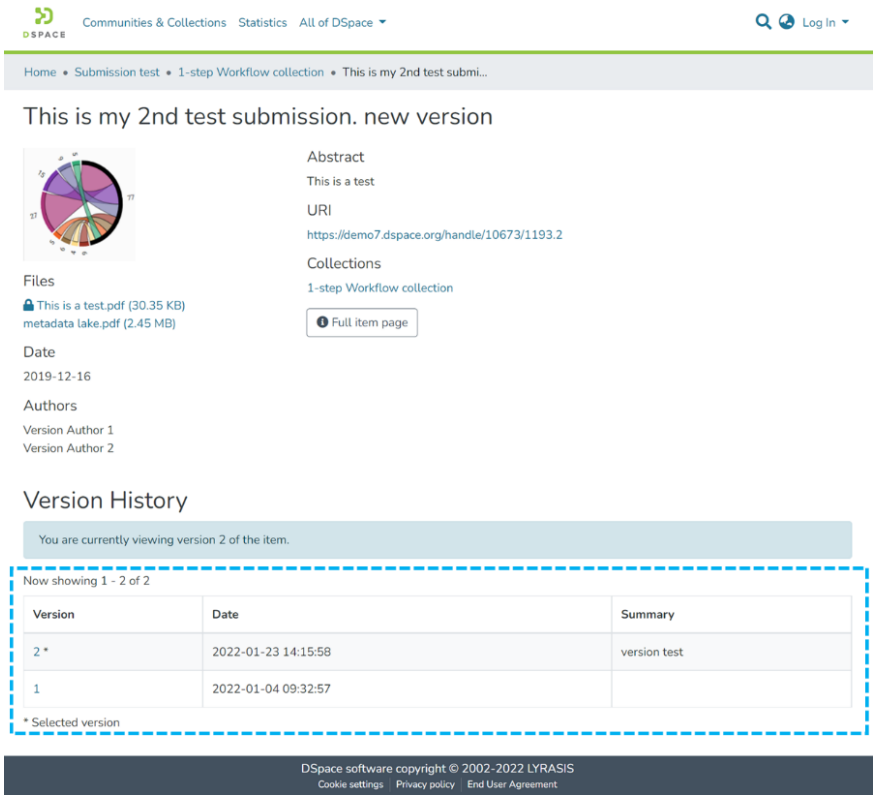
Access Item’s versions

Step 1: Users can reach an item through various methods, which are listed below

1. Finding an item using search functionalities of the DSpace
2. Reaching an item by browsing communities and collections

Step 2: Users can scroll down the item details page to see its version history, as illustrated below. Version history table shows the following details:

- **Version:** The version number of the item. The illustration shows that the selected version has * next to it.
- **Date:** Version creation date and time as per the server.
- **Summary:** Summary added by the user during version creation.

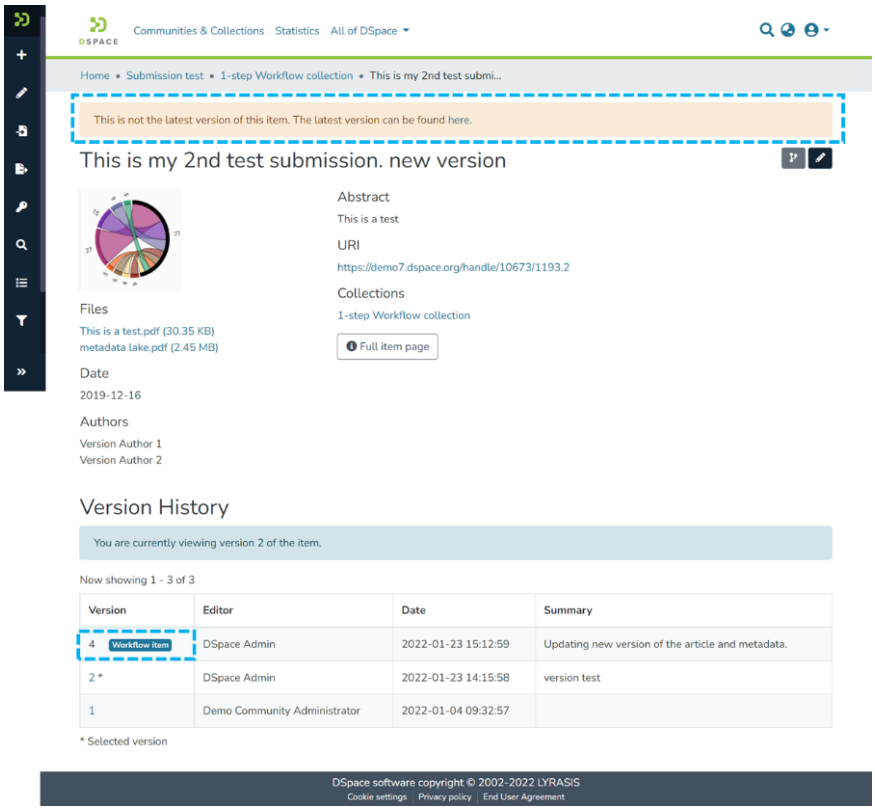


Step 3: The item details page shows information from the latest version. Users can click on the previous version id to see it.

Additional information for the version creator

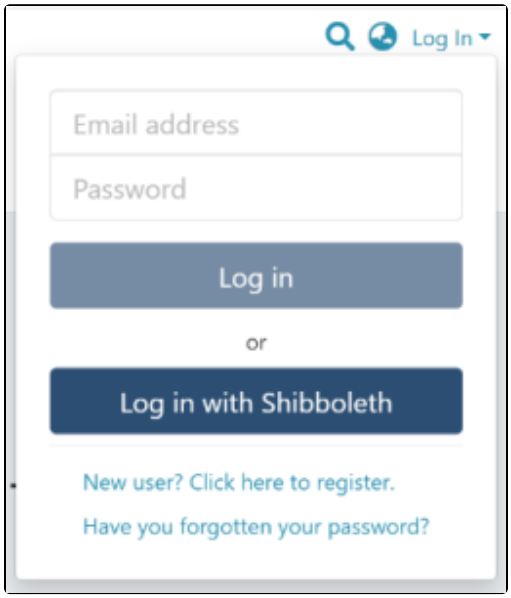
In addition to the above details, the version creator will see the following information related to versioning.

- ***The version in the approval workflow:*** *If a version is unpublished due to pending approval, then the "Workflow Item" tag will appear next to such versions. These versions are not visible to all users.*
- ***Alert about the latest version:*** *An alert confirms the page is not the newest version, and a link to the newest version appears at the top of the page.*


Communities & Collections
Statistics
All of DSpace
Home • Submission test • 1-step Workflow collection • This is my 2nd test submi...
This is not the latest version of this item. The latest version can be found here.
This is my 2nd test submission. new version
Abstract
This is a test
URI
https://demo7.dspace.org/handle/10673/1193.2
Collections
1-step Workflow collection
Full item page
Files
This is a test.pdf (30.35 KB)
metadata lake.pdf (2.45 MB)
Date
2019-12-16
Authors
Version Author 1
Version Author 2
Version History
You are currently viewing version 2 of the item.
Now showing 1 - 3 of 3
Version
Editor
Date
Summary
4 Workflow item
DSpace Admin
2022-01-23 15:12:59
Updating new version of the article and metadata.
2 *
DSpace Admin
2022-01-23 14:15:58
version test
1
Demo Community Administrator
2022-01-04 09:32:57
* Selected version
DSpace software copyright © 2002-2022 LYRASIS
Cookie settings
Privacy policy
End User Agreement

Withdraw an item

Step 1: Login using the DSpace credentials

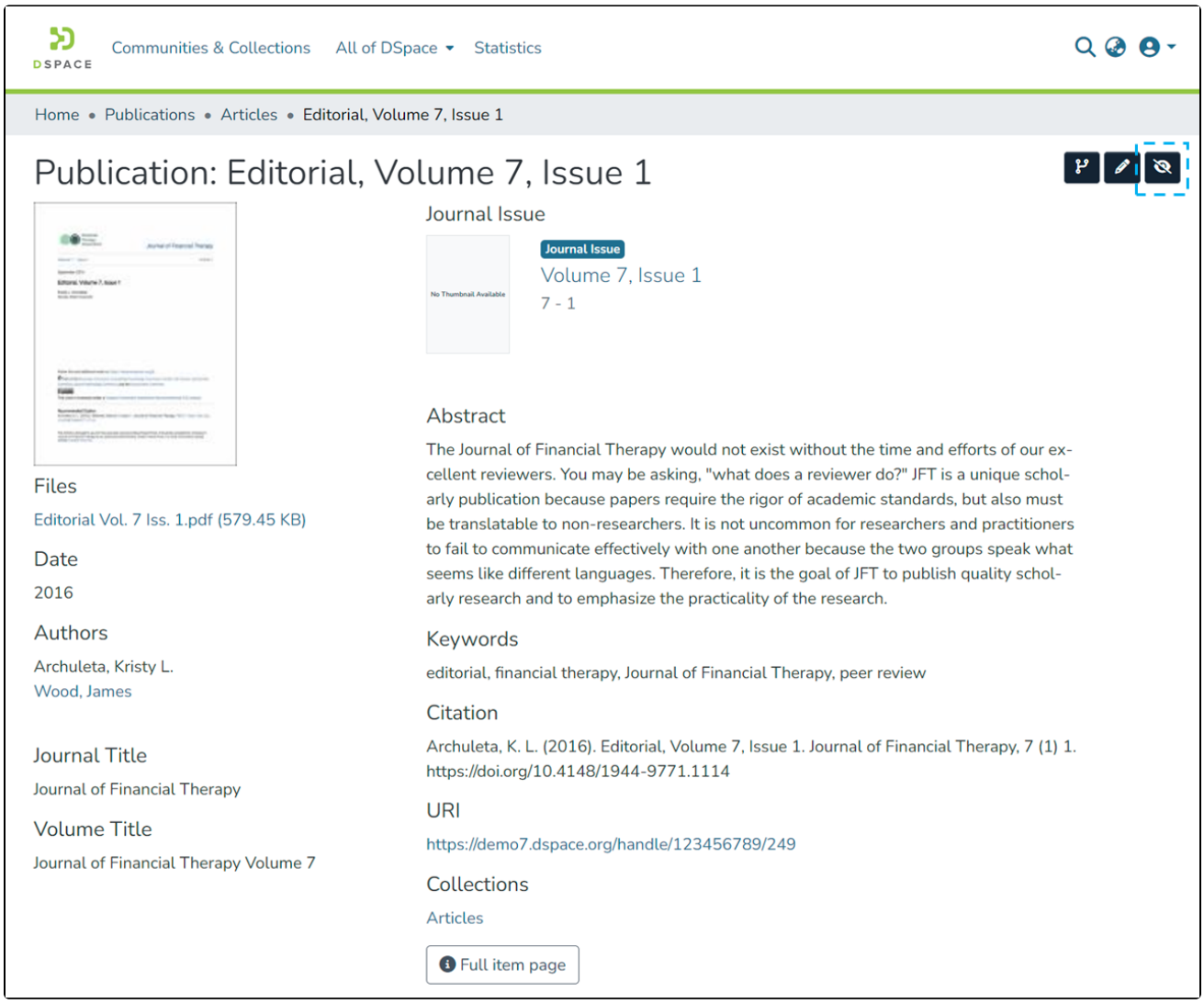


Step 2: Users can go to the item they want to edit

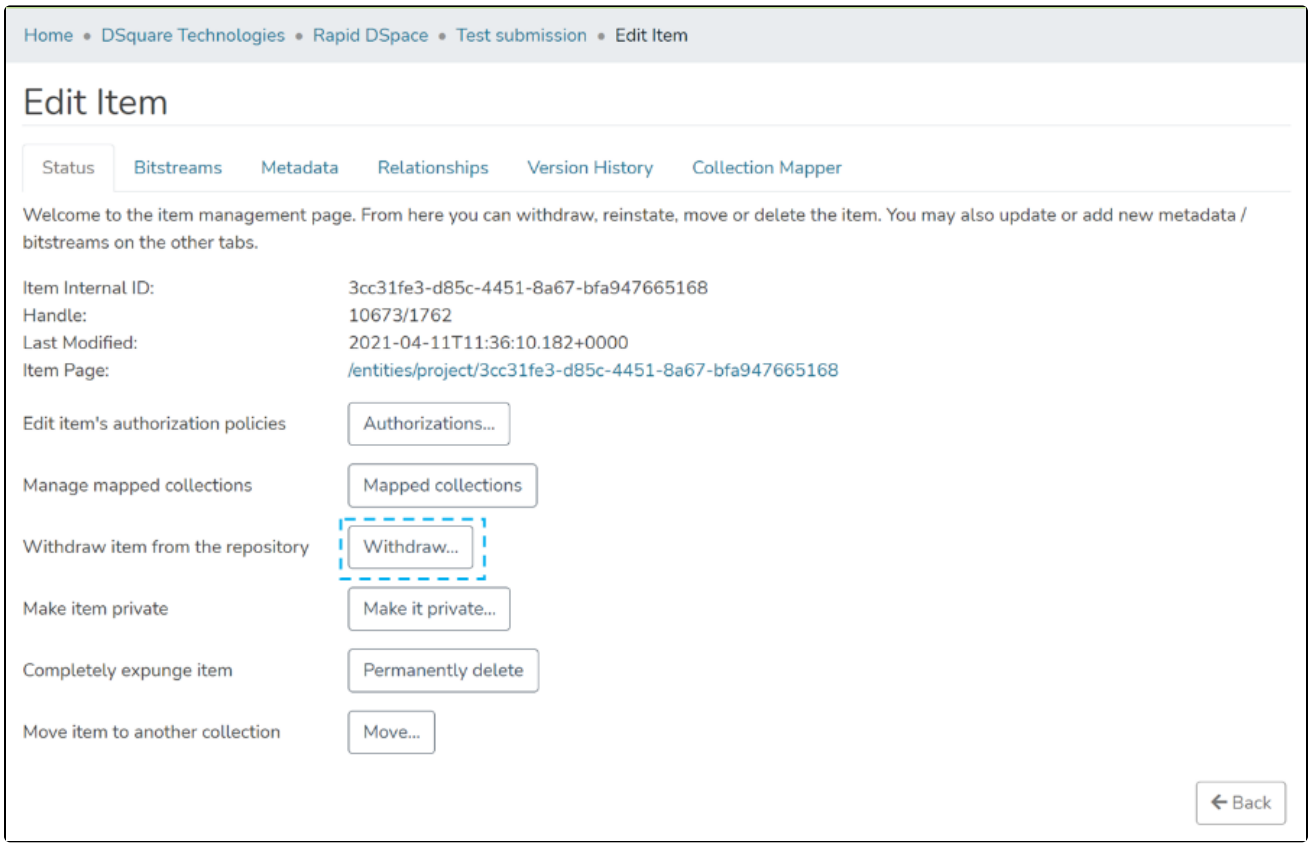
Users can reach an item through multiple methods, which are listed below

1. Finding an item using search functionalities of the DSpace
2. Reaching an item by browsing communities and collections
3. Finding an item in the Administration section at Edit > Item

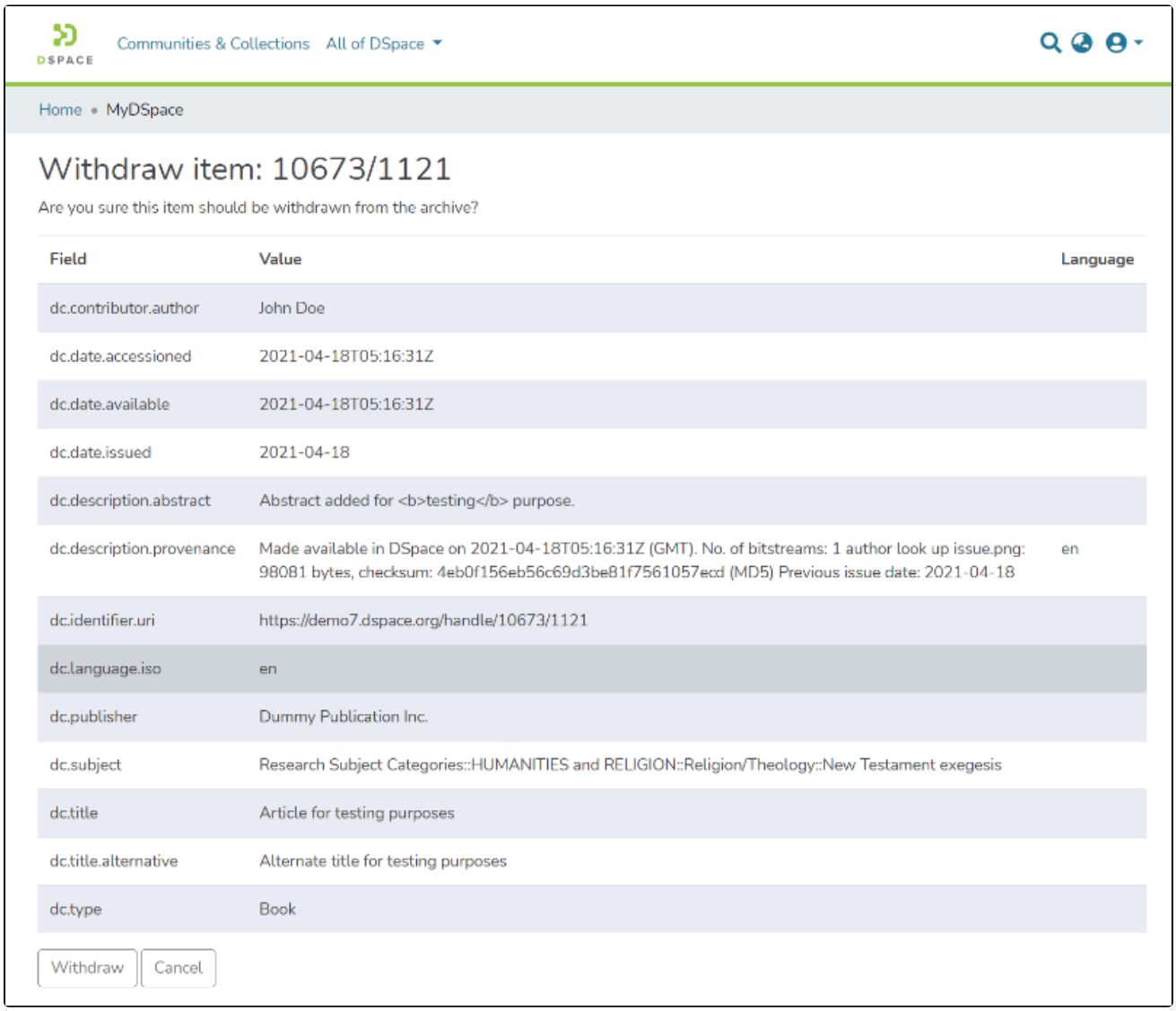
Click the “Withdraw” button on the right-hand side of the item title.



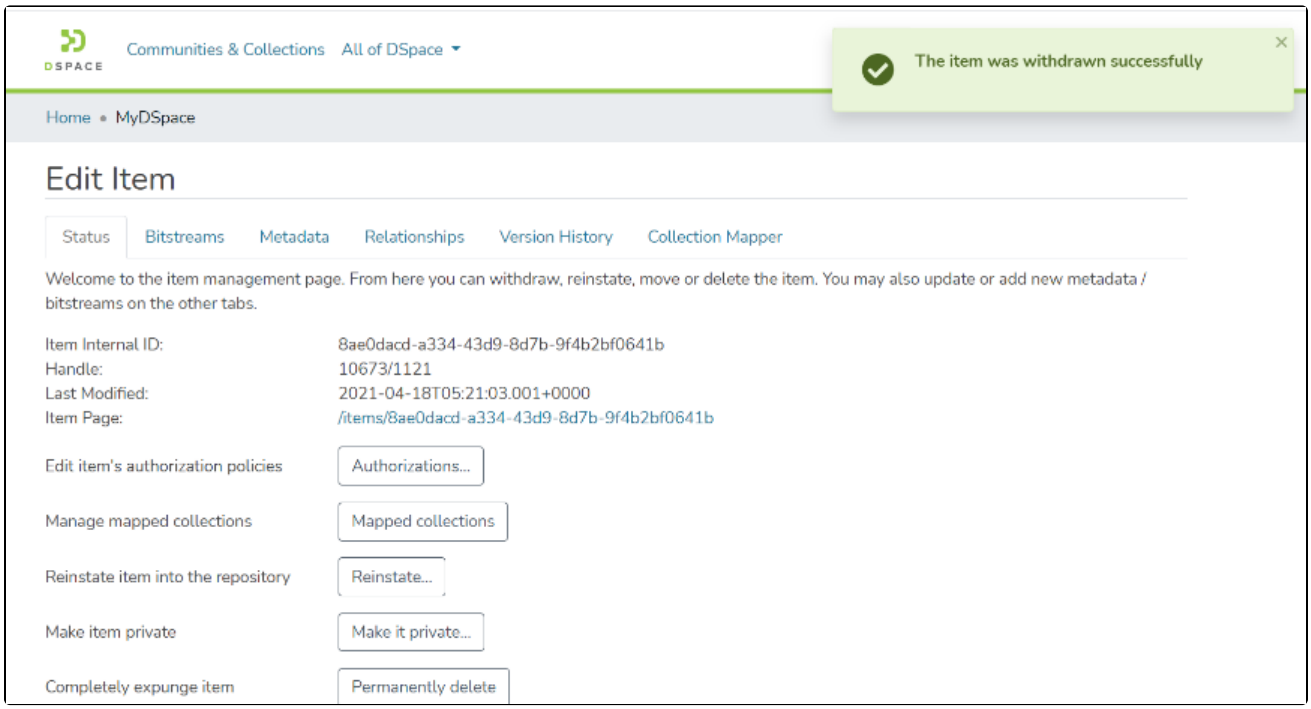
Step 3: Click the “Withdraw” button under the “Status” tab to withdraw the item from the archive.



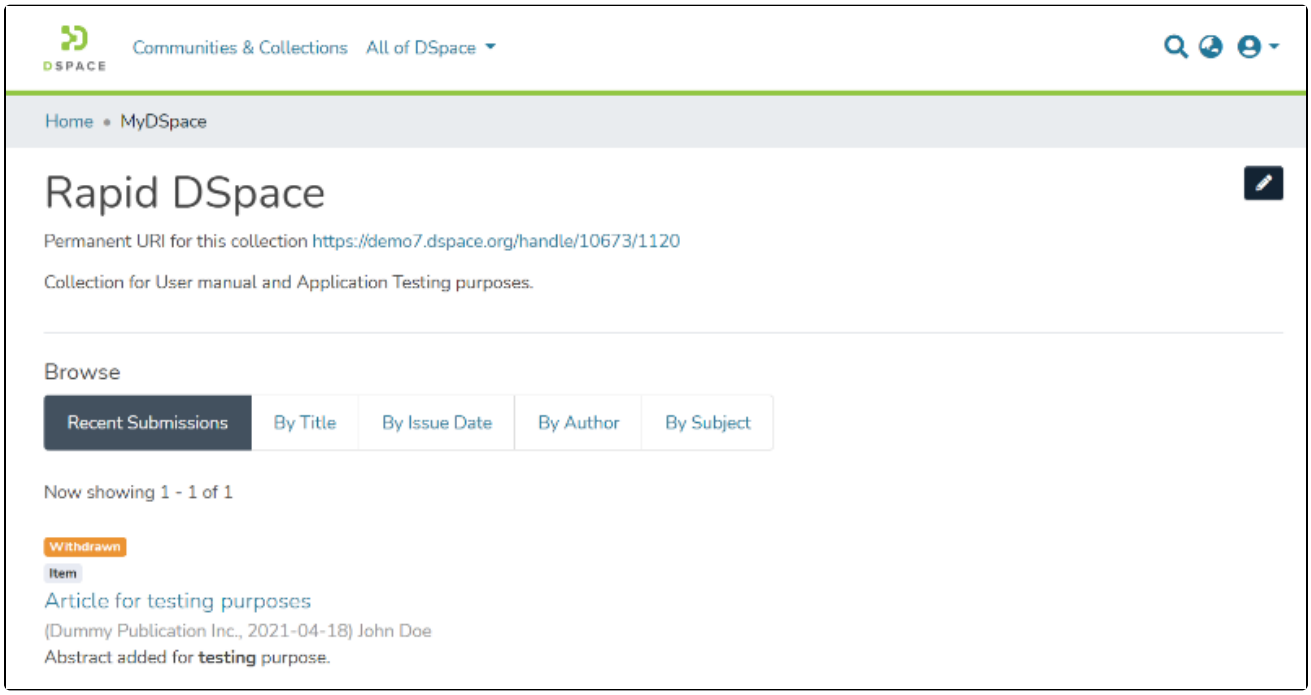
Step 4: Click on the “Withdraw” button to withdraw the item or click the “Cancel” button to cancel the operation.



Step 5: Users will see a success prompt confirming the item withdrawal, as shown below.



Step 6: Users will notice that the item will appear with a “Withdrawn” tag.



Embargo an item

“Embargo an item” helps restrict the Item’s attachment’s access until a future date. A user can embargo an item while submitting it or later by editing it. Both methods to embargo an item are explained below.

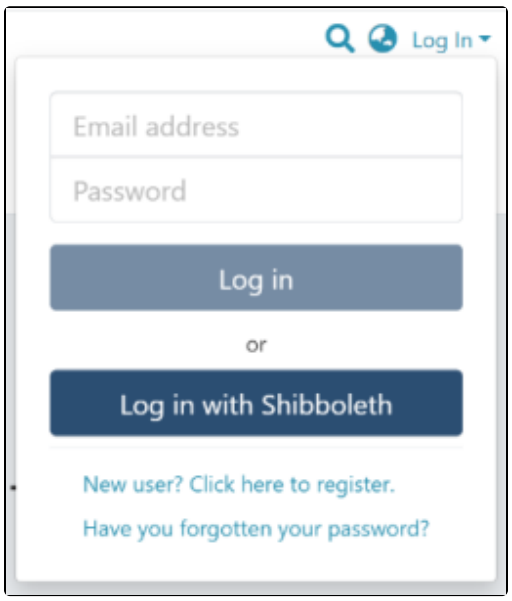
- Audience
- Embargo an item during the item submission
- Embargo an item via edit item

Audience

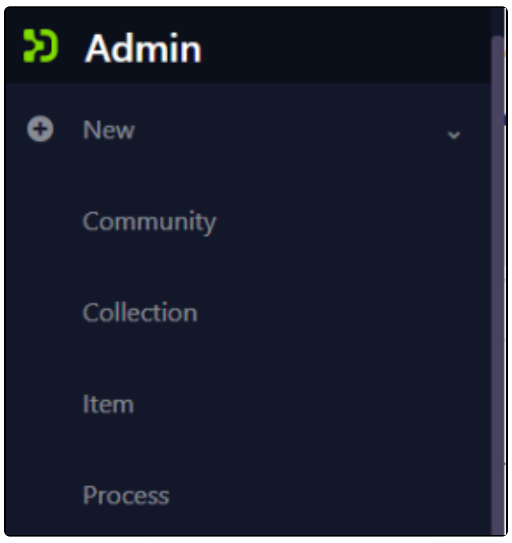
1. Repository Administrator
2. Community Administrator
3. Collection Administrator
4. Item Administrator/submitter

Embargo an item during the item submission

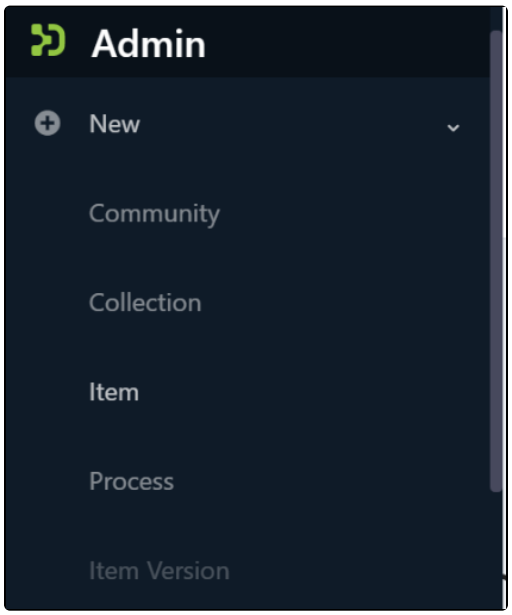
Step 1: Login using your credentials



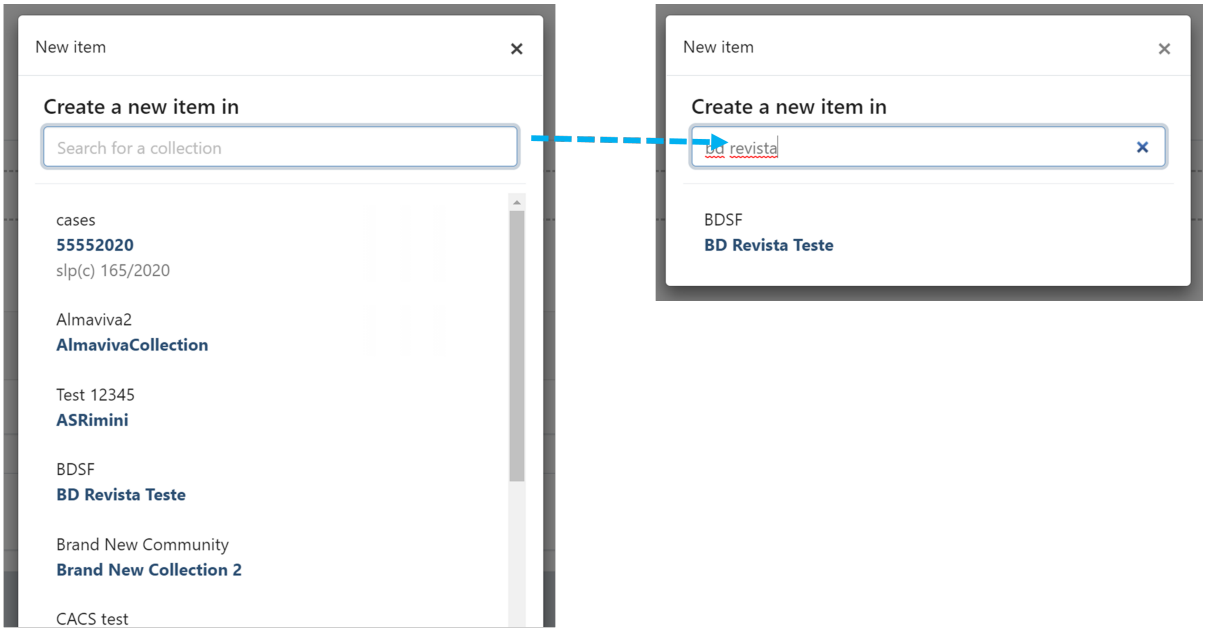
Step 2: Roll over the cursor on the “+” sign.



Step 3: Click on “New” and click on “item” to proceed further in the Item addition process

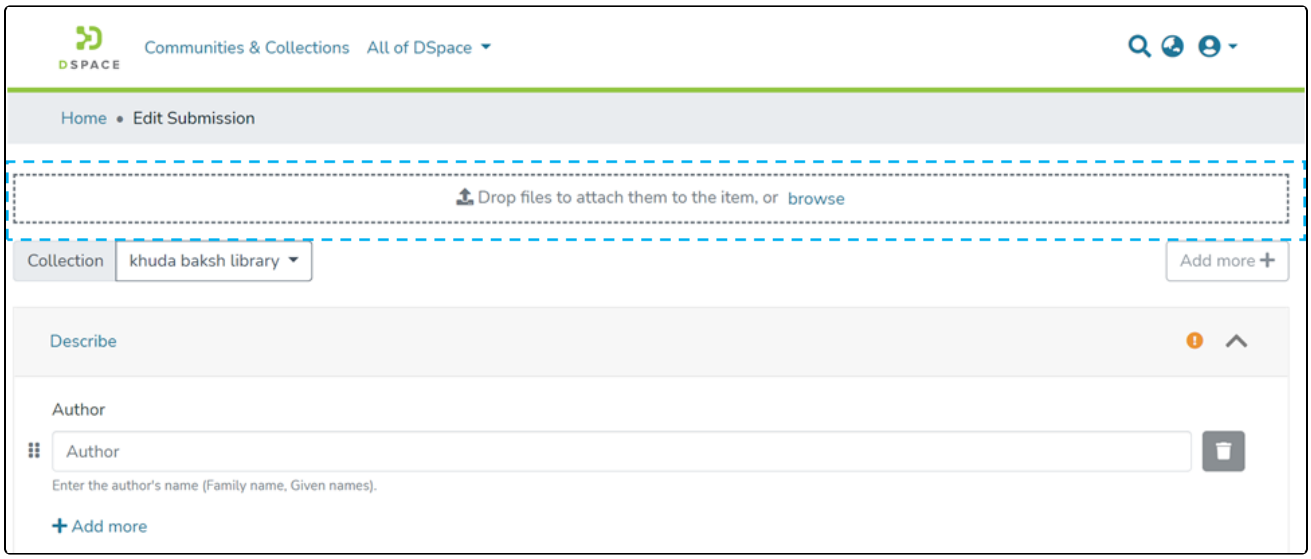


Step 4: A popup window with a collection list will appear. The user can select the target collection by typing its name or scrolling down the collection list. Then, click on the collection to initiate item submission.

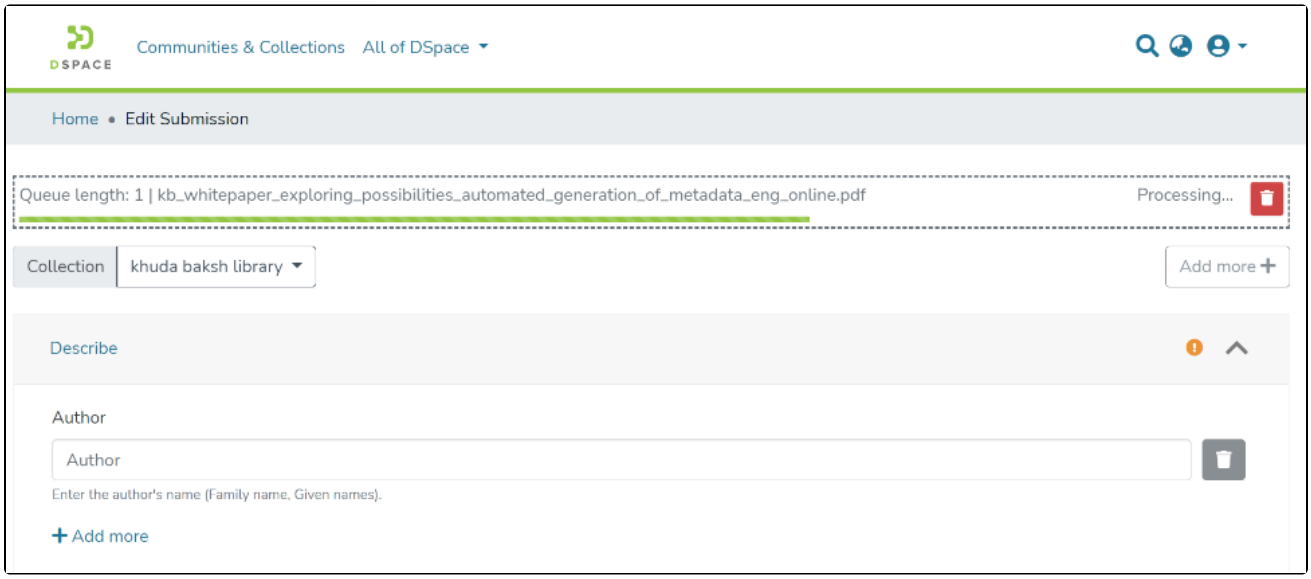


Step 5: Users will see the item submission form after selecting the target collection. The first step is to upload attachment(s) in the Item. In DSpace terminology, an attachment is known as a “bitstream.”

Click on the “browse” link to upload attachment(s). Users can upload multiple files by selecting them together or dragging them into the space.



A progress bar showing bitstream upload progress will appear, as demonstrated in the illustration below. In addition, after a successful bitstream upload, a prompt confirming success or failure will appear.



Bitstream upload in progress

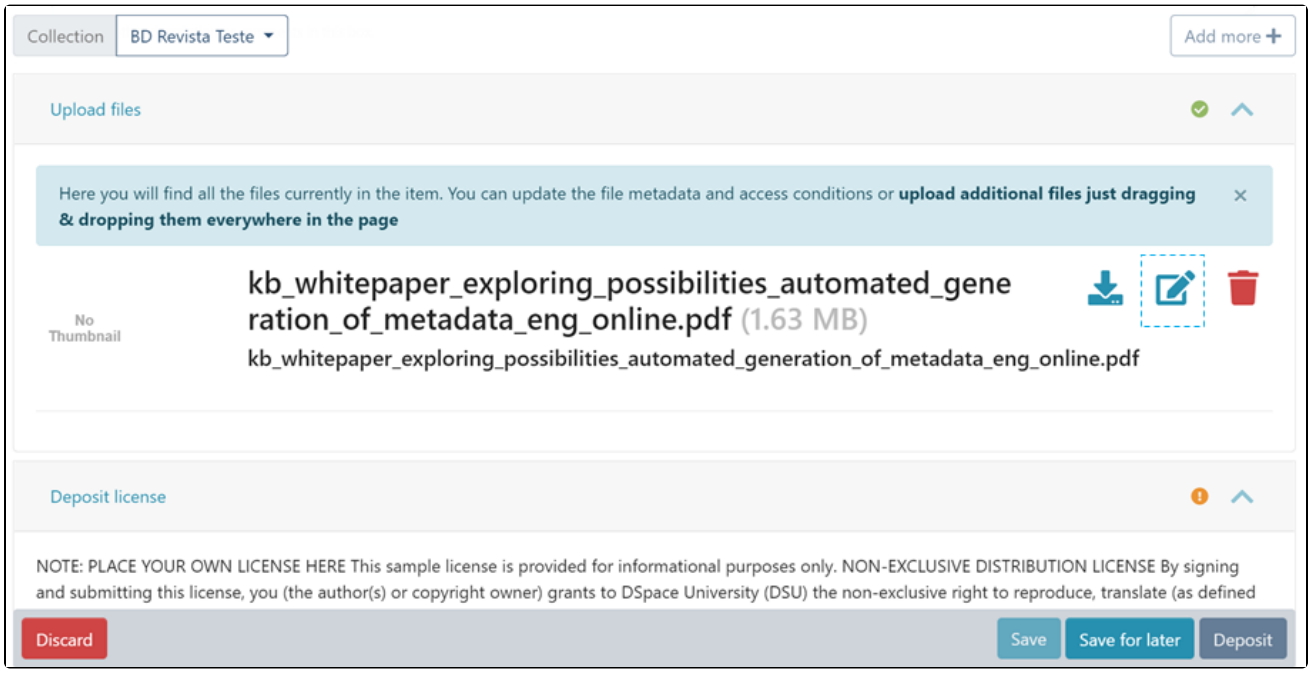


Bitstream Upload Successful

Step 6: After uploading bitstream, the next step is to describe the Item by adding metadata.

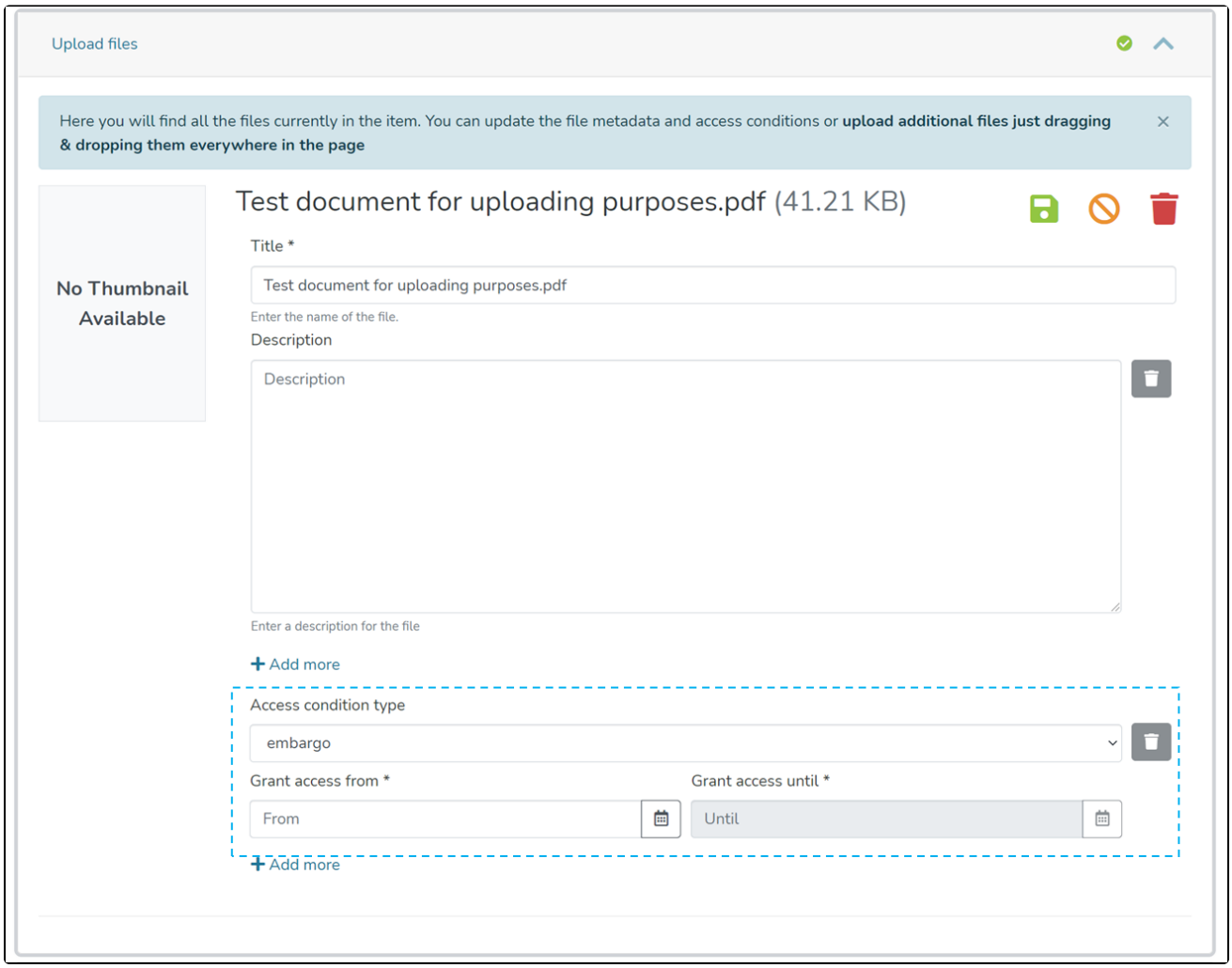
Please refer **Add Item** process for detailed documentation on populating information in the metadata fields.

Step 7: Click on the edit button against any attachment to add embargo policy.



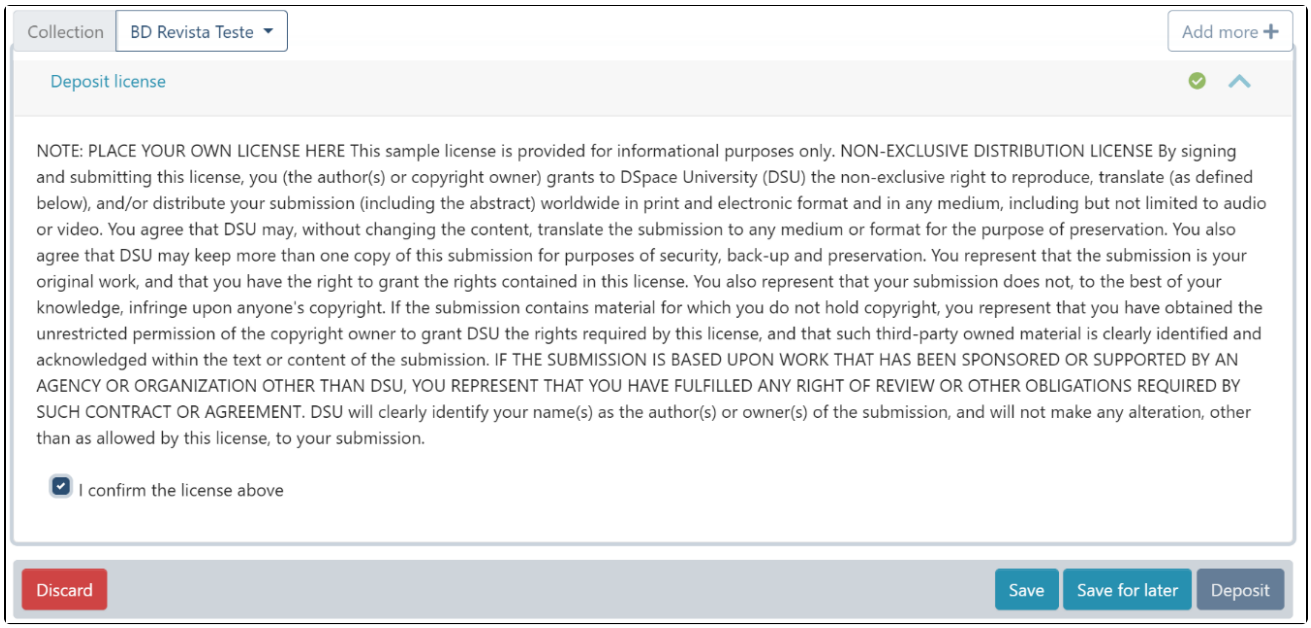
Step 8: Users can apply multiple policies on an attachment, and there are various available options. Click on the dropdown list under the “Access condition” type and select embargo, as highlighted in the screenshot below.

After selecting embargo in the dropdown list, the “Grant access from” date field will be activated. Next, users can choose the future date, after which the attachment should be accessible to the larger set of DSpace users.



Users can add multiple policies to the attachment by clicking the “Add more” link. For example, a user can define an embargo on an item until a future date. Likewise, a lease policy can keep the attachment open access until another date in the future.

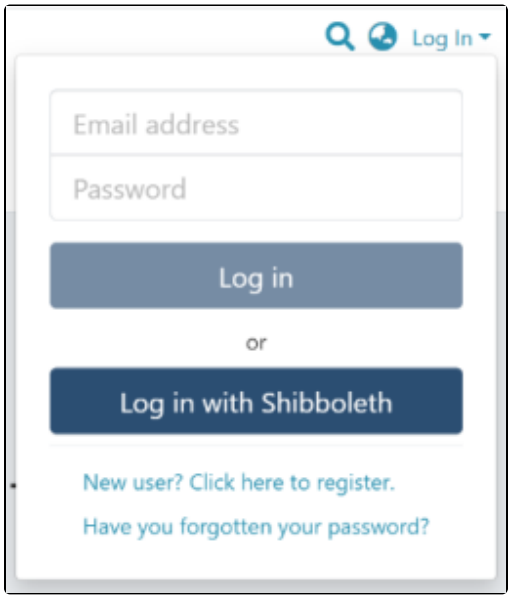
Step 9: After updating all information, the submitter clicks on the “I confirm the license above” checkbox to accept the repository’s license.



Step 10: Click on the “Deposit” button to submit the Item in DSpace. Users will get to see a confirmation prompt upon successful submission of the Item.

Embargo an item via edit item

Step 1: Login using your credentials



Step 2: Go to the Item you want to edit

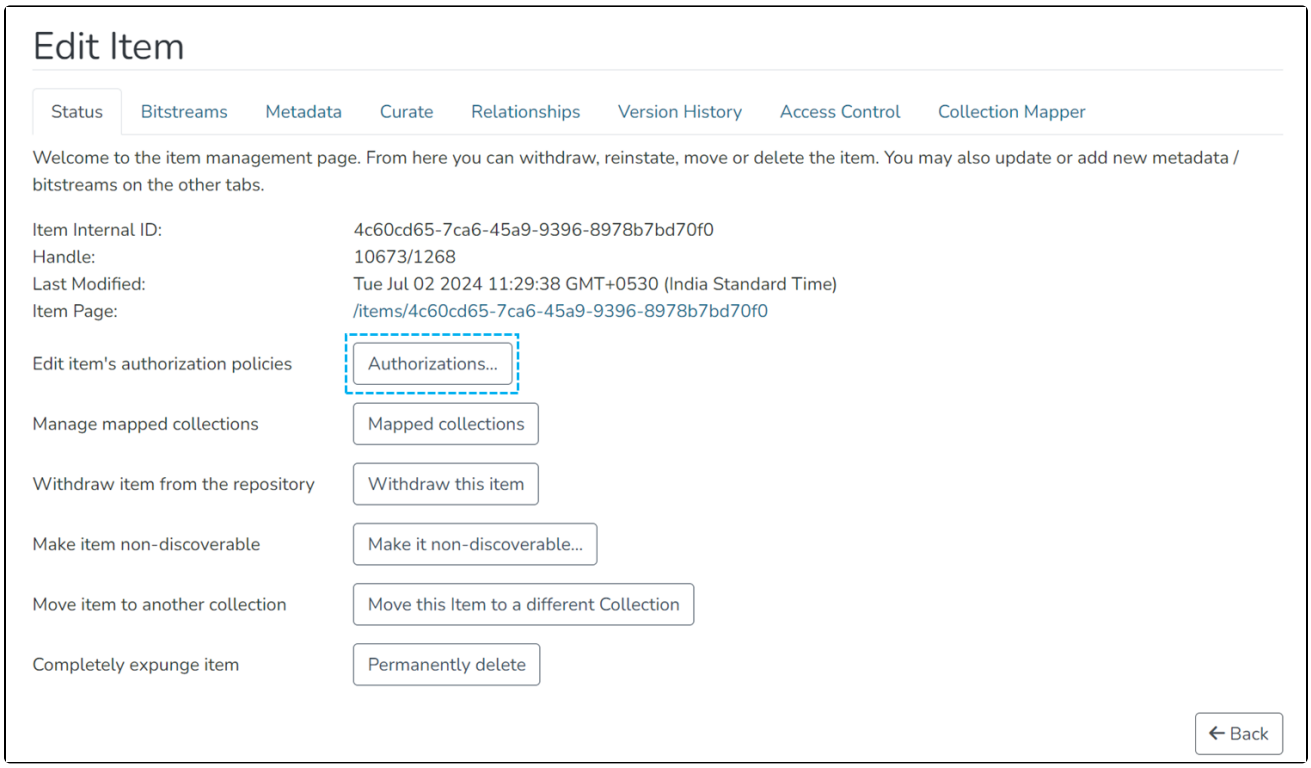
Users can reach an item through multiple methods, as listed below:

1. Search an item
2. Browse communities and collections
3. Finding an item in the Administration section at Edit > Item

Click on the “Edit” button appearing on the right-hand side of the item title.



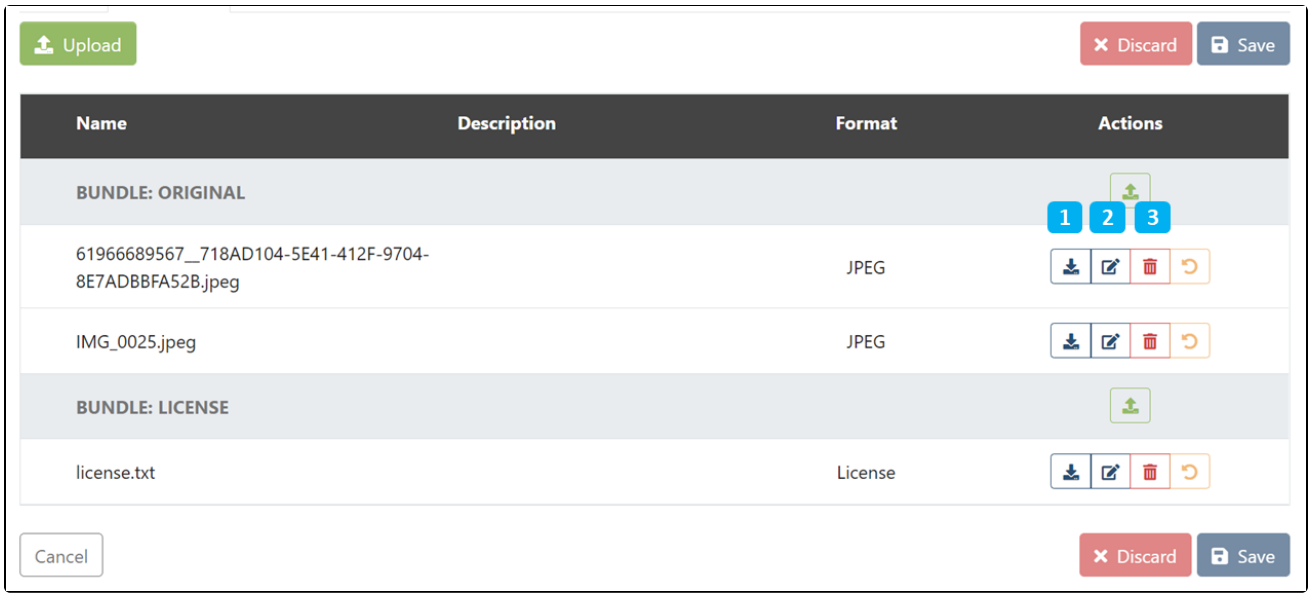
Step 3: Click on the “Authorizations” button under the “Status” tab to continue with adding the embargo policy.



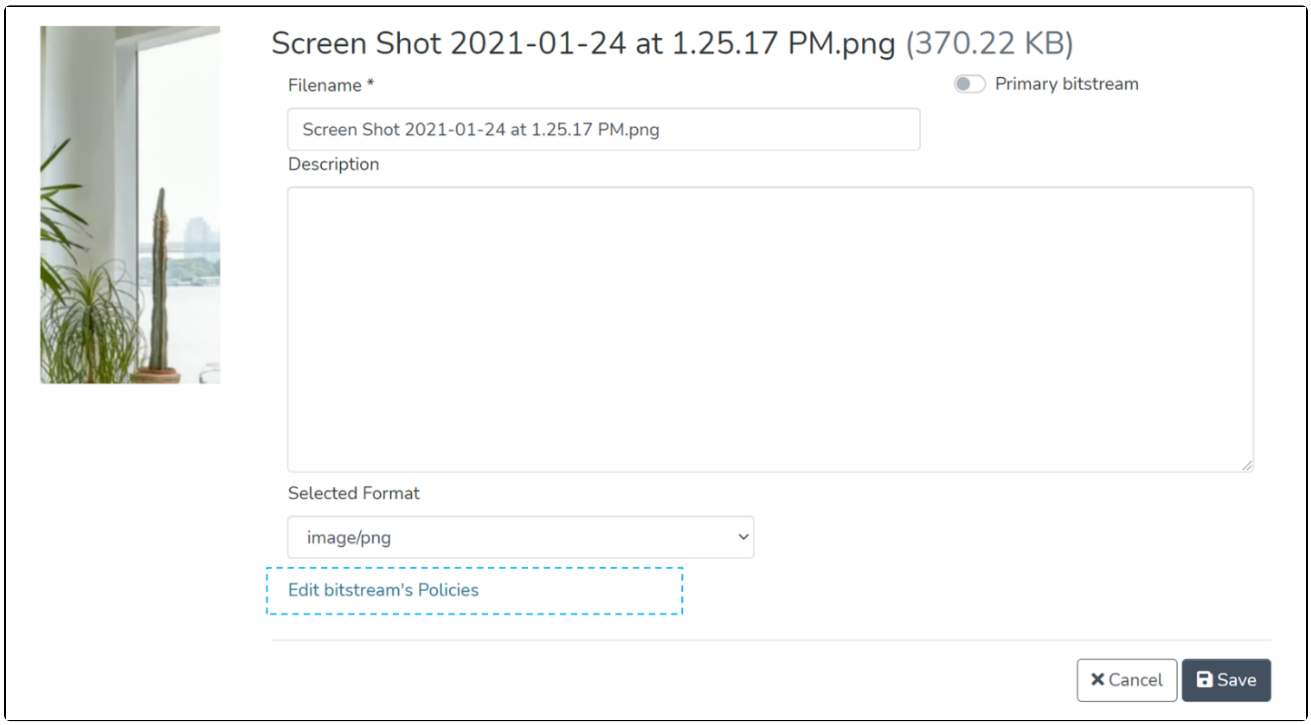
Step 4: The user will see multiple options against each attachment as explained below:

1. **Download Bitstream:** Click on this button to download the attachment on your local device for view.
2. **Edit Bitstream:** Click on the “Edit bitstream” button for editing details. Explained in the next step
3. **Delete Bitstream:** Click on the “Delete Bitstream” button to delete bitstream from the bundle.

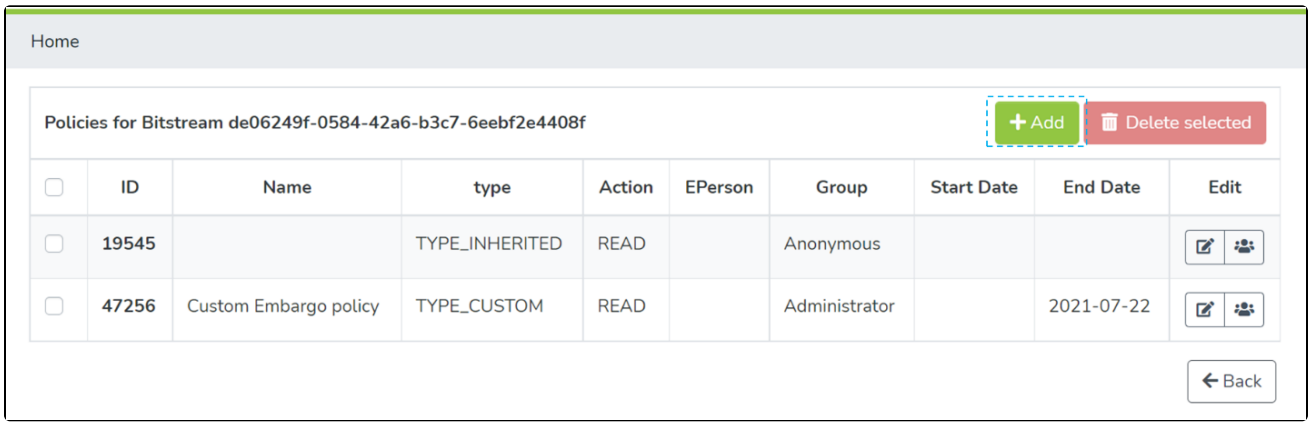
Click on the “edit bitstream” button to embargo the attachment.



Step 5: Click on “Edit bitstream’s Policies” to continue with the embargo process.

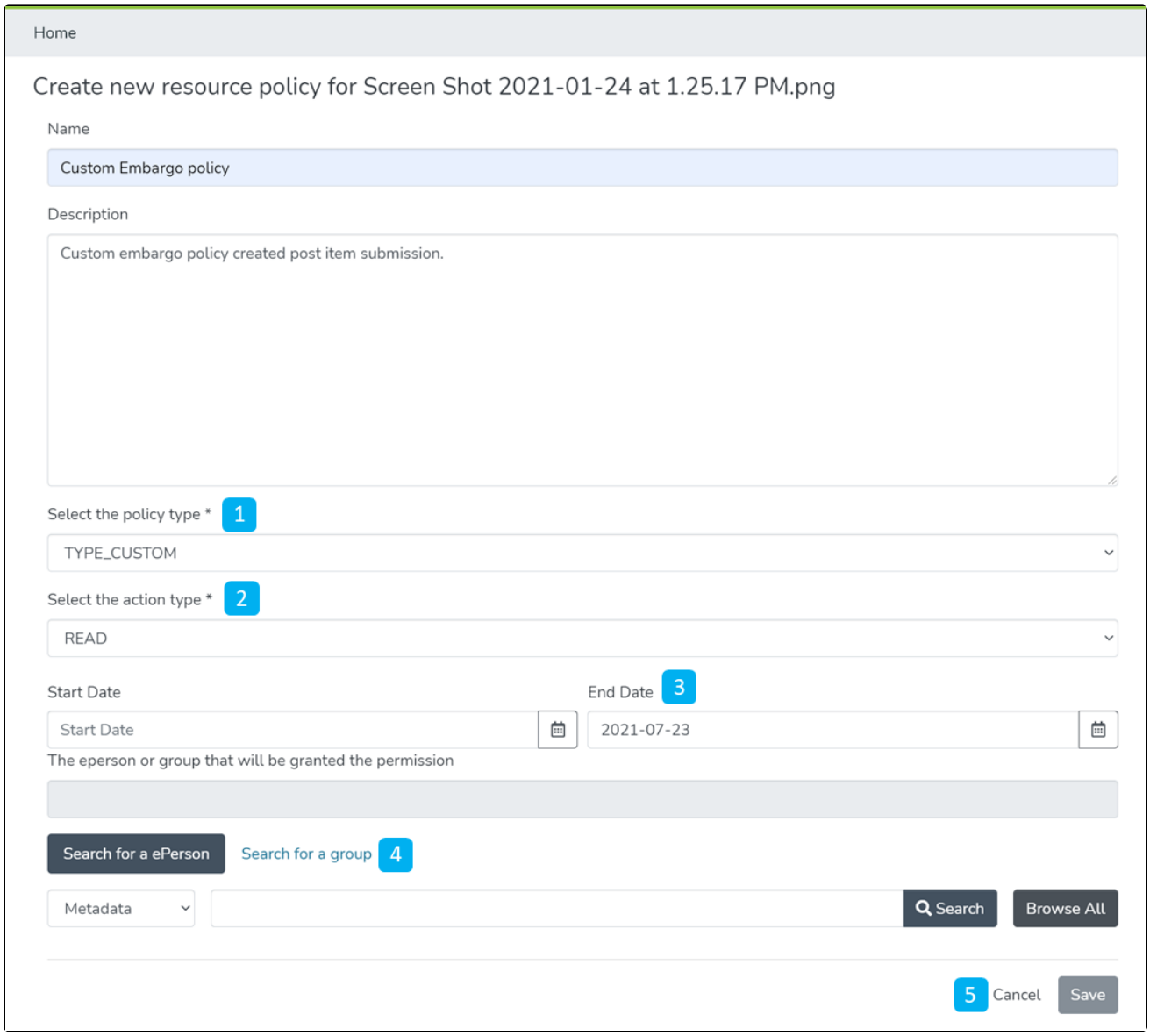


Step 6: Click on the “Add” button to create the custom embargo policy for the attachment.



Step 7: Enter details for creating the embargo policy on this form and perform the following actions:

1. Select ‘TYPE_CUSTOM’ under ‘Select the policy type’ dropdown menu
2. Select ‘READ’ under the ‘Select the action type’ dropdown list.
3. Select a future date under the Start Date to determine when the embargo will end. (The "Start Date" represents the embargo date, as it is when the policy will begin to take effect,)
4. Select E-Person or a User group
5. Click on Save to create the policy



Detailed documentation on various possibilities on this form is available under ‘Edit Bitstream’ user documentation.

Users will see a success prompt upon creating the policy and will be redirected to the bitstream policy page.

Lease an item

“Lease an item” helps restrict the Item’s attachment’s access after a future date. A user can lease an item during the submission or later by editing it. Both methods to lease an item are explained below.

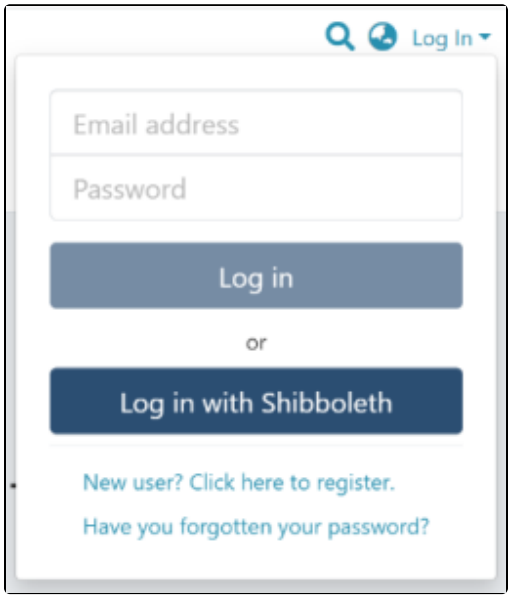
- Audience
- Lease an item during the item submission
- Lease an item via edit item

Audience

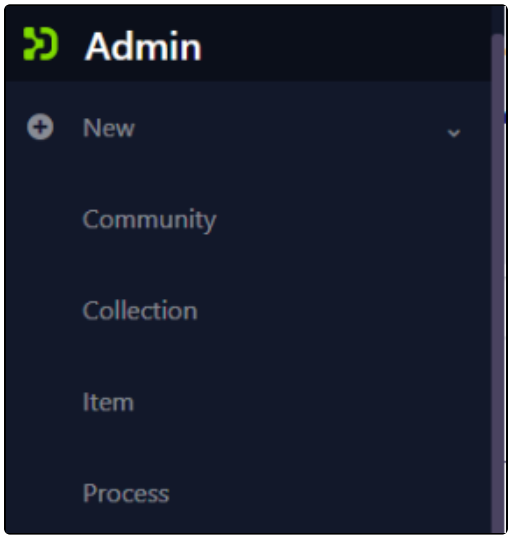
1. Repository Administrator
2. Community Administrator
3. Collection Administrator
4. Item Administrator/submitter

Lease an item during the item submission

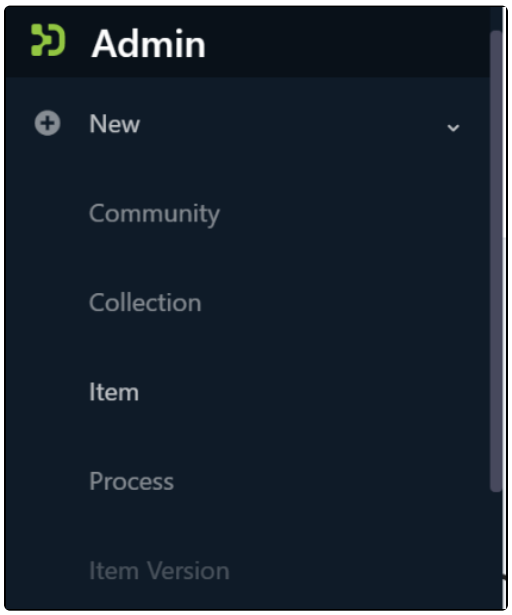
Step 1: Login using your credentials



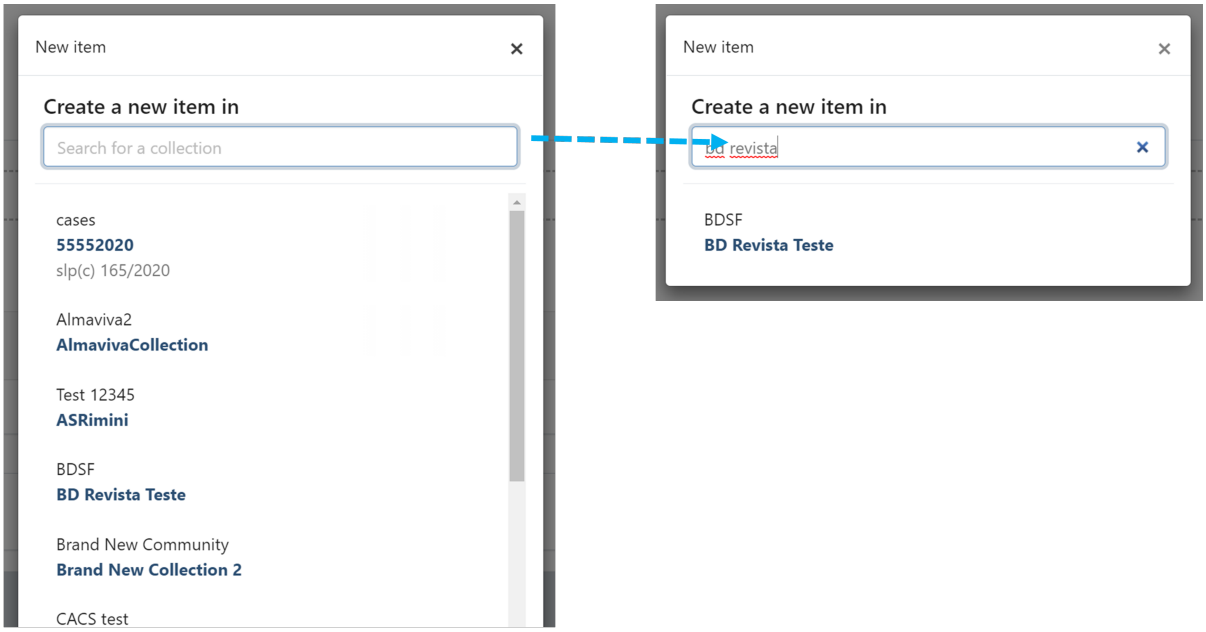
Step 2: Roll over the cursor on the “+” sign.



Step 3: Click on “New” and click on “item” to proceed further in the Item addition process

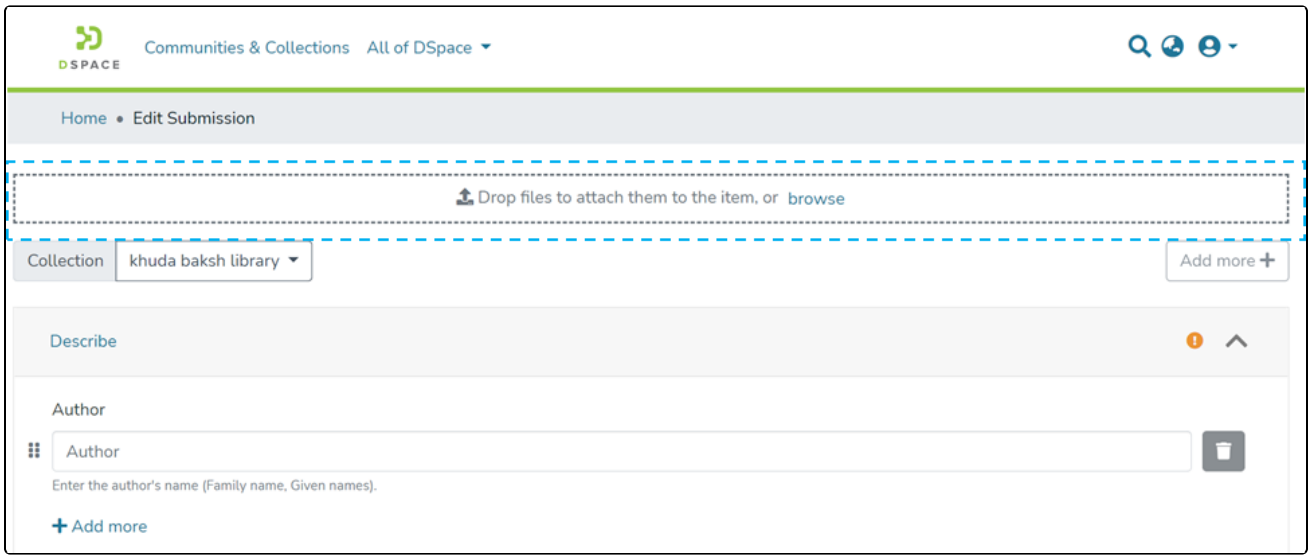


Step 4: A popup window with a collection list will appear. The user can select the target collection by typing its name or scrolling down the collection list. Then, click on the collection to initiate item submission.

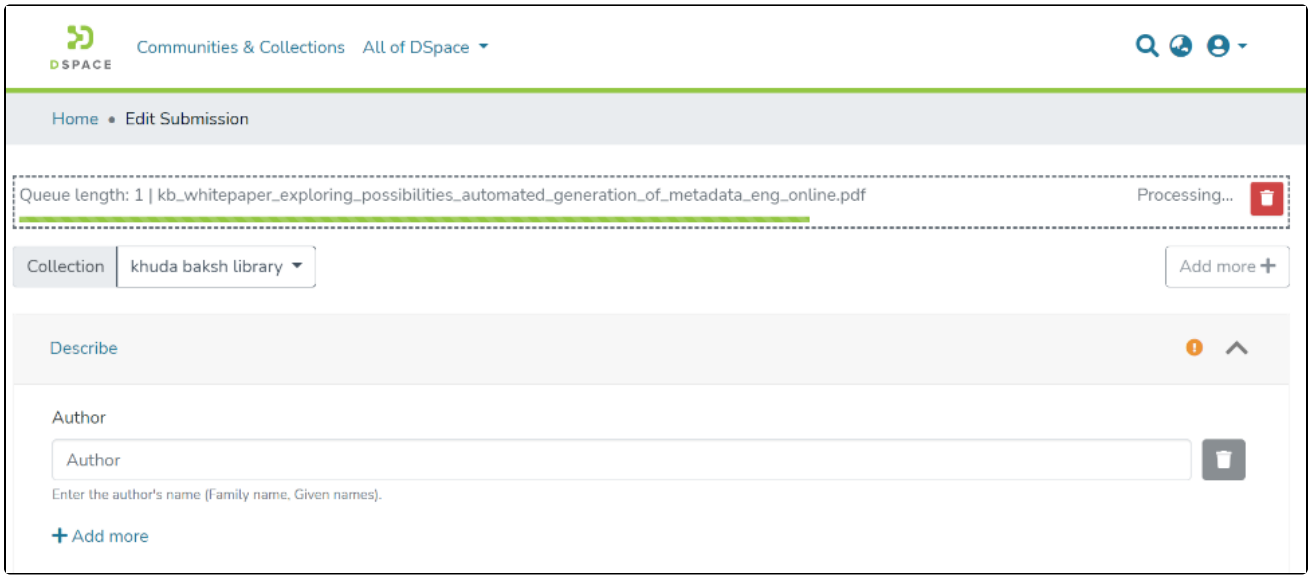


Step 5: Users will see the item submission form after selecting the target collection. The first step is to upload attachment(s) in the Item. In DSpace terminology, an attachment is known as a “bitstream.”

Click on the “browse” link to upload attachment(s). Users can upload multiple files by selecting them together or dragging them into the space.



A progress bar showing bitstream upload progress will appear, as demonstrated in the illustration below. In addition, after a successful bitstream upload, a prompt confirming success or failure will appear.



Bitstream upload in progress

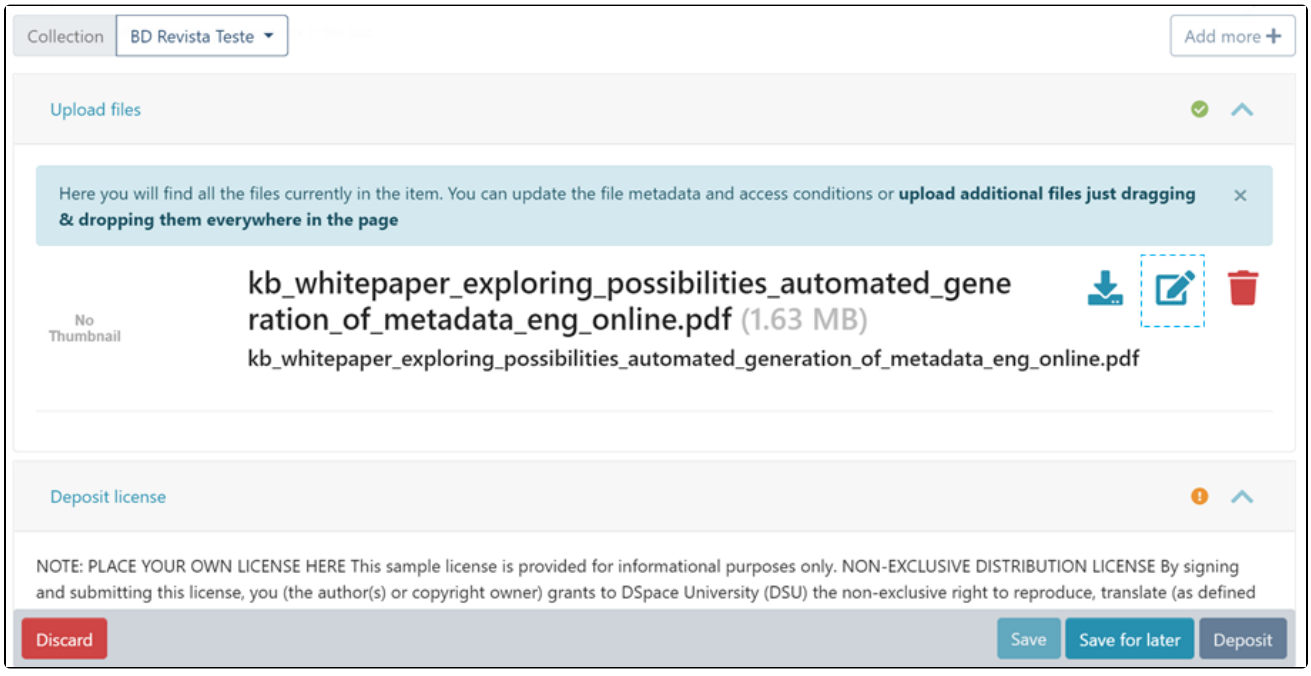


Bitstream Upload Successful

Step 6: After uploading bitstream, the next step is to describe the Item by adding metadata.

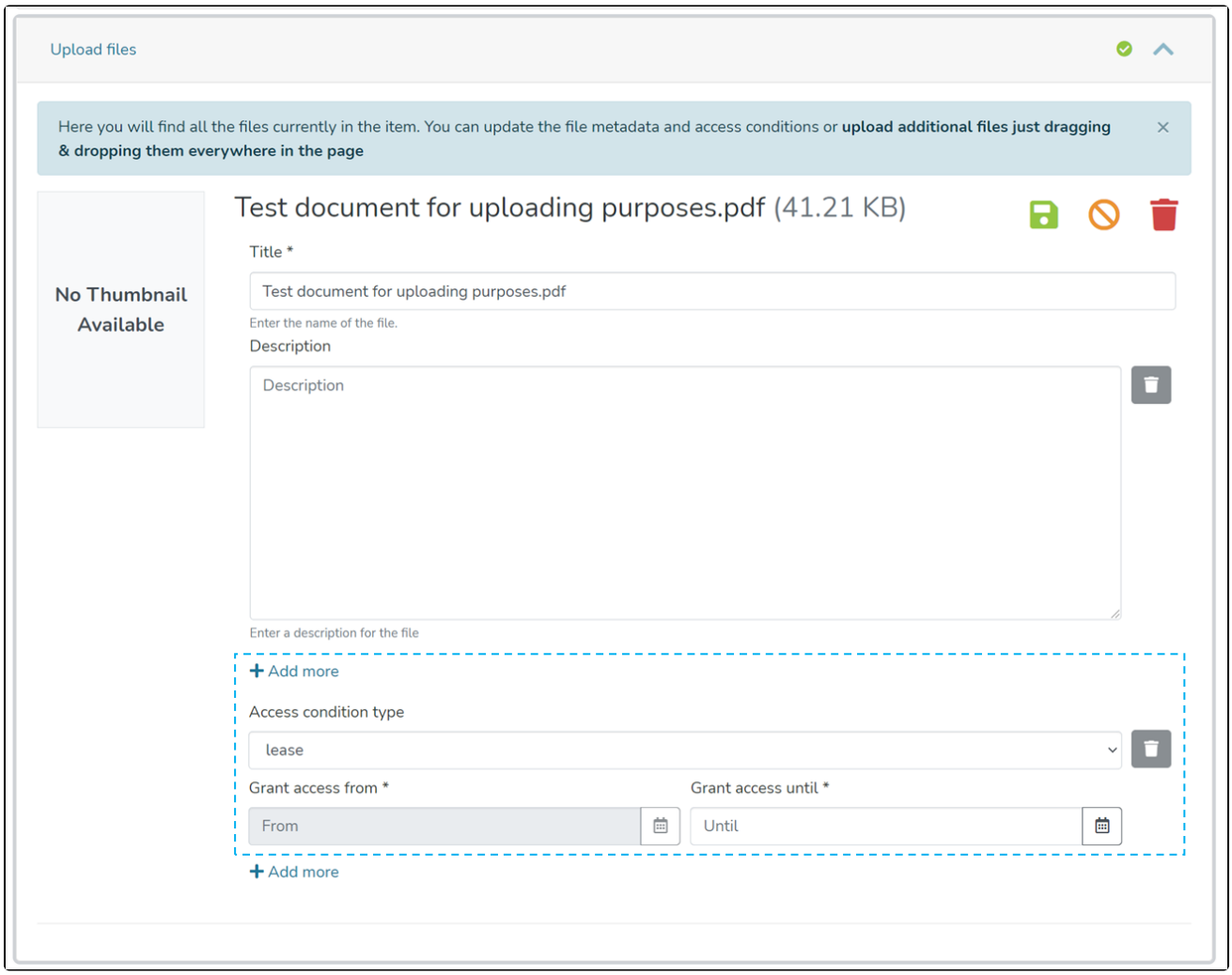
Please refer **Add Item** process for detailed documentation on populating information in the metadata fields.

Step 7: Click on the “edit” button against any attachment to lease it.



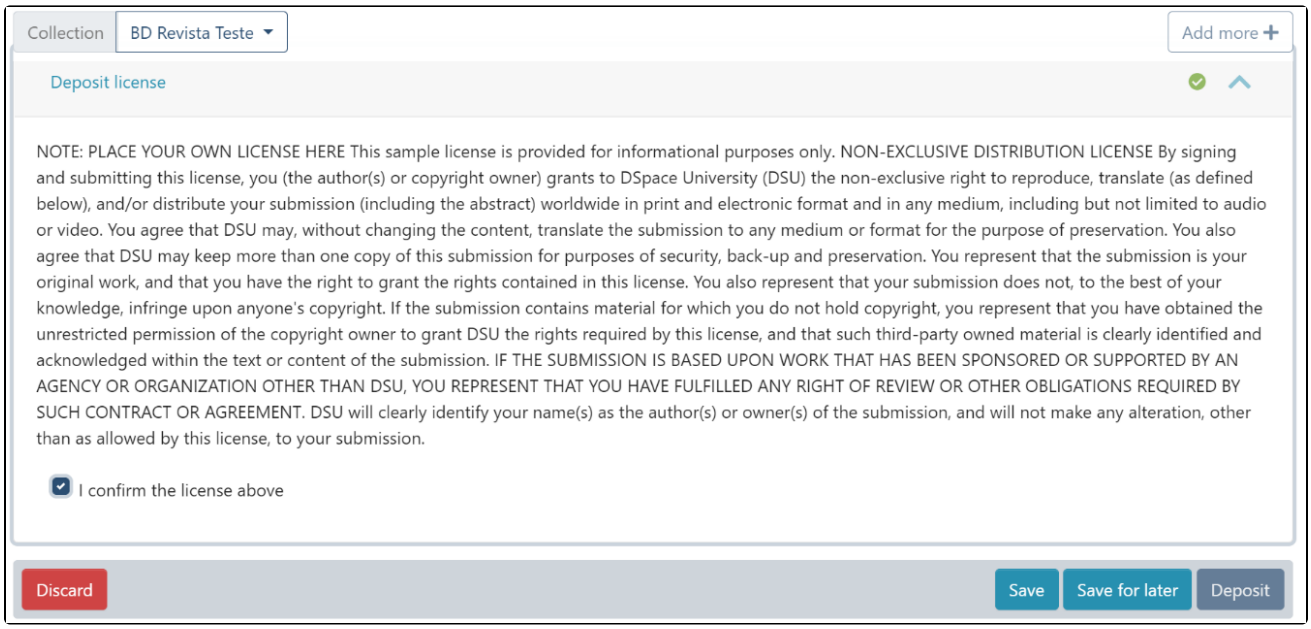
Step 8: Users can apply multiple policies on an attachment from available options. Click on the dropdown list under the “Access condition” and select lease, as highlighted in the screenshot below.

After selecting the lease, the “Grant access until” date field will be activated. Next, choose the future date, after which the attachment should be restricted to the larger set of DSpace users.



Users can add multiple policies to the attachment by clicking the “Add more” link. For example, a user can define a lease on an item until a future date. And after that date, another policy can be defined for the next period.

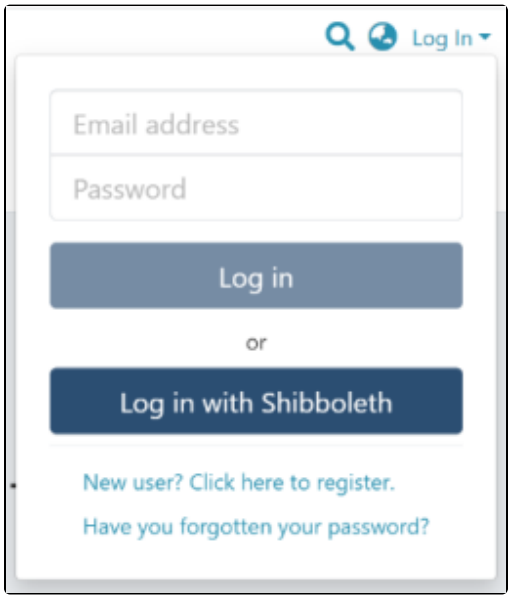
Step 9: After updating all information, the submitter clicks on the “I confirm the license above” checkbox to accept the repository’s license.



Step 10: Click on the “Deposit” button to submit the Item in DSpace. Users will get to see a confirmation prompt upon successful submission of the Item.

Lease an item via edit item

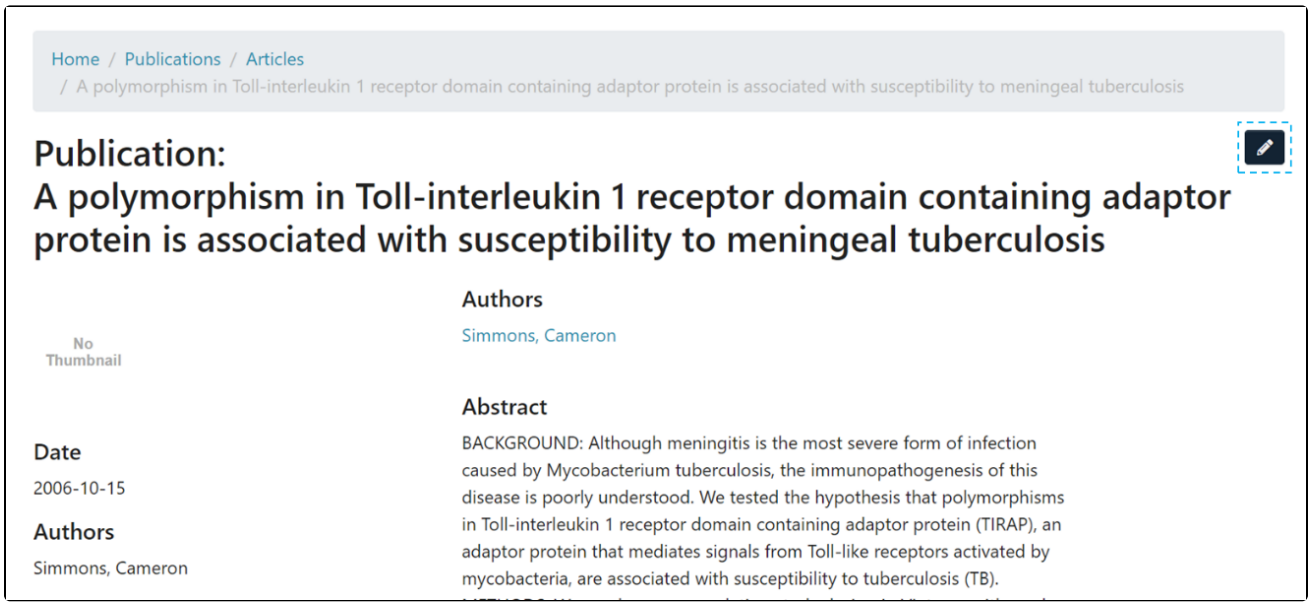
Step 1: Login using your credentials



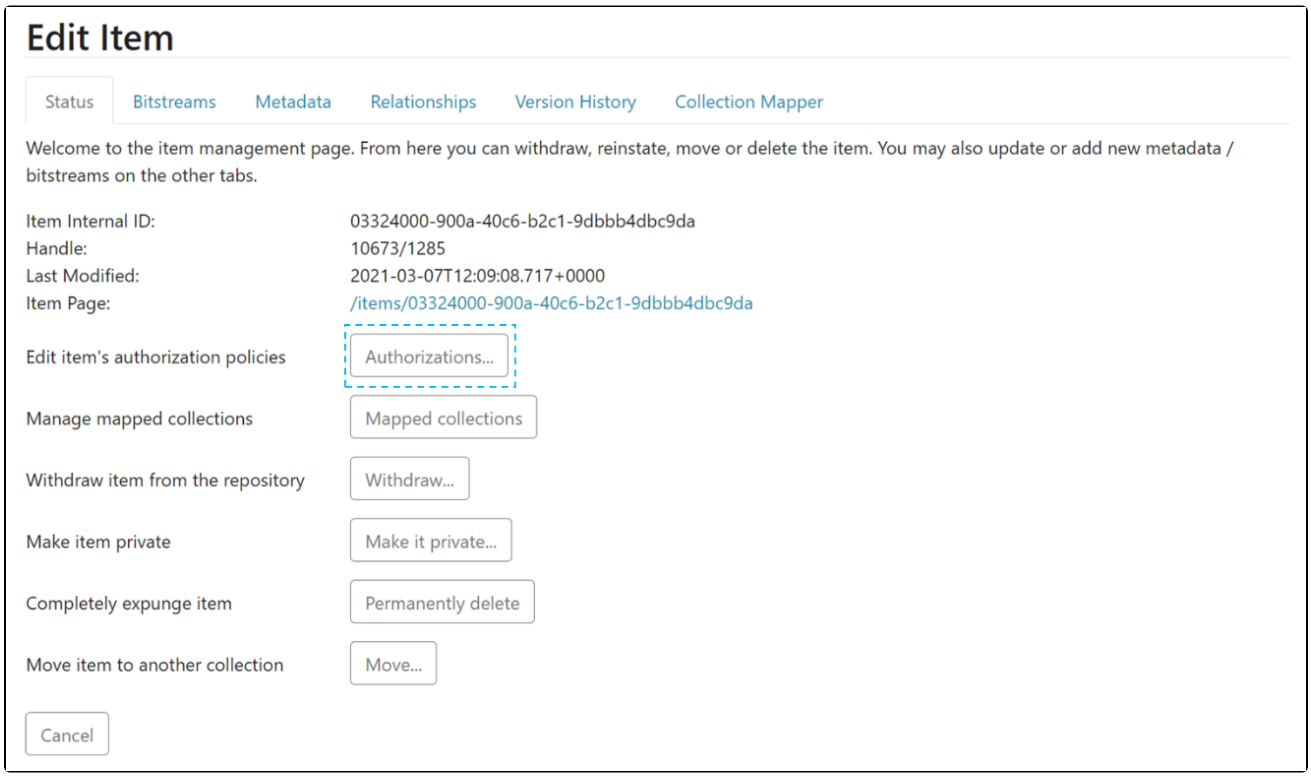
Step 2: Go to the Item you want to edit

Users can reach an item through multiple methods, as listed below:

1. Search an item
2. Browse communities and collections
3. Finding an item in the Administration section at Edit > Item
4. Click the “Edit” button on the right side of the item title.



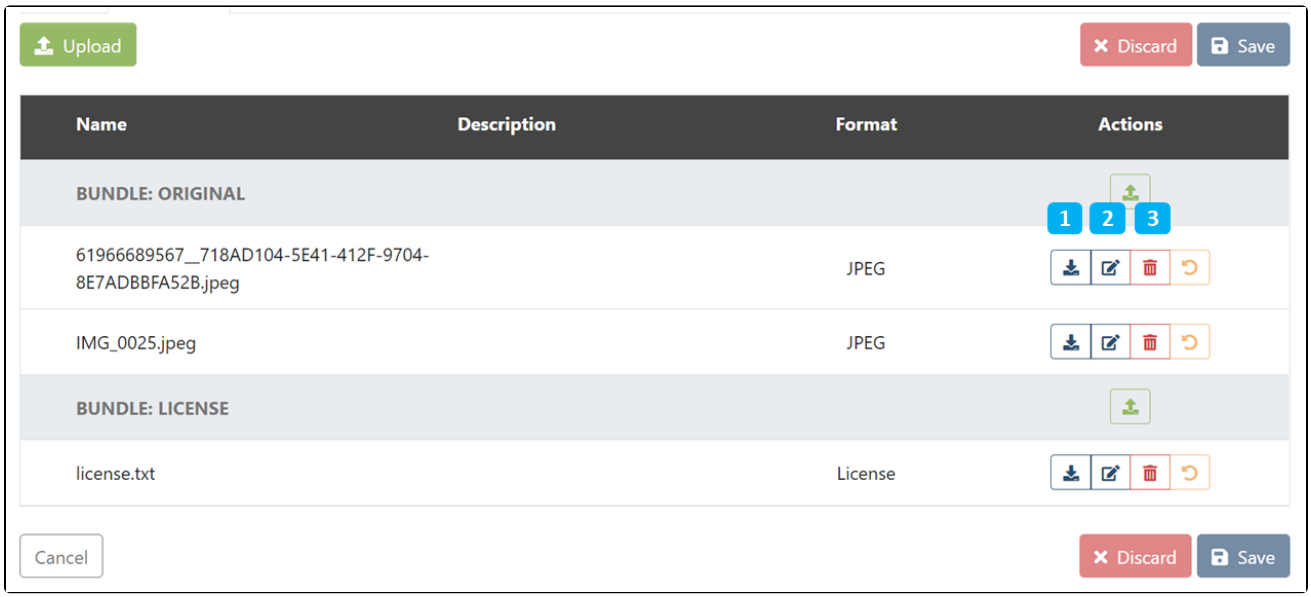
Step 3: Click on the “Authorizations” button under the “Status” tab to continue with leasing attachment(s).



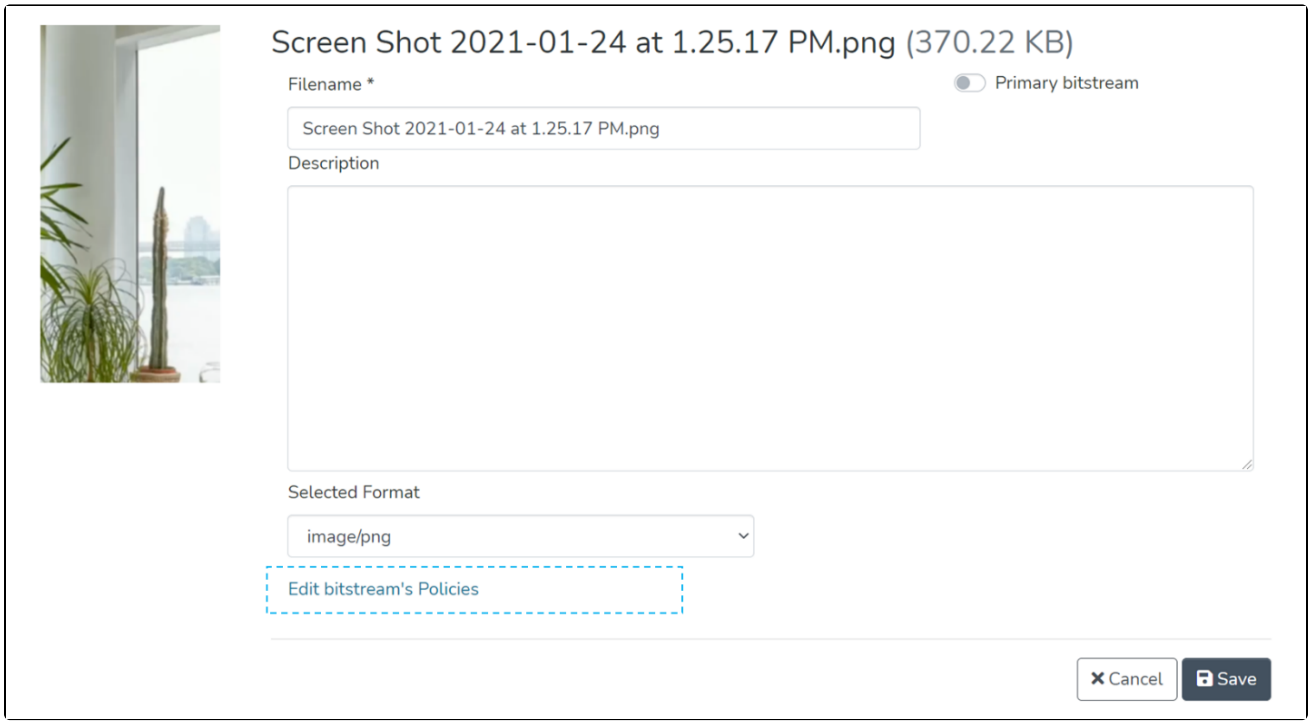
Step 4: The user will see multiple options against each attachment as explained below:

1. **Download Bitstream:** Click on this button to download the attachment on your local device for view.
2. **Edit Bitstream:** Click on the “Edit bitstream” button for editing details. Explained in the next step
3. **Delete Bitstream:** Click on the “Delete Bitstream” button to delete bitstream from the bundle.

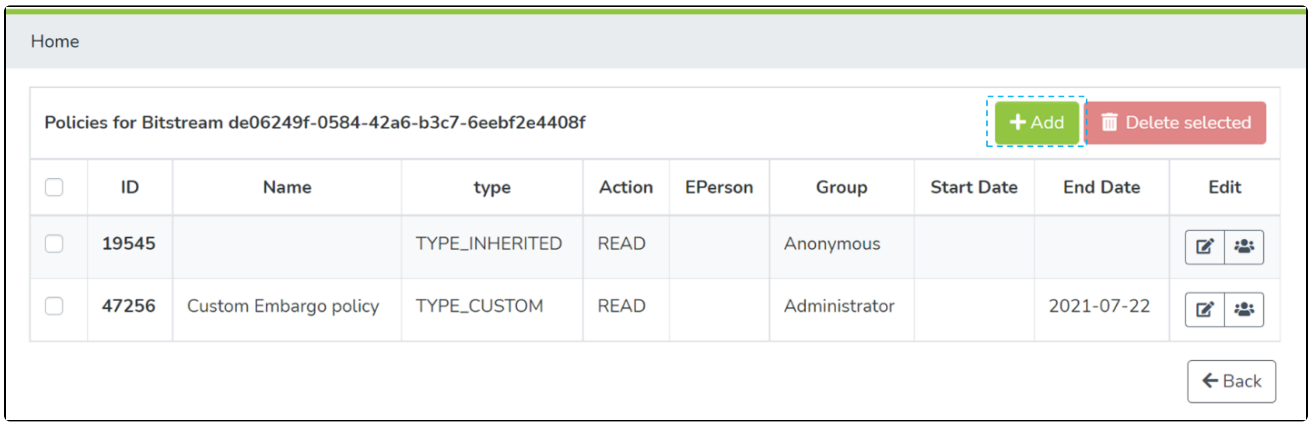
Click on the “edit bitstream” button to lease the attachment.



Step 5: Click on “Edit bitstream’s Policies” to continue the leasing process.

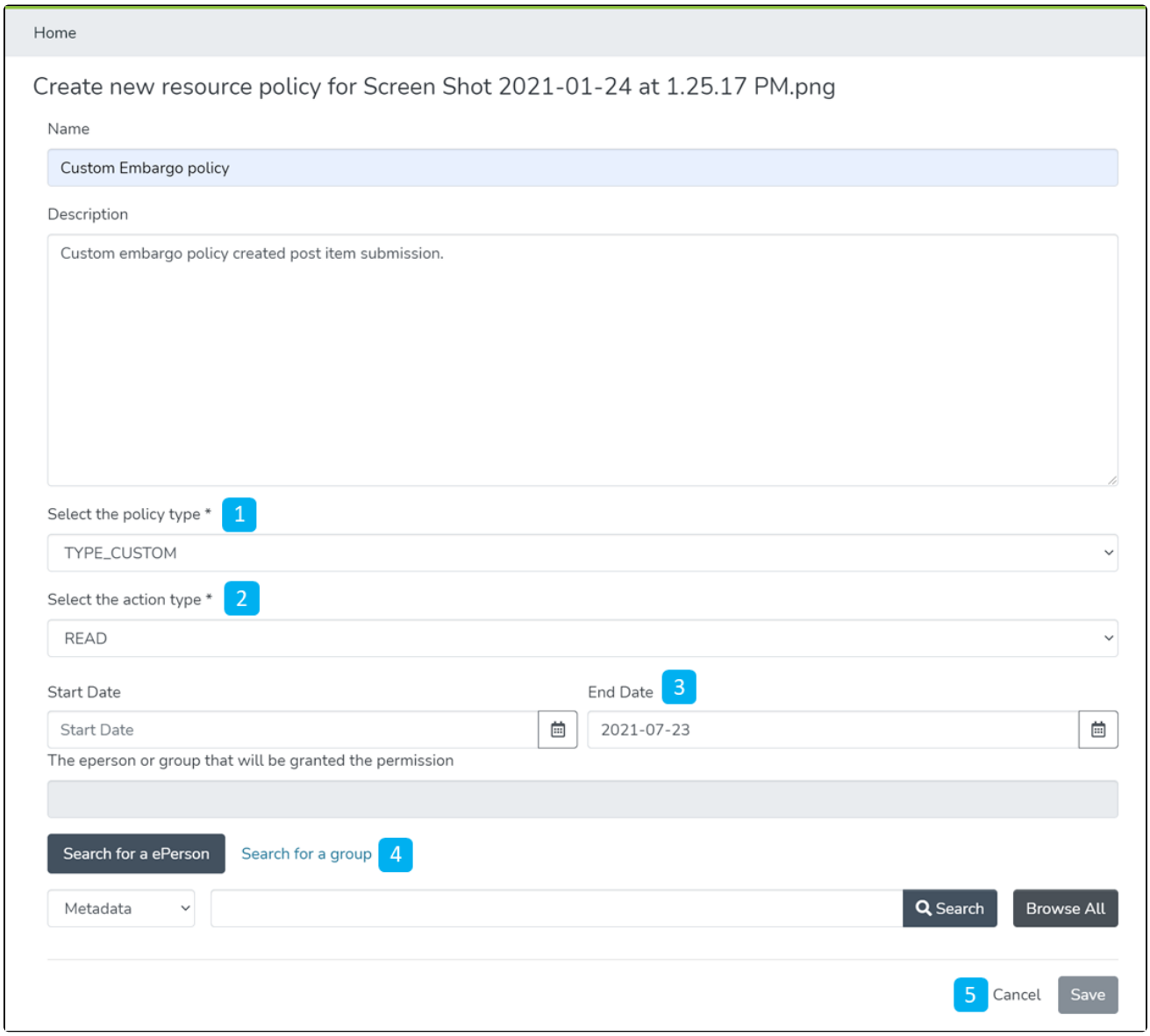


Step 6: Click on the “Add” button to create the custom lease policy for the attachment.



Step 7: Enter details for creating the lease policy on this form and perform the following actions:

1. Select ‘TYPE_CUSTOM’ under ‘Select the policy type’ dropdown menu
2. Select ‘READ’ under the ‘Select the action type’ dropdown list.
3. Select a future date under the End Date to determine when the embargo will end.
4. Select E-Person or a User group
5. Click on Save to create the policy



Detailed documentation on various possibilities on this form is available under ‘Edit Bitstream’ user documentation.

Users will see a success prompt upon creating the policy and be redirected to the bitstream policy page.

DSpace Demo Quick Start

Front end

https://demo.dspace.org

OAI-PMH

https://demo.dspace.org/server/oai/request?verb=Identify

REST-API

https://demo.dspace.org/server/

SWORD v1

https://demo.dspace.org/server/sword/servicedocument

SWORD v2

https://demo.dspace.org/server/swordv2/servicedocument

Management sidebar

Many of the administrative functions can be accessed from the Management sidebar. This list maps the menu to more detailed information.

- New
  - Community
  - Collection
  - Add item
  - Process
- Edit
  - Community
  - Collection
  - Item
- Import
  - Metadata
- Export
  - Metadata
- Notifications
- Access Control
  - People
  - Groups
- Admin Search
- Reports (Beta release)
- Registries
  - Metadata
  - Format Aligner à droite
- Curation Task
- Processes
- Administer Workflow
- Health
- COAR Notify

New

Quickly create or edit objects from anywhere in the system. Either browse to the object first, or search for it using the Admin sidebar. - Release Notes

Community

Collection

Add item

Process

Processes UI (video) allows Administrators to run backend scripts/processes while monitoring their progress & completion. - Release Notes

See Command Line Operations for more detail about these commands.

Edit

- Quickly create or edit objects from anywhere in the system. Either browse to the object first, or search for it using the Admin sidebar.
- Bitstream Editing (video) has a drag-and-drop interface for re-ordering bitstreams and makes adding and editing bitstreams more intuitive.
- Metadata Editing (video) introduces suggest-as-you-type for field name selection of new metadata. - Release Notes

Community

- Edit
- Delete

Collection

- Edit
- Delete

Item

- Edit
- Delete

Import

You can drop or browse CSV files that contain batch metadata operations on files. When "Validate Only" selected, the uploaded CSV will be validated. You will receive a report of detected changes, but no changes will be saved.

Metadata

Export

Metadata

Notifications

The Notifications include the possibility to claim publication suggested by external system, for example ORCID and the Quality assurance. You can find more information in **Quality Assurance**.

Access Control

Login As (Impersonate) another account allows Administrators to debug issues that a specific user is seeing, or do some work on behalf of that user. (Login as an Admin, Click "Access Control" in sidebar, Click "People". Search for the user account & edit it. Click the "Impersonate EPerson" button. You will be authenticated as that user until you click "Stop Impersonating EPerson" in the upper right.) - Release Notes

People

Groups

Admin Search

Administrative Search (video) combines retrieval of withdrawn items and private items, together with a series of quick action buttons. - Release Notes

Reports (Beta release)

Two different kind of reports are available to administrator: **Filtered Collection and Metadata Query. DSpace report are released in DSpace 8 as a Beta release. See DSpace 8 Administrator Reports (Beta Release) for details**

Registries

Metadata

Format Aligner à droite

Curation Task

Processes

Processes UI (video) allows Administrators to run backend scripts/processes while monitoring their progress & completion. - Release Notes

Details about each of the available processes/scripts can be found in the "scripts" directory of the REST API docs: https://github.com/DSpace/RestContract/blob/main/script

Additional information can also be found in the Command Line Operations documentation.

Administer Workflow

Administer Active Workflows (video) allows Administrators to see every submission that is currently in the workflow approval process. - Release Notes

Health

Admin "Health" menu provides basic control panel functionality *(based on 6.x Control Panel).* When logged in as an Administrator, select "Health" from the side menu. You'll see a "Status" tab which provides useful information about the status of the DSpace backend, and an "Info" tab which provides an overview of backend configurations and Java information. - Release Notes

COAR Notify

COAR Notify supports the exchange of linked data notifications across partner organisations and the workflows to manage notifications in those platforms and systems. The application can receive and send LDN messages concerning items with external systems. The LDN system is the protocol of message exchanging; the Quality Assurance system is the mechanism used to approve or reject item updates. See **COAR Notify** for functionality details.

Administrator Reports (Beta feature)

Beta Release

The DSpace Reports are released in DSpace 8.0 as beta feature. This mean some fonctionalities need improvments and they are not completly production ready. The reports are still actively developed. Be aware that in their current state, DSpace reports can cause performance issues. The following issues can help improve the reports:

- [DSpace 8 Admin Reports] Performance issues caused by high number of item loadings. https://github.com/DSpace/dspace-angular/issues/2906
- [DSpace 8 Admin Reports] Performance issues caused by large page sizes. https://github.com/DSpace/dspace-angular/issues/2907
- [DSpace 8 Admin Reports] Move part of the code to a full-featured DataService. https://github.com/DSpace/dspace-angular/issues/2908

The ability to export in CSV as been added and is available in DSpace 9 ([DSpace 8 Admin Reports] Allow Metadata Query report results to be exported in CSV format. https://github.com/DSpace/dspace-angular/issues/2909).

The Beta release of the DSpace reports only provides the ability to run the reports and display the results in the Angular UI. As such, reports feature must be used with caution.

The reports are an Administrator tool. Only users with Administrator permissions can access them.

There are 2 reports:

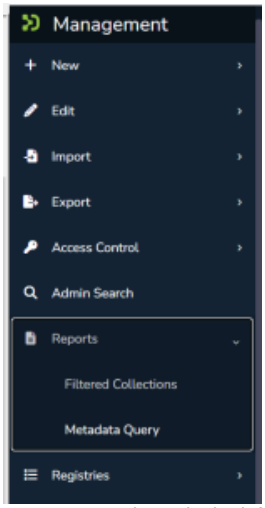
- **Filtered Collection**: this report is counting the numbers of item per community/collection. A large variety of filter can be used (ex: Discoverable Items - Not Private ; Item has No Original Bitstreams, Has unusually large PDF, Has document bitstream without TEXT item, Item Image Bitstreams are Supported, Item has Restricted Original Bitstream etc.). Results table is displayed in the UI.
  The filtered collection reports is documented in DSpace Filtered Collections report endpoint.
- **Metadata Query** : this report build a report based on different main criterias (Collection Selector, Metadata Field Queries, Limit/Paginate Queries, Filters, Additional data to return). Item Results are displayed in the UI
  The metadata query report is documented in DSpace Metadata query (aka Filtered Items) report endpoint.

How to enable

The reports are disabled by default. The reports can be enable or disable in contentreport.cfg

contentreport.enable = true|false	**true** =>the 2 reports will be enabled **false** => the 2 reports will be disabled

Once enable, the reports are available in the admin menu as shown below:



Reports are shown in the left side admin menu, 7th element from the top

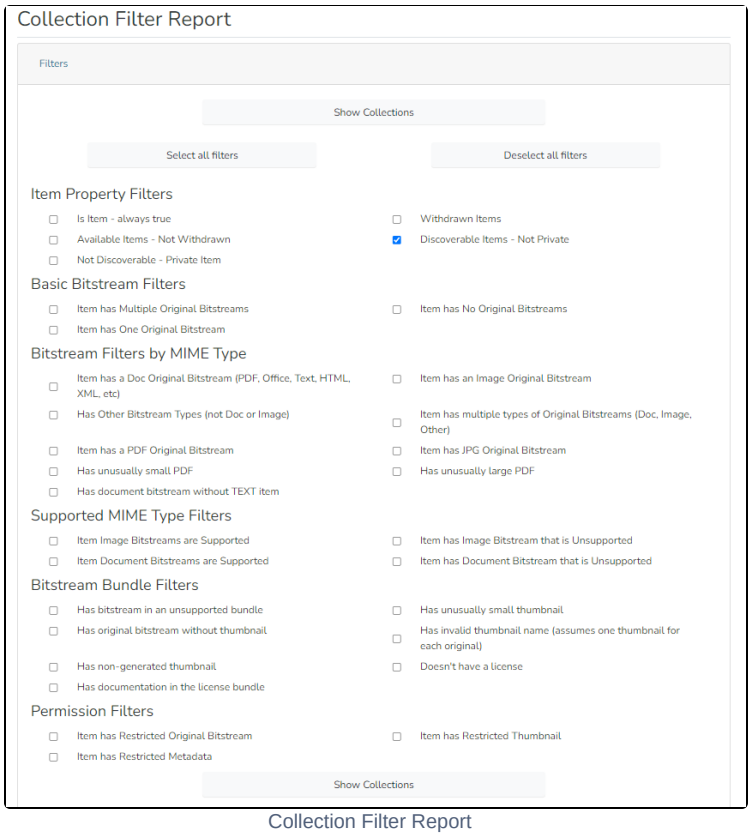
Filtered collection report

The filter collection report will display total number of item in each available collection in DSpace. By default, if no filter are selected, all item are counted, no matter their status (available, withdrawn, non discoverable)

Filter section

The filtered collection report provides 6 categories of filter:

1. Item Property Filters
2. Basic Bitstream Filters
3. Bitstream Filters by MIME Type
4. Supported MIME Type Filters
5. Bitstream Bundle Filters
6. Permission Filters



Collection Filter Report

List of filters and their meaning

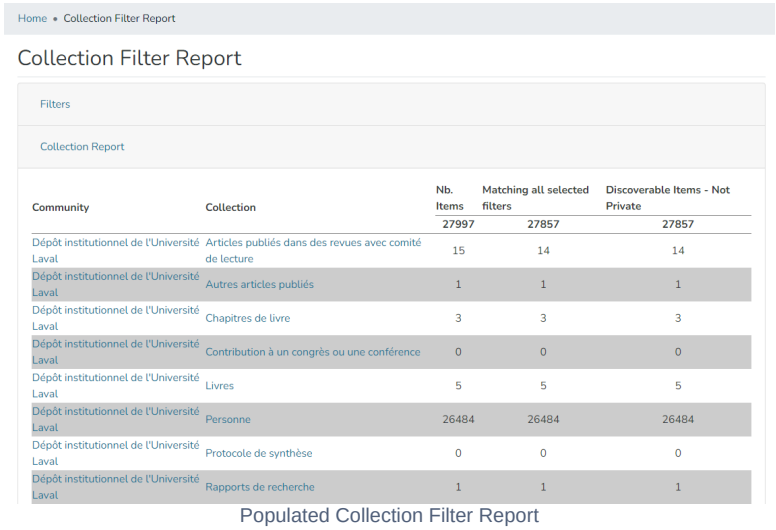
Category	Filter	Expected behavior
Display Filtered collections report page		The report page is displayed. Is it structured in 2 sections : Filters and Collection reports that can be expended or reduced. All filter available are expended by default. A “Show collection” button appears on top and bottom of the Filters Section
Display all item per collections		Count all item no matter their property (public, withdrawn, private, etc.) Results are display in the “Collection report” section in table form. First column is Community ; second Collection ; third total count ; fourth filtered count. Item are shown by communities and collections

Item Property Filters		
	Is Item - always true	The filtered count column only counts items
	Withdrawn Items	The filtered count column only counts the withdrawn items
	Available Items - Not Withdrawn	The filtered count column only counts available items
	Discoverable Items - Not Private	The filtered count column only counts findable items
	Not Discoverable - Private Item	The filtered count column only count private item
Basic Bitstream Filters		
	Item has Multiple Original Bitstreams	Count items that have more than one bitstreams in the ORIGINAL Bundle
	Item has No Original Bitstreams	Count items that have no bitstreams in the ORIGINAL Bundle
	Item has One Original Bitstream	Count items that have only one bitstreams in the ORIGINAL Bundle
Bitstream Filters by MIME Type		
	Item has a Doc Original Bitstream (PDF, Office, Text, HTML, XML, etc)	
	Item has an Image Original Bitstream	
	Has Other Bitstream Types (not Doc or Image)	
	Item has multiple types of Original Bitstreams (Doc, Image, Other)	
	Item has a PDF Original Bitstream	
	Item has JPG Original Bitstream	
	Has unusually small PDF	File must be < 20ko to be considered unusually small.
	Has unusually large PDF	Test with a > 100Mo file
	Has document bitstream without TEXT item”	Count item for which at least one of the documents in the ORIGINAL Bundle as not corresponding file in the TEXT Bundle (contains OCR text file for indexing).
Supported MIME Type Filters		
	Item Image Bitstreams are Supported	
	Item has Image Bitstream that is Unsupported	
	Item Document Bitstreams are Supported	
	Item has Document Bitstream that is Unsupported	
Bitstream Bundle Filters		
	Has bitstream in an unsupported bundle	
	Has unusually small thumbnail	
	Has original bitstream without thumbnail	
	Has invalid thumbnail name (assumes one thumbnail for each original)	
	Has non-generated thumbnail	
	Doesn't have a license	Count Items that do not have a LICENSE Bundle

	Has documentation in the license bundle	
Permission Filters		
	Check “Item has Restricted Original Bitstream”	
	Check “Item has Restricted Thumbnail”	
	Check “Item has Restricted Metadata”	

Filtered collection results

The results will always give the total count (without any filters applied) and the filtered count.



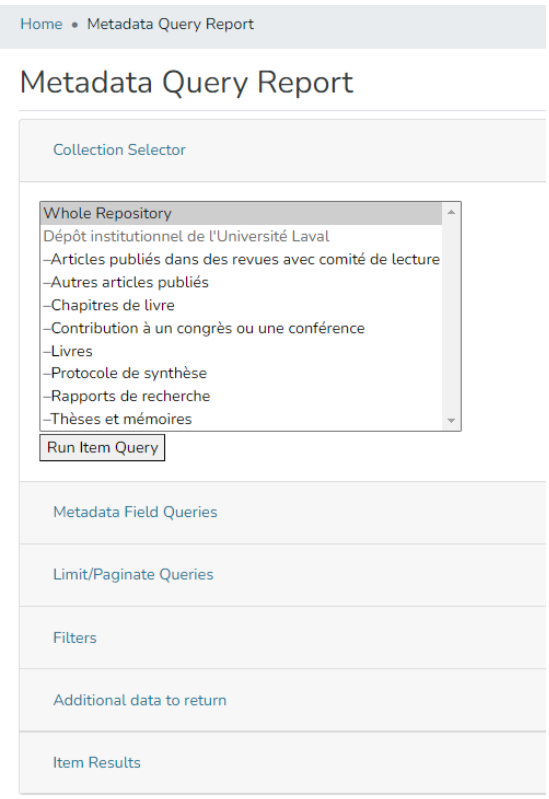
Populated Collection Filter Report

Metadata Query Report

The metadata query offer the capability to search for items using a variety of operators, conditions and filters.

The report contains 6 sections, each of which is a step in building a query:

1. Collection selector : select one or many collections
2. Metadata fields query: where to build your query using predefine query or custom query.
3. Limit/Paginate Query: select the number of results per page (optional)
4. Use Filters (optional): they are exactly the same that those of the Filtered Collection report.
5. Additional metadata: select additional metadata to return. By default, only UUI, Collection URI and Title are returned.
6. Item Results : the section where the results will be displayed.



Metadata Query Report

Queries documentation

Category	Description	Expected behavior
Collection selector	Running a request with no queries and filters selecting individual collection	List all items of the selected collection
	Running a request with no queries and filters selecting multiple collection	List all items of the selected collections
	Running a request with no queries and filters selecting Whole repository	List all items
Metadata fields predefined queries	Has No Title (use “does not exist” operator)	
	Has No dc.identifier.uri (use “does not exist” operator)	
	Has c dc.contributor.author	Must contain the “and” operator (Smith, John and Doe, John)
	Has compound dc.creator	Must contain the “and” operator (Smith, John and Doe, John)
	Has URL in dc.description (use “matches” operator)	
	Has unbreaking metadata in description	
	Has full text in dc.description.provenance	
	Has non-full text in dc.description.provenance	
	Has non-ascii character in metadata	

	Has empty metadata	
	Has XML entity in metadata (use “matches” operator)	
New Query Report		
	Exists	
	does not exist	
	equals	Need exact, complete character string
	does not equal	Need exact, complete character string
	Like (permet d’ajouter des troncature ex % (0, 1 ou plusieurs caractères) ou _ (un seul caractère)	
	not like	
	contains	
	does not contain	
	matches (permet de mettre des expressions régulières dans la requête – regex ou regexp)	
	Does not match	
	Multi-field queries	
Limit/Paginate Queries	Limit to 100 then 25	First 100 items are shown, then 25.
Addition data to return	Select 3 additional metadata	Appears in results
Item Results	Run a query	Default results display UUID Collection, URI and Title

Metadata Query Results

By default, only UUI, Collection URI and Title are included in the results. Additional metadata can be added when building the query.

Metadata query export

Export in CSV is available in DSpace 9. Results can be exported using the export button at the end of the result page. The export trigger a standard process that can be seen in the Process page (admin menu)

COAR Notify



notify COAR logo

The COAR Notify Initiative is developing and accelerating community adoption of a standard, interoperable, and decentralised approach to linking research outputs hosted in the distributed network of repositories with resources from external services such as overlay-journals and open peer review services, using linked data notifications. As part of this project, COAR is funding the development of platforms and systems to support the exchange of linked data notifications across partner organisations and the workflows to manage notifications in those platforms and systems.

The application can receive and send LDN (Linked Data Notifications) messages concerning items with external systems. The LDN system is the protocol of message exchanging; the Quality Assurance system is the mechanism used to approve or reject item updates.

Enable COAR Notify

To enable COAR Notify, you MUST first enable the following:

- Quality Assurance

In addition, the following settings MUST be added to your local.cfg:

- ldn.enabled = true
- Add the "ldnmessage" consumer to the list of consumers in "event.dispatcher.default.consumers"

Available COAR Notify configurations:

ldn.enabled = true|false (REQUIRED)	**true** => message is received and managed the server responds with an HTTP 202 code **false** => message is refused and the server responds with an HTTP 404 code
event.dispatcher.default.consumers (REQUIRED)	Add the "ldnmessage" consumer to this list of default consumers. This consumer is used to store LDN (Linked Data Notification) messages.
ldn.notify.inbox =${dspace.server.url} /ldn/inbox	Where the ldn inbox rest service is mapped on the current DSpace instance. Default is *${dspace.server.url} /ldn/inbox*
ldn.ip-range.enabled = true|false	enables the validation against the IP of received ldn message against the registered range
ldn.template.path = ${dspace.dir} /config/ldn	restricts LDN message templates to the configured path

COAR Notify Support page in User Interface

By default, when COAR Notify is enabled, the COAR Notify logo will appear in the footer of the site. Clicking that logo will bring you to a basic Support Page (/info/coar-notify-support) which provides details for how other systems may send notifications to your DSpace.

If you wish to disable the COAR Notify logo & Support page from appearing, that can be done in the following User Interface configuration (in your "config.*. yml"):

config.*.yml

```
# When set to "true", the COAR Notify logo will appear in the footer linking to the Support page.
# When set to "false", the COAR Notify logo will not appear, and the Support page will return a 404.
# NOTE: This setting only impacts the logo & support page. If you have enabled COAR Notify,
# it will still function even when this logo / support page is not displayed.
info:
  enableCOARNotifySupport: true
```

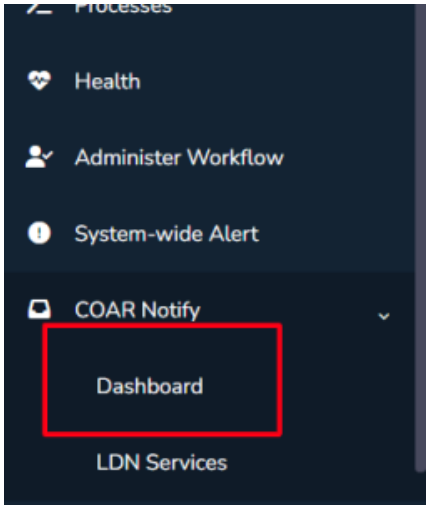
COAR Notify - Dashboard

- Introduction
- Dashboard tabs
  - Metrics
    - Meaning of each box
      - Accepted
      - Processed LDN
      - Failure
      - Untrusted
      - Delivered
      - Queued
      - Queued for retry
      - Failure
      - Involved Items
    - Boxes configuration
      - REST (Discovery configuration)
      - Angular
    - User Interface
  - Logs Inbound Tab
    - Why a LDN notification should be reprocessed?
  - Logs Outbound Tab
    - Why a LDN notification should be reprocessed?

Introduction

To complement the developments for the COAR Notify protocol COAR Notify Documentation, the Notify Administrative Dashboard has been implemented to monitor the general usage of the COAR Notify protocol across the repository.

The dashboard is accessible to repository administrators only via the menu voice **COAR Notify** > **Dashboard.**



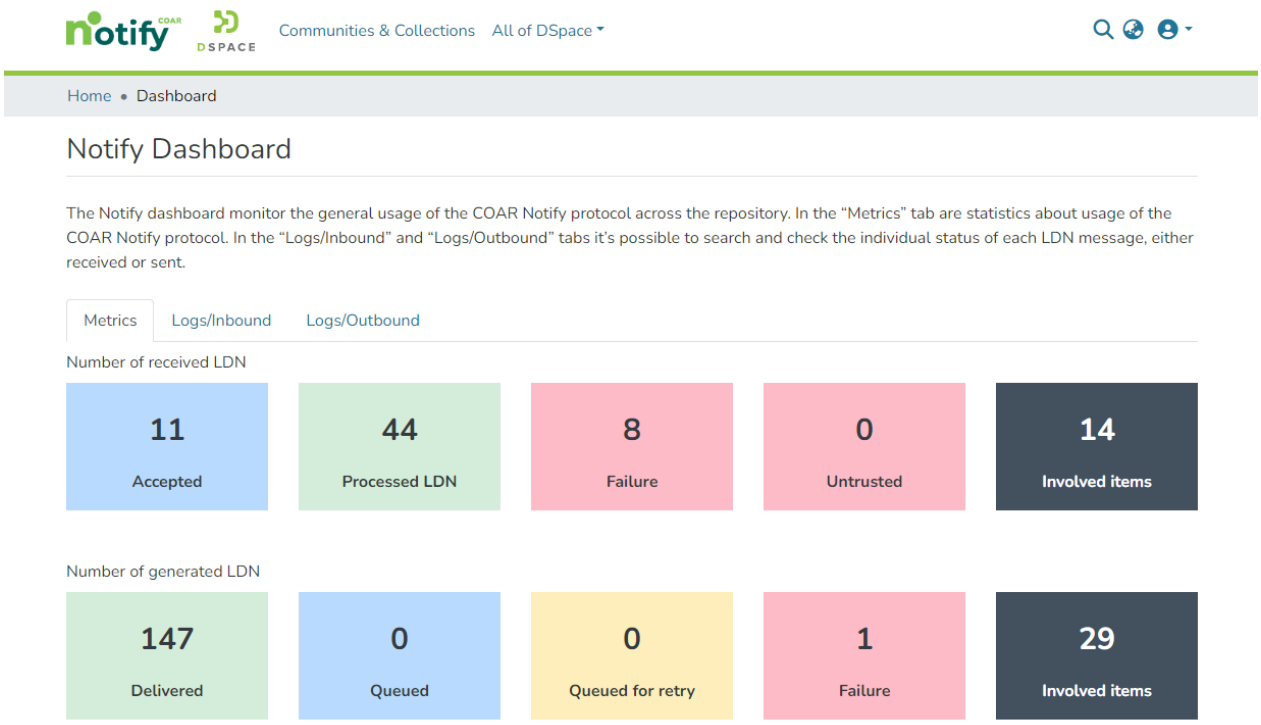
The dashboard is organised in three tabs:

- Metrics
- Logs/Inbound
- Logs/Outbound

Dashboard tabs

Metrics

The “Metrics” tab displays data about usage of the COAR Notify protocol, showing the number of received (accepted/processed/failure/untrusted) and generated (delivered/queued/queued for retry/failure) LDNs, and the total number of items involved.



Any LDN Message is considered in this view as long as the Notification is stored.

Meaning of each box

Each box describes a different status the notifications are

Accepted

This box displays the number of LDN received as an **acceptance** to a previous request sent from the repository.

This means We have received an **acceptance notification** from the external service (Notice this is not the same as receiving the Review/Endorsement or any other object)
so the external service has accepted our previous notification an has replied with a confirmation to the LDN.

When an **Acceptance** LDN is received as reply to a previously sent notification this counter is increased.

Processed LDN

This box displays the number of LDN received and **processed correctly** (acceptance notifications are included as well since acceptance might be defined as a subset of processed notification)

Notifications in this status have been correctly processed and a corresponding action was triggered

Failure

This box displays the number of LDN received but **not correctly processed**.

This status includes **QUEUE_STATUS_FAILED** and **QUEUE_STATUS_UNMAPPED_ACTION**. So any failing notification is reported to be reviewed by the user

Untrusted

This box displays the number of LDN received but **not even processed**.

Any notification in this status has not even been evaluated since the **Service was not recognized** or the **IP range was not respected**

Delivered

This box displays the number of **LDN Sent without any error**.

Any notification successfully delivered will increase this counter

Queued

This box displays the number of **LDN waiting to be sent** (LDN is still queued)

Since LDN are not immediately sent to the external service this status is displaying how many items are still queued

Queued for retry

This box displays the number of **LDN waiting for reprocessing**

This means the **failing (outgoing)** notification has been re-queued to be reprocessed

Failure

This box displays the number of **Failing LDN**

Generally in this case the **external service didn't receive** the notification (an error code is expected when delivering the LDN)

Involved Items

These boxes are duplicated for each row for **Incoming** and **Outgoing** LDN.

They display the total number of items in the repository involved in the workflow process for COAR NOTIFY.

In few words these boxes display the number of items a LDN was receive/generated for

Boxes configuration

REST (Discovery configuration)

Each box is related to a different discovery configuration as mentioned above

- **IdnMessageEntityBaseConfig** is the "base" configuration for all the different LDN discovery config.
  This configuration mainly defines the **search filter** and **facets** used by any discovery config.
- **NOTIFY.incoming.accepted** is the configuration for the "**Accepted**" box
- **NOTIFY.incoming.processed** is the configuration for the "**Processed LDN**" box
- **NOTIFY.incoming.failure** is the configuration for the "**Failure (for incoming)**" box
- **NOTIFY.incoming.untrusted** is the configuration for the "**Untrusted**" box
- **NOTIFY.incoming.involvedItems** is the configuration for the "**Involved items (for incoming)**" box
- **NOTIFY.outgoing.delivered** is the configuration for the "**Delivered**" box
- **NOTIFY.outgoing.queued** is the configuration for the "**Queued**" box
- **NOTIFY.outgoing.queued_for_retry** is the configuration for the "**Queued for retry**" box
- **NOTIFY.outgoing.failure** is the configuration for the "**Failure (for outgoing)**" box
- **NOTIFY.outgoing.involvedItems** is the configuration for the "**Involved items)for outgoing)**" box

Angular

On the angular application we can decide which box has to be displayed, the text color, the background color and the title for each one

The angular configuration involved is found in **default-app-config.ts**

Here's an example of the angular configuration

```
notifyMetrics: AdminNotifyMetricsRow[] = [

 {
  title: "admin-notify-dashboard.received-ldn",
  boxes: [
   {
    color: "#B8DAFF",
    title: "admin-notify-dashboard.NOTIFY.incoming.accepted",
    config: "NOTIFY.incoming.accepted",
    description: "admin-notify-dashboard.NOTIFY.incoming.accepted.description",
   },


   ........


   {
    color: "#43515F",
    title: "admin-notify-dashboard.NOTIFY.incoming.involvedItems",
    textColor: "#fff",
    config: "NOTIFY.incoming.involvedItems",
    description: "admin-notify-dashboard.NOTIFY.incoming.involvedItems.description",
   },
  ],
 },
 {
  title: "admin-notify-dashboard.generated-ldn",
  boxes: [
   {
    color: "#D4EDDA",
    title: "admin-notify-dashboard.NOTIFY.outgoing.delivered",
    config: "NOTIFY.outgoing.delivered",
    description: "admin-notify-dashboard.NOTIFY.outgoing.delivered.description",
   },


   ......


   {
    color: "#FDBBC7",
    title: "admin-notify-dashboard.NOTIFY.outgoing.failure",
    config: "NOTIFY.outgoing.failure",
    description: "admin-notify-dashboard.NOTIFY.outgoing.failure.description",
   },
   {
    color: "#43515F",
    title: "admin-notify-dashboard.NOTIFY.outgoing.involvedItems",
    textColor: "#fff",
    config: "NOTIFY.outgoing.involvedItems",
    description: "admin-notify-dashboard.NOTIFY.outgoing.involvedItems.description",
   },
  ],
 },
];
```

The above layout configuration allows to configure the whole **Dashboard/Metrics Tab**

Using **AdminNotifyMetricsRow** we are allowed to define a **custom number of rows** (The default is two)

Each **Row** has **title** and **boxes** elements

- **title** is the title being displayed on the row
- **boxes** is the list of boxes in the current row

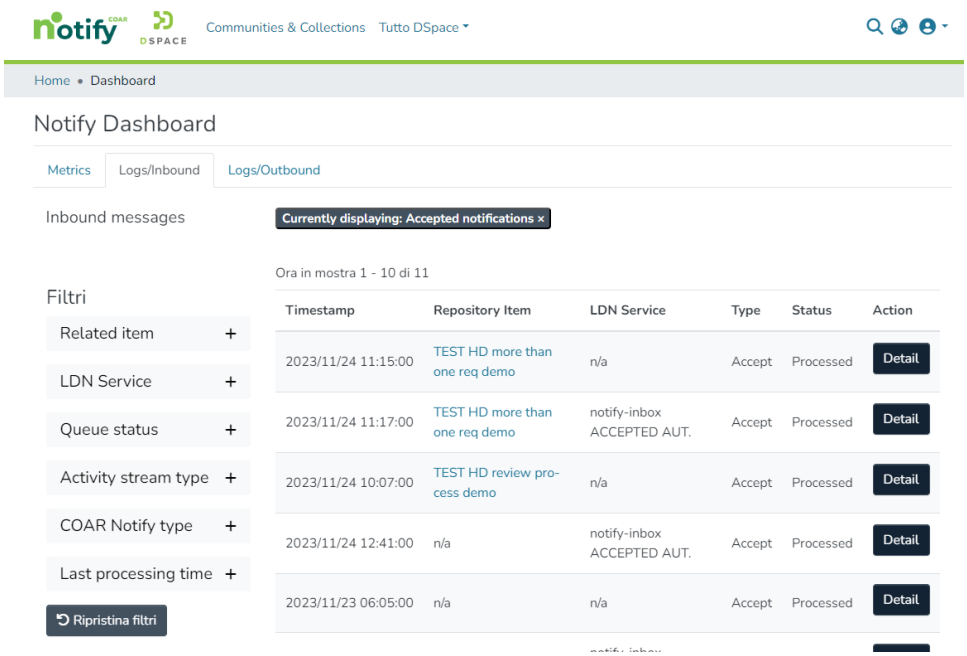
Each **Box** object has the following mandatory fields: **color**, **title**, **config** and **description** and one optional parameter **textColor**

- **color** is the background color for the box (as **HEX color code**)
- **title** is the text displayed on the box (as key for the label)
- **config** is the discovery configuration to use
- **description** is the tool-tip text displayed (as key for the label)
- **textColor** is the font color used for the title of the box (as **HEX color code**)

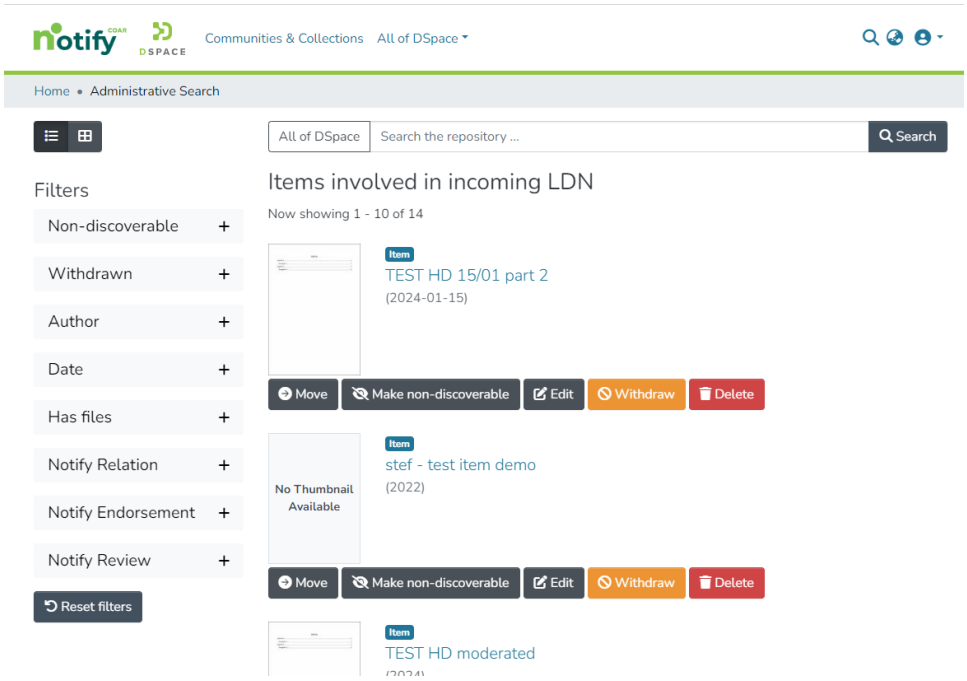
User Interface

Each colored box is clickable and behind each box there's a different discovery configuration:

- **Statuses Boxes** will redirect the user to the corresponding log's tab (either **inbound** or **outbound**), showing the LDN messages filtered by the corresponding selected criteria



- **Involved item boxes** will redirect the user to the **Administrative Search page**, where only the involved items, either by incoming or outgoing LDN messages, will be shown in the results list.
  Items can than be further filtered also for any COAR Notify metadata (**notify.relation.endorsedBy**, **datacite.relation.isReviewedBy**, and **datacite.relation.isReferencedBy**) using the facets.



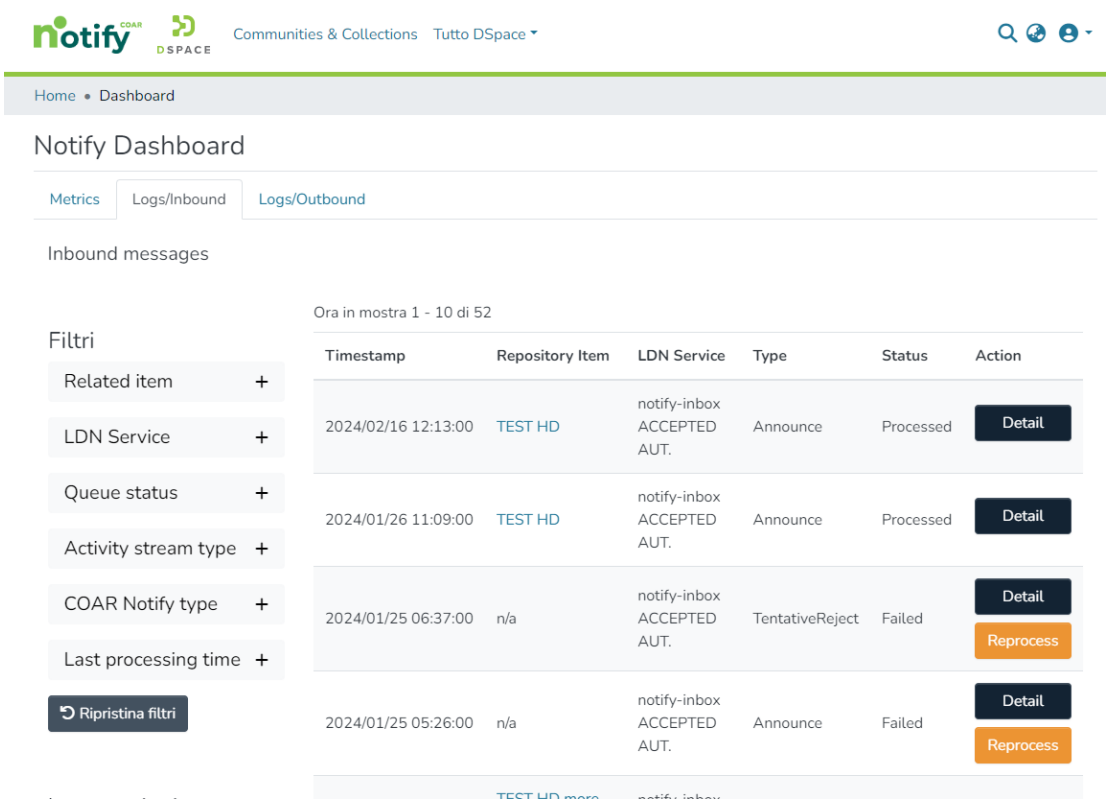
Logs Inbound Tab

In the "Logs/Inbound" tab it's possible to search and check the individual status of each LDN message received by the repository.

Logs are listed in a table displaying the following columns:

- Timestamp
- Repository item
- LDN Service
- Type

- Status
- Action

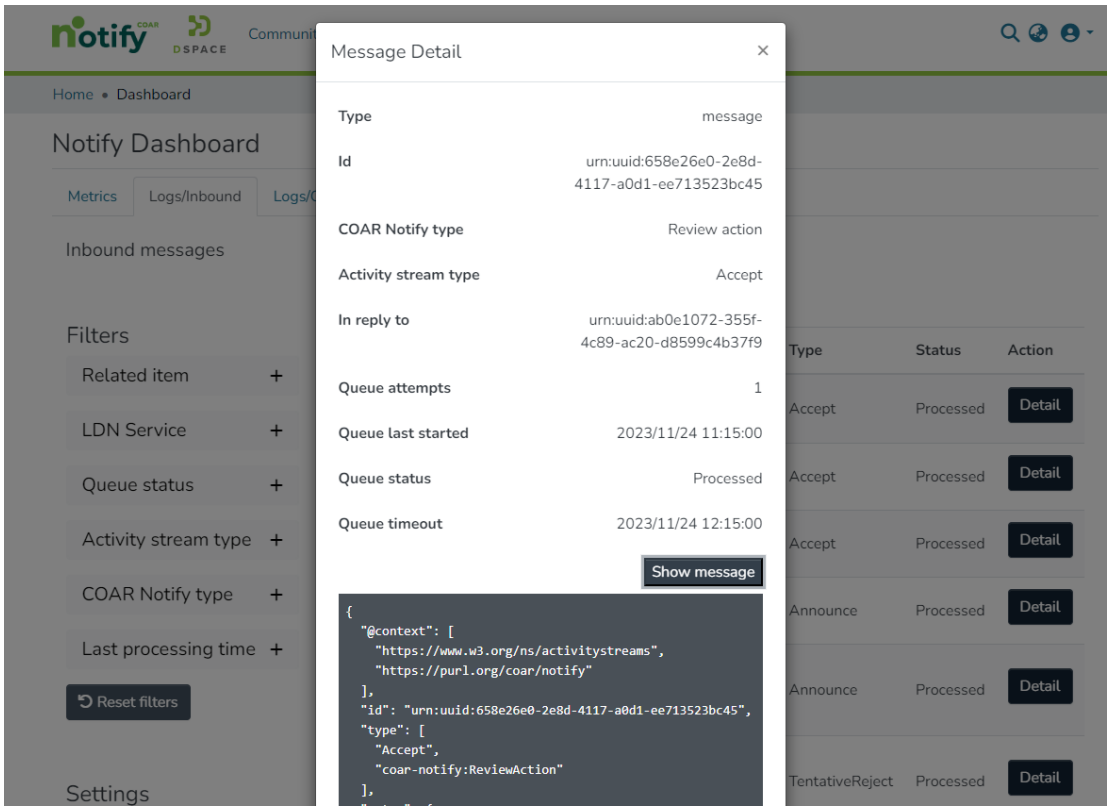


Logs can be scrolled and/or filtered by:

- Related item
- LDN Service
- Queue status
- Activity stream type
- COAR Notify type
- Last processing time

In the logs' table, the value in the “Repository Item” column links to the involved item in the repository.

By clicking on the 'detail' button corresponding to each log, a modal will open displaying the technical information of the received LDN message and it is also be possible to inspect the JSON notification which might be relevant for some purposes.



In case of messages with a queue status equal to “**untrusted**” and “**unmapped action**”, a second button ”Reprocess” will be displayed.

By clicking this button, a request to reprocess the LDN message is triggered.
This allows administrators to reprocess, for instance, messages received from LDN services that have not yet been registered (once the proper mapping has been added on the related spring config file),
or notification which hadn't been correctly processed the first time.

In case of deleted items, LDN services, etc., “n/a” will be shown in the logs' table in the corresponding "Repository Item" coloumn.

Logs with “Unmapped Action” status don’t display a timestamp, as they have not been processed.

Why a LDN notification should be reprocessed?

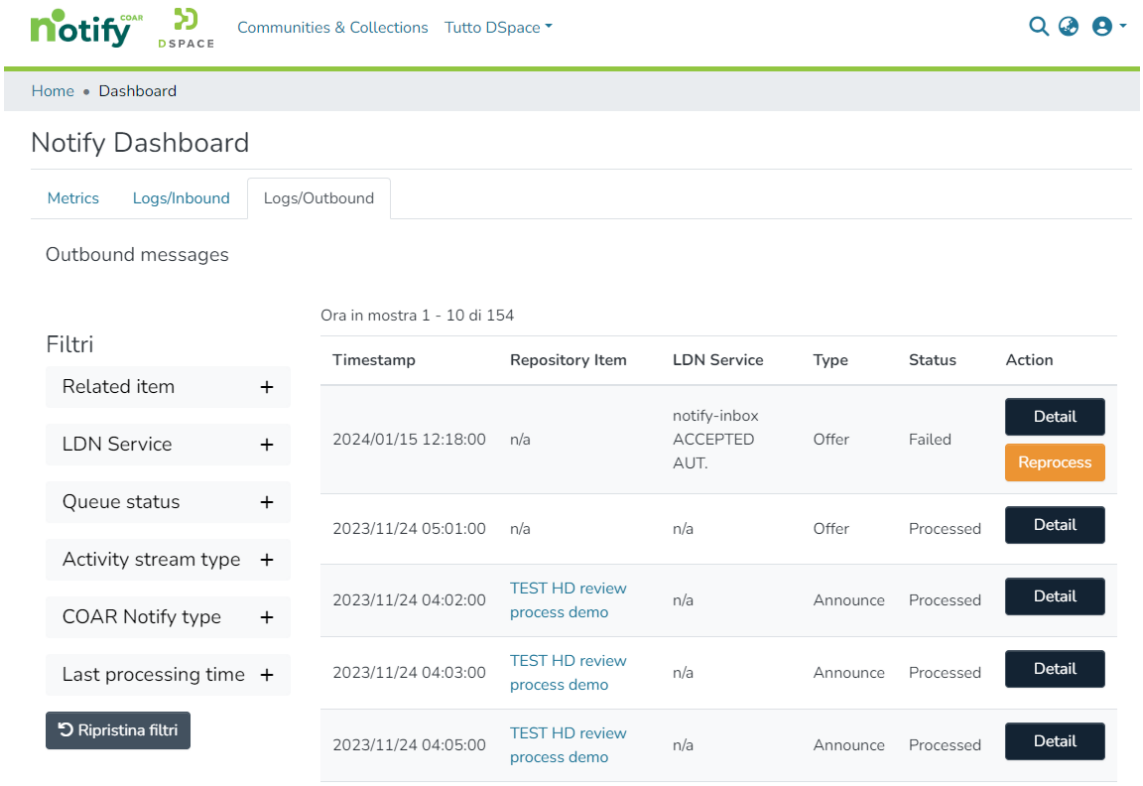
- The external service might have changed the IP and it doesn't match any longer with the rage provided on the LDN Service
- The LDN Notification had an unsupported type an this led to an **unmapped action** status.
  In this case we should configure a new mapping for the missing type before reprocessing the notification

Logs Outbound Tab

In the “Logs/Outbound” tab it’s possible to search and check the individual status of each LDN message received by the repository.

Logs are listed in a table displaying the following columns:

- Timestamp
- Repository item
- LDN Service
- Type
- Status
- Action

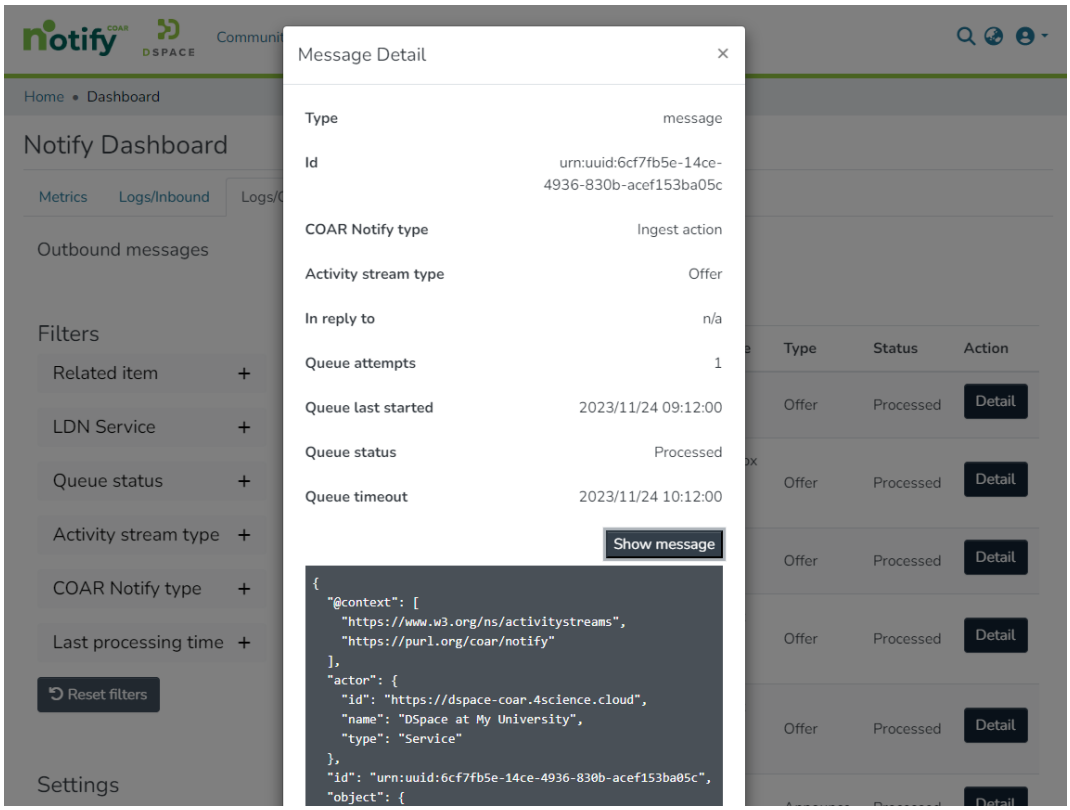


Logs can be scrolled and/or filtered by:

- Related item
- LDN Service
- Queue status
- Activity stream type
- COAR Notify type
- Last processing time

In the logs' table, the value in the “Repository Item” column links to the involved item in the repository.

By clicking on the 'detail' button corresponding to each log, a modal will open displaying the technical information of the received LDN message and it is also be possible to inspect the JSON notification which might be relevant for some purposes.



In case of messages with a queue status equal to “**failed**”, a second button ”**Reprocess**” will be displayed. By clicking this button, a request to reprocess the message is triggered.
This allows administrators to reprocess messages received from LDN services that have not yet been registered in the repository, or whose data (i.e. service URL and IP range) have not been correctly entered in the LDN service registry.

Why a LDN notification should be reprocessed?

- The Service was temporarily down at the moment the LDN notification was sent
- The Service changed the Inbox Url and the notification was not deliverable

In case of deleted items, LDN services, etc., “n/a” will be shown in the logs' table.

Logs with “Untrusted” status don’t display a timestamp, as they have not been processed.

COAR Notify - LDN Services

- Relation with the Quality Assurance Correction Service
- LDN Autodiscovery mechanism
- LDN Services Directory (Registering Services)
- LDN Inbox queueing
- Notify status boxes
- Configuring automatic QA evaluation using the Level of Trust
- Sending LDN Notifications during the submission of an item
- Automatic Inbound pattern triggering
- Item filters for Inbound pattern
- The LDN Consumer
- Understanding the structure of the LDN Notification
    - @context
    - actor
    - context
    - id
    - inReplyTo
    - object
    - origin
    - target
    - type
- POSTMAN COLLECTION

Relation with the Quality Assurance Correction Service

The LDN system, as a message exchanging i/o system, has an inbox and an outbox. Every LDN message refers to a Notify Service: all the Notify Services are configured manually from the admin application form. A Notify Service is just like an authority labelled on LDN messages.

The Quality Assurance system is the implementation of item metadata updates operations. A Quality Assurance Event contains informations for item metadata updates: QAEvent are stored into QAEvent solr collection. All of the QAEvent are shown on an administration form. Every QAEvent can be accepted, ignored or removed: if accepted some metadata of the referred item are modified, if ignored or removed nothing about the item is modified. Every QAEvent has a property named **source** : qa events created by processing an LDNMessage has ***source="coar-notify"***.

To process an LDN message means to create a QAEvents; as soon as the QAEvent is accepted the referred item is updated. We do it by routing LDN Messages from the queue to the LDN Router. The LDN Router is designed into the spring file */dspace/config/spring/api/ldn-coar-notify.xml* on the bean *ldn Router*. The router de-multiplex two attributes read from the LDN Message (two string values inside the array called ***type***) content to java classes called Processors: see **org.dspace.app.ldn.processor.*** package to view them all.

The match between LDN message ***type*** and the QAEvent **topic** (suddenly created by the evaluation of the ldn message from the queue) is configured onto *ldn-coar-notify.xml* spring file. Every Processor owns a list of actions. An action is often an email to be sent and an action to create the qa event. The LDN processor receives the QAEvent topic as a parameter and creates the relative QAEvent.

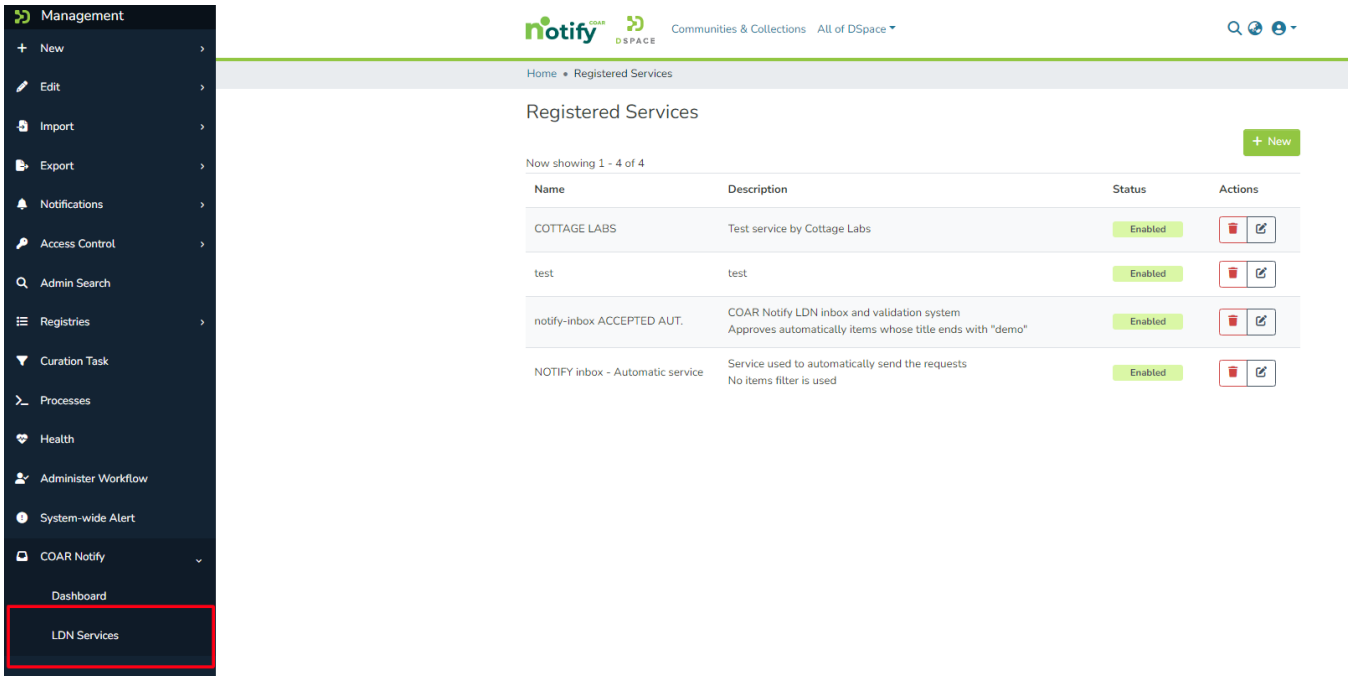
LDN Autodiscovery mechanism

Third party system can retrieve the location of the repository LDN InBox via the LDN autodiscovery mechanism, nevertheless to be able to interact with DSpace they need to be approved by a Repository Administrators and listed in the LDN Services Directory (see next paragraph); otherwise their messages will be flagged as untrusted and not processed at all.

LDN Services Directory (Registering Services)

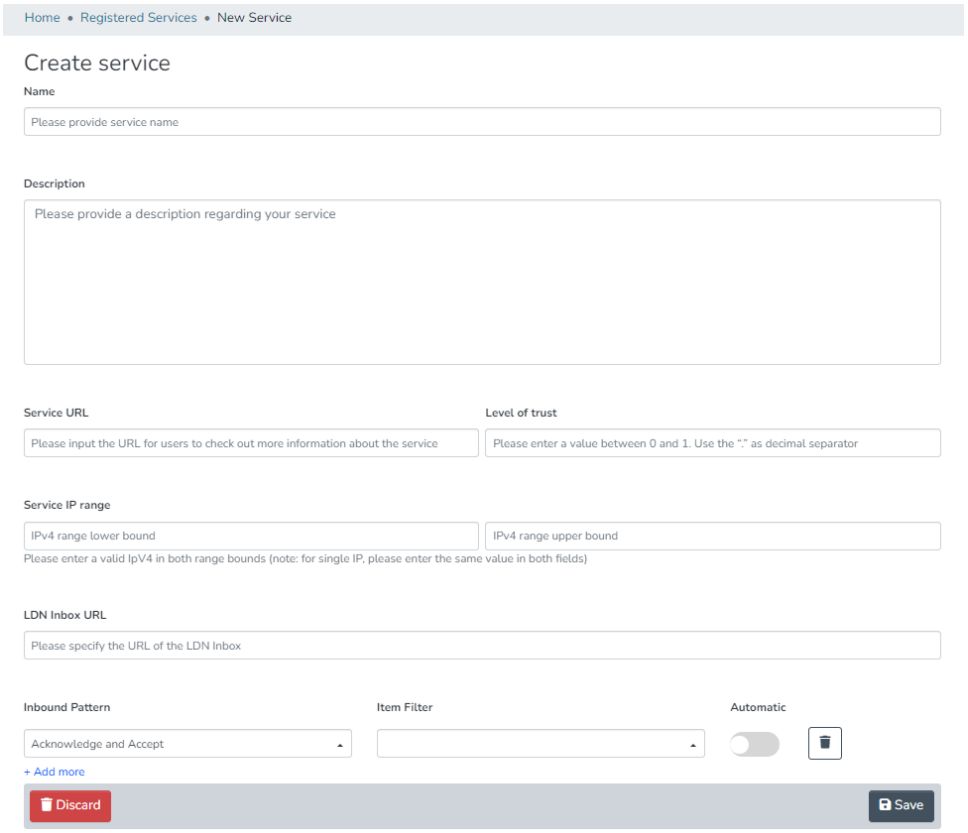
You need to register the external services to allow the handling of the incoming LDN messages.

These services are also used to send LDN notification during the submission process of an item (Please take a look at LDN Inbound pattern)



Administrators can manage services using the menu **LDN Services**.

The following page is used to submit a new LDN Service:



Name: a label for the Notify Service used on the UI

Description: the description of the Notify Service - add details here

Service URL: the url of the remote Notify Service. This is mostly used as a descriptive URL when sending notification to an external system.

The service URL must not be confused with the inbox url. As said this url is descriptive so we expect the main application URL to be added here

Level of Trust: floating point number value accepted between 0 and 1.

This value is used to assign to the service a TRUST value that describes how much reliable the external service is.

This value is not used if the automatic processing of QA Events is disabled

Check the configuration file *dspace/config/spring/api/qaevents.xml* to enable the automatic approval

When the automatic approval is configured the trust value of the LDN Service is check against configured score thresholds.

Thresholds for automatic Approval/Rejection/Ignoring can be set.

Also within a specified range QA Events will need a manual approval

Service IP Range: two IP addresses expected as minimum and maximum.

This range is used to validate the received LDN Messages.

When the LDN notification doesn't match the configured IP Range the notification is stored as UNTRUSTED

LDN Inbox URL: this is the url used to send the LDN Notifications.

This url is also used when a notification is received to retrieve the registered LDN service it belongs to

This URL must is unique among the registered LDN Services

Inbound Pattern: the section for Inbound pattern is the section which describes what operations/actions are supported by the external service

The **pattern** itself can be selected from the dropdown as there's a list of pattern. To better understand pattern usage please refer to the official documentation https://notify.coar-repositories.org/patterns/

The **item filter** is used to describe which item can be processed for the current LDN Service/Inbound pattern.
if the item filter is not set any item is allowed. If the filter is selected only for matching items the LDN Notification will be sent (in case of automatic LDN Notification the notification is not sent, In case of NON-automatic service a validation will prevent requests to be sent to the external service)

The **automatic** flag when set to true the ldn message exchange is performed automatically right after the item submission has finished.
this Automatic workflow generates an Outgoing LDN message targeting the Notify service for the just submitted item.
The automatic flag involves only the submission phase of an item.
If no item filter is set - the LDN notification is generated for any submitted item.

LDN Inbox queueing

LDN incoming messages are stored into the **ldn_message** database table. As far as the property **ldn.enabled** is true and the incoming json is valid, the LDN Message is stored on the table. Together with the storage of the record, the queue status of the message is initialized. The *queue_status* column of the table contains the status of the LDN message inside the queue. All the possible queue_status values are described into the java class ***org.dspace.app.ldn.LDNMessageEntity*** as integer constants.

Status name	Value in DB	Description
QUEUE_STATUS_UNTRUSTED_IP QUEUE_STATUS_UNTRUSTED	0 5	Message must not be routed as it is not trusted. This may occur if the IP address of the notifications' sender doesn't match the provided "IP Range" or if the service inbox URL in the origin section of the message doesn't match with any registered LDN Service entry's Inbox URL.
QUEUE_STATUS_QUEUED	1	Message is waiting in the queue to be processed by the Extractor
QUEUE_STATUS_PROCESSING	2	Message is currently being processed by the Extractor
QUEUE_STATUS_PROCESSED	3	Message has been **successfully** processed by the Extractor
QUEUE_STATUS_FAILED	4	Message has been evaluated but its routing has failed

If all validations are run successfully the LDN Notification status is set as **QUEUE_STATUS_QUEUED** and the notification **will** be processed as soon as the extractor retrieves it from the queue.

The LDN Message logical queue is managed by:

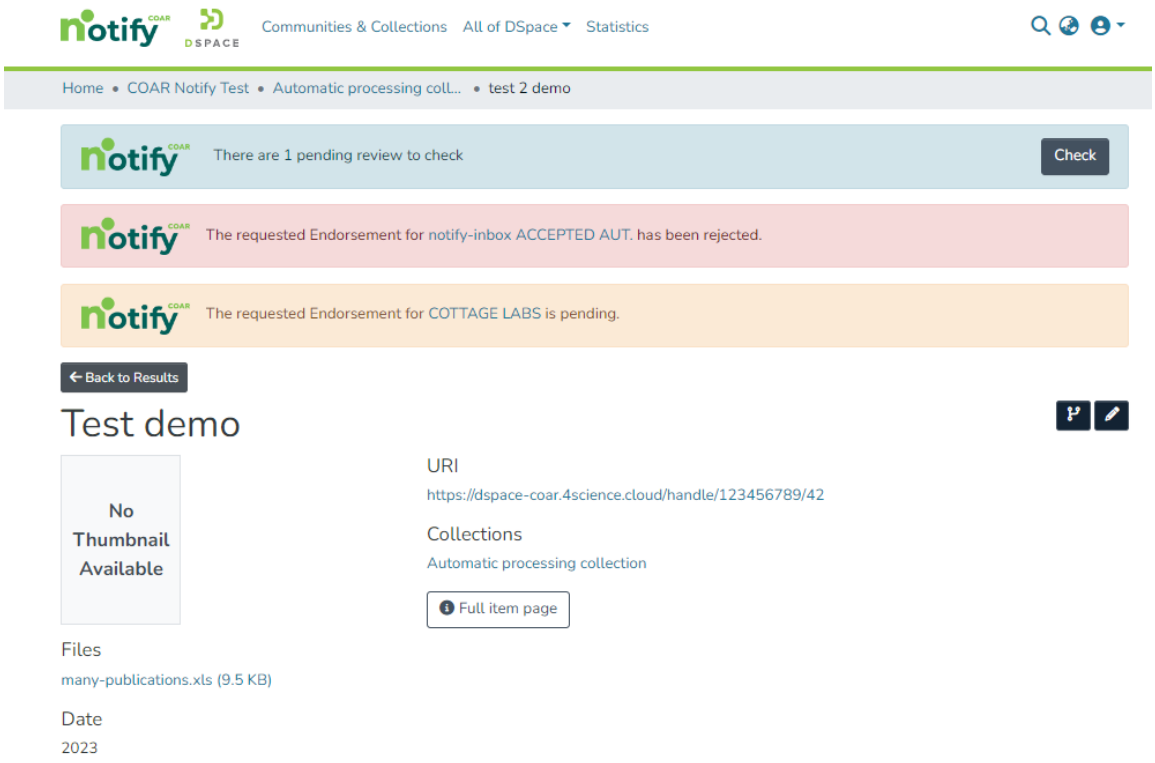
- The **LDN Message Extractor** is an asynchronous DSpace task which retrieves the oldest processable LDN message.
  A LDN notification is processable by the Extractor only if its queue status is **QUEUE_STATUS_QUEUED** other status will not be considered by the extractor.
  The extractor process ends as soon as the LDN extracted message is routed and processed (either with success of failure).
- The **LDN Message Timeout Checke**r is an asynchronous DSpace task that looks for timed-out messages with attempts less than Y, where Y= the value of the configuration property *ldn.processor.max.attempts*. Cron configuration at property *ldn.queue.timeout.checker.cron*
  Each notification if not successfully processed is retrieved in the next execution of the Checker and attempts are kept up to date (increased by 1 and the timeout is increased by X minutes), where X= the value of the configuration property *ldn.processor.queue.msg.timeout* | defaulted to 60.

Please consider that this means that the corresponding QAEvent is not automatically created as soon as the LDN Message is received.
The QA Event related to the LDN Notification will be created only once the notification is successfully processed by the extractor (LDN queue status set to **QUEUE_STATUS_PROCESSED**).

Notify status boxes

Considering these possible scenarios here at: COAR Notify Protocol: Example Scenarios

We have to keep the user updated about the item situation. We do it with colored boxes on its landing (handle) page.



When the Offer message (being it review, endorsement or ingest) has been sent as an outgoing LDN message, and nothing else about it has been received, the yellow box is shown.

When the Offer message has been followed by a related incoming Acknowledgement message: if the ack is a tentative rejct the box shown is red.

When Offer message has been followed by a related Announce incoming message, the box shown is blue! It is blue because receiving an Announce means - then the message is extracted - we produce a new QA Event. As every QA Event it is shown on Item Page and myDSpace page.

Configuring automatic QA evaluation using the Level of Trust

Score is a number 0 < # < 1.
If the proper configuration is enabled an automatic check for approval of QAEvents is run once the LDN message is extracted.
On *qaevents.xml* file the bean ***qaScoreEvaluation*** we can configure three different boundaries to automatically approve/reject/ignore the level of trust:

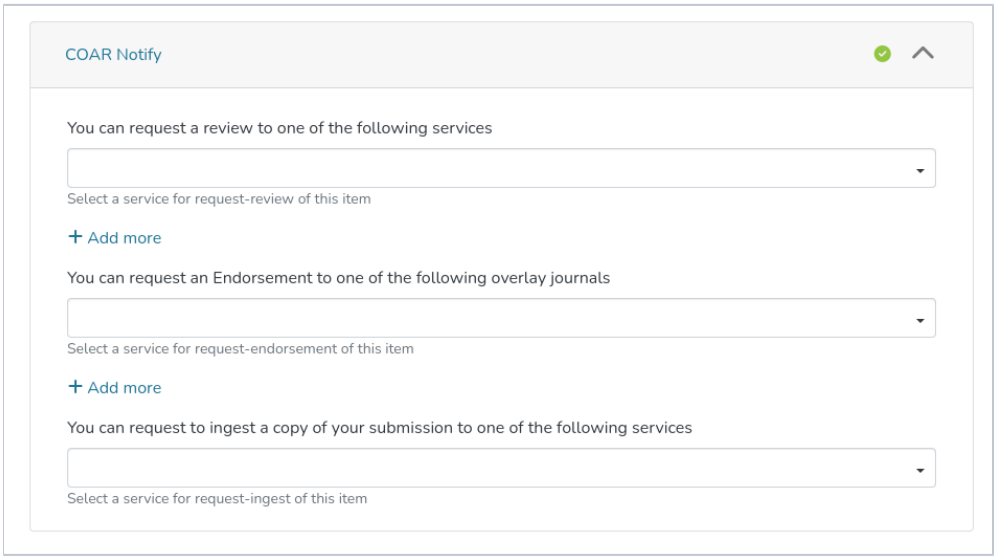
```
<property name="scoreToReject" value="0.3" />
<property name="scoreToIgnore" value="0.5" />
<property name="scoreToApprove" value="0.8" />
```

- if score <= rejection deletes the QAEvent;
- if score <= ignore discards the QAEvent;
- if score >= approval accepts the QAEvent automatically
- if score respects the following: `scoreToIgnore` < score < `scoreToApprove`
  no automatic action is performed and the user must check manually the QA event

This feature must not be confused with the automatic triggering for LDN Inbound pattern

Sending LDN Notifications during the submission of an item

In the example below it is shown the submission of an Item with a new section for COAR-Notify.
In this section we can manually choose **ldn-services** (*which were not marked as automatic*) in order to ask for Review, Endorsement and Ingest to external services.
If No (non-automatic) pattern has been configured for the LDN Service this service will not be displayed in any of the dropdown.



Automatic Inbound pattern triggering

When adding new LDN Inbound pattern to a LDN Service we can select if a pattern has to be automatic or not.

Flagging a pattern as automatic means that the request described by the current pattern is sent automatically to the external service:

- No action is required during the submission to select the LDN Service for a specific pattern
- The service in not even listed in the COAR notify step of the submission

Item filters for Inbound pattern

When adding inbound pattern to a service we have the possibility to add an item-filter for each described pattern

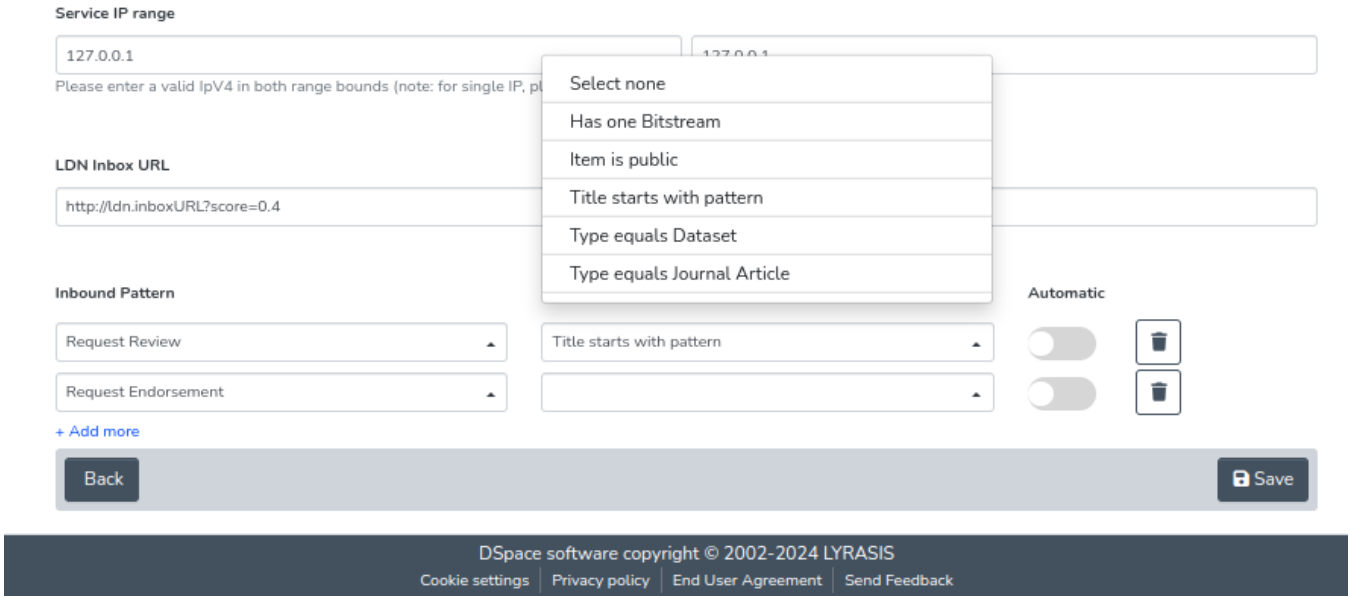
An item filter allows to filter items according to specified criteria.

- If the pattern is flagged as automatic the item filter is evaluated automatically
  - if the item filter is respected the LDN Notification is stored and queued for sending
  - if the item filter is not respected the LDN Notification is not even generated an nothing will be sent to the external service
- If the pattern is **not** flagged as automatic the item filter is evaluated when choosing the service from the dropdown for Notify in the submission
  - if the pattern is not respected an error message will be displayed saying the selected LDN service cannot be used for the current item
  - if the pattern is respected no error message is displayed

Item Filter is a non-mandatory field. It activates a filter to be applied to the item during the submission.
If the item matches the item filter condition the submission will be successful: otherwise the submission will fail with a talkative message.
Visible item filters are configured into a static list mapped into *item-filters.xml* file - on a bean named *ldnItemFilters*. We established for the following item filter to be shown:

Has one Bitstream = the item must have exactly one bundle named "ORIGINAL";
Item is public = anonymous user can see/read the item;
Title starts with pattern = the dc.title metadata of the item must start with 'Pattern';
Type equals dataset = the dc.type metadata of the item must be 'Dataset';
Type equals dataset = the dc.type metadata of the item must be 'Journal Article';

item-filter.xml has all item filters declarations: only the ones declared inside the *ldnItemFilters* map are shown.



The LDN Consumer

The LDN Consumer is the consume responsible for generating the **outgoing** LDN Notification.

This consumer is triggered any time an Item is successfully installed and a check against the registered LDN Services/patterns is done to generate the appropriate LDN Notification

This consumer is responsible for the handling of both automatic a non-automatic patterns

event.consumer.ldnmessage.class = org.dspace.app.ldn.LDNMessageConsumer

event.consumer.ldnmessage.filters = Item+Install

Understanding the structure of the LDN Notification

The LDN Notification includes some important section many of these if improperly set might led to errors in evaluations on the notification

Let's take a deeper look at the LDN Notification structure

@context

The context section is the same for any LDN Notification

```
  "@context": [
    "https://www.w3.org/ns
/activitystreams",
    "https://purl.org/coar/notify"
  ]
```

actor

This section describes the actor which performs the request
This is a descriptive section and doesn't include important information

```
  "actor": {
    "id": "https://review-
service.com",
    "name": "Review Service"
,
    "type": "Service"
  }
```

context

This is one of the most important section.
In this section we have a description of the Item being involved in the LDN Notification

- **id** is the identifier URL of the item in the repository
  this field is really important to identify and link the Item to the current notification

other fields are descriptive fields for the item

```
  "context": {
    "id": "https://research-organisation.org/repository/preprint/201203/421/",
    "ietf:cite-as": "https://doi.org/10.5555/12345680",
    "type": "sorg:AboutPage",
    "ietf:item": {
      "id": "https://research-organisation.org/repository/preprint/201203/421
/content.pdf",
      "mediaType": "application/pdf",
      "type": [
        "Article",
        "sorg:ScholarlyArticle"
      ]
    }
  }
```

id

The identifier of the LDN notification (Each notification has a different id)

```
"id": "urn:uuid:94ecae35-dcfd-4182-8550-22c7164fe23f"
```

inReplyTo

This field is used to reply to other LDN notification so that the system knows if the current notification is a response

```
"inReplyTo": "urn:uuid:0370c0fb-bb78-4a9b-87f5-bed307a509dd"
```

object

This is a descriptive section of the received Review, Endorsement etc..

- **id** is the URL of the review/endorsement etc.

```
  "object": {
    "id": "https://review-service.com/review/geo
/202103/0021",
    "ietf:cite-as": "https://doi.org/10.3214
/987654",
    "type": [
      "Document",
      "sorg:Review"
    ]
  }
```

origin

This is an important section since it is used (for incoming notifications) to determine the service among the registered ones.
If the service is not recognized as registered the notification is **untrusted**

The field being used for this check is **inbox**

```
  "origin": {
    "id": "https://review-service.
com/system",
    "inbox": "https://review-
service.com/inbox/",
    "type": "Service"
  }
```

target

This section is used to describe the target system involved in the notification workflow

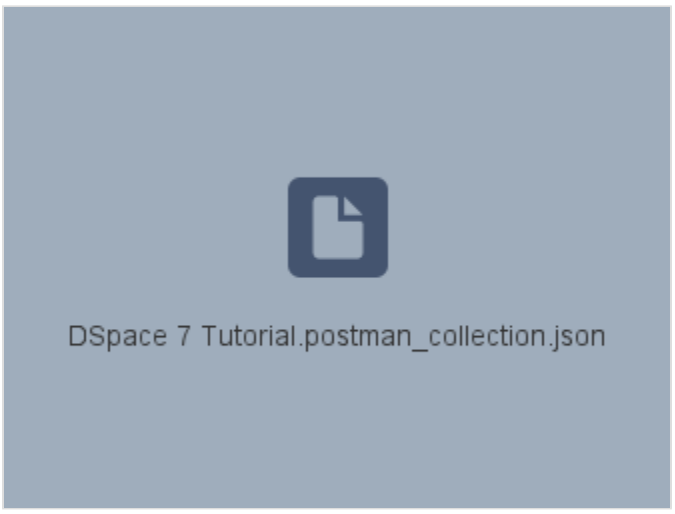
```
  "target": {
    "id": "https://generic-service.com
/system",
    "inbox": "https://generic-service.com
/system/inbox/",
    "type": "Service"
  }
```

type

This type section is important since it's determining how to process/evaluate any received notification

```
  "type": [
    "Announce",
    "coar-notify:
ReviewAction"
  ]
```

POSTMAN COLLECTION


DSpace 7 Tutorial.postman_collection.json

Use Cases - Getting two DSpace application to communicate with each other

- Prerequisites
- COAR Notify Workflow Reference
- Register the other DSpace System/Service in the current DSpace instance
    - DSpace 1 configuration
    - DSpace 2 configuration
    - Notes about configuration
  - Further customization
    - Filters
    - Relation Metadata
  - Real use case
  - Notes
  - Some UI examples
    - Registering the Service (both DSpace instances)
    - Configuring the Announce Relationship Pattern (both DSpace instances)
    - Start a submission and fill the related item field (Configure it if not present) (DSpace 1)
    - The system receives the notification (DSpace 2)
    - Approve the generated QA Event (DSpace 2)
    - Check the Item for the newly added relation (DSpace 2)

Prerequisites

Having two DSpace instances:

- **DSpace 1**
- **DSpace 2**

COAR Notify Workflow Reference

https://coar-notify.net/catalogue/workflows/repository-relationship-repository/

Register the other DSpace System/Service in the current DSpace instance

DSpace 1 configuration

Create a new LDN Service on **DSpace 1**, ensuring that the LDN Inbox is correctly configured. Generally this URL is in the format: **https://<baseurl>/server/ldn/inbox.**

Configure the **Service IP range** and **Level of trust** as required.

Make sure to configure the **Announce Relationship** as supported pattern and flag it as **Automatic**. This is very important since the **Announce Relationship** is the only pattern that allows two DSpace to intercommunicate, and this pattern can only be used in the **Automatic** mode since no submission additional step allows this workflow to be triggered.

DSpace 2 configuration

Create a new LDN Service on **DSpace 2**, ensuring that the LDN Inbox is correctly configured. Generally this URL is in the format: **https://<baseurl>/server/ldn/inbox.**

Configure the **Service IP range** and **Level of trust** as required.

Make sure to configure the **Announce Relationship** as supported pattern and flag it as **Automatic**. This is very important since the **Announce Relationship** is the only pattern that allows two DSpace to intercommunicate, and this pattern can only be used in the **Automatic** mode since no submission additional step allows this workflow to be triggered.

Notes about configuration

As seen above, each DSpace service must be configured to know and recognize the other DSpace instance, so that the current system knows the external system and is aware of what kind of notifications should be managed.

Also, the IP authorization is checked, otherwise the notification is not processed and is flagged as **untrusted.**

Further customization

Filters

The Announce Relationship allows to notify a created relationship between systems.

Since multiple LDN Services could be configured and they could all be using the **Announce Relationship** pattern, it makes sense to define an item filter for each provided pattern for the defined LDN Service. These filters are not out of the box (only basic filters are provided) but they could be defined on **item-filters.xml**. These filters can be configured in order to allow the LDN message to be generated/triggered only on a given condition (for instance: a specific relation URL pattern is recognized, item is openaccess and so on..).

Relation Metadata

By default the Announce Relationship pattern retrieves the **dc.relation** metadata, which is currently not configured in any of the standard submission forms and must be added manually.

This metadata can be overridden by setting the java property **ldn.notify.relation.metadata** so that any custom metadata could be used.

Real use case

After making sure all the above configurations are properly set proceed with the following:

1. Start a submission on **DSpace 1** for an item and set the dc.relation metadata (or any other metadata, in case it is overridden by the property **ldn.notify.relation.metadata**)
2. Once the submission is completed on **DSpace 1**, wait for the notification to be processed and sent (access the LDN Dashboard to check the notification status)
3. Once the **LDN Message** is delivered to **DSpace 2**, move to the **DSpace 2** system and check that the notification is correctly received (and processed)
4. Check the **Quality Assurance** page on **DSpace 2** for the processed notifications and approve them (Approval is not requested if a configuration for automatic approval has been provided, check the documentation page COAR Notify - LDN Services#ConfiguringautomaticQAevaluationusingtheLevelofTrust)
5. Once the notification is approved on **DSpace 2**, a new metadata is added to the item. This metadata is the reference to the **DSpace 1** related item (i.e. the item involved in the relationship)

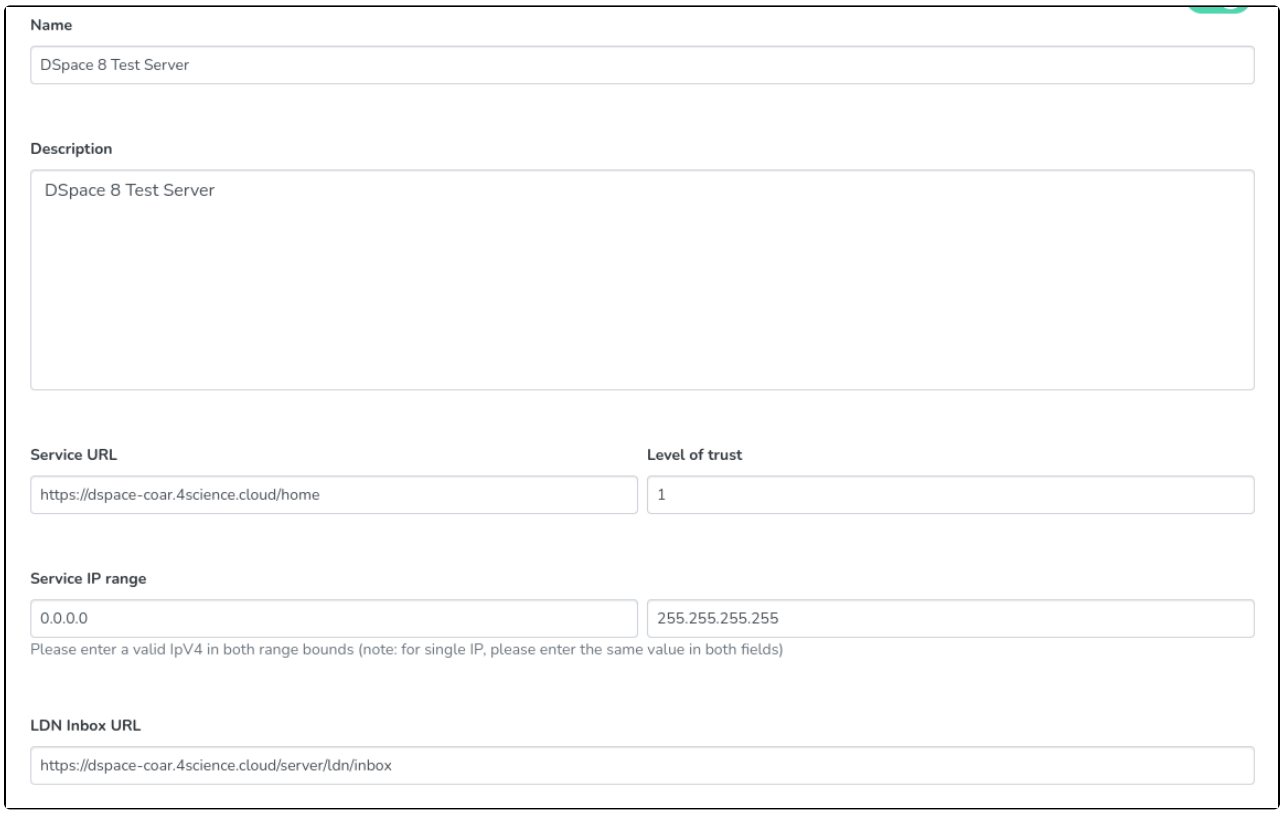
Notes

The metadata added to the Item(s), once the QA event for the LDN Notification is approved, can be customized in **qaevents.xml** in the following bean:

```
    <bean id="RelationMetadataAction" class="org.dspace.qaevent.action.QANotifyMetadataMapAction">
            <property name="types">
                <map>
              <entry key="default" value="datacite.relation.isReferencedBy" />
              <entry key="http://purl.org/vocab/frbr/core#supplement" value="datacite.relation.
isSupplementedBy" />
                </map>
            </property>
    </bean>
```

Some UI examples

Registering the Service (both DSpace instances)



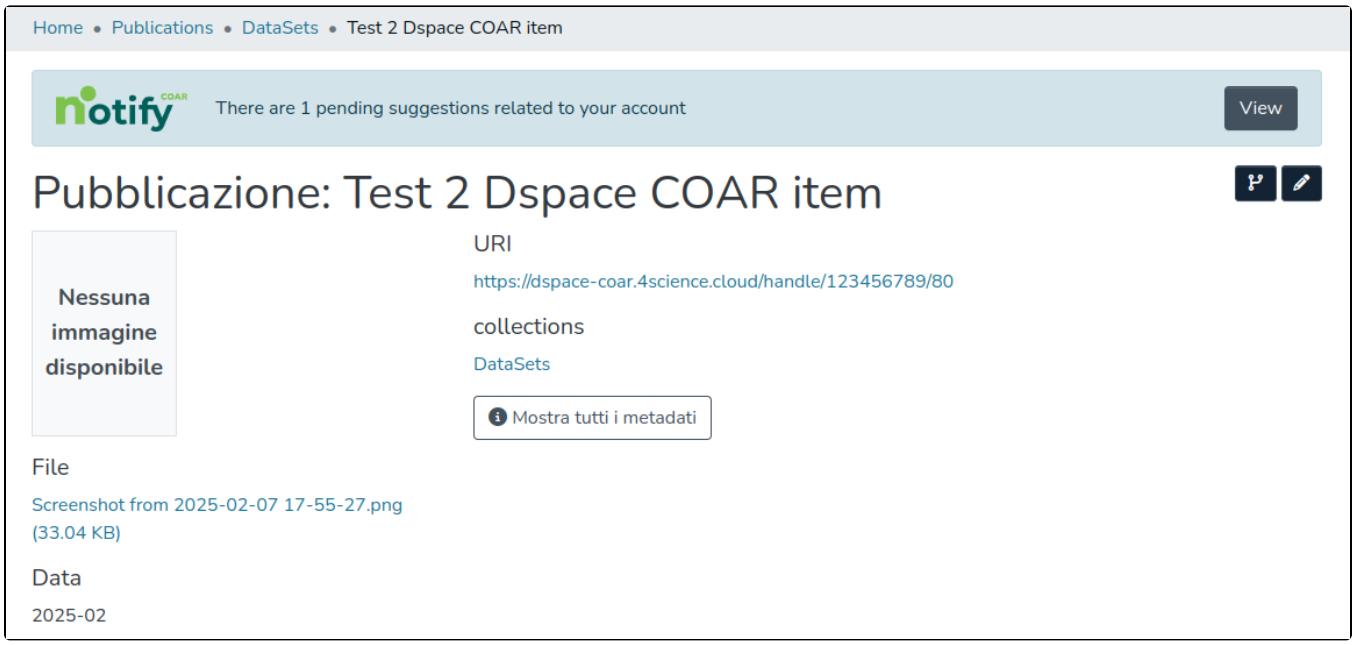
Configuring the Announce Relationship Pattern (both DSpace instances)



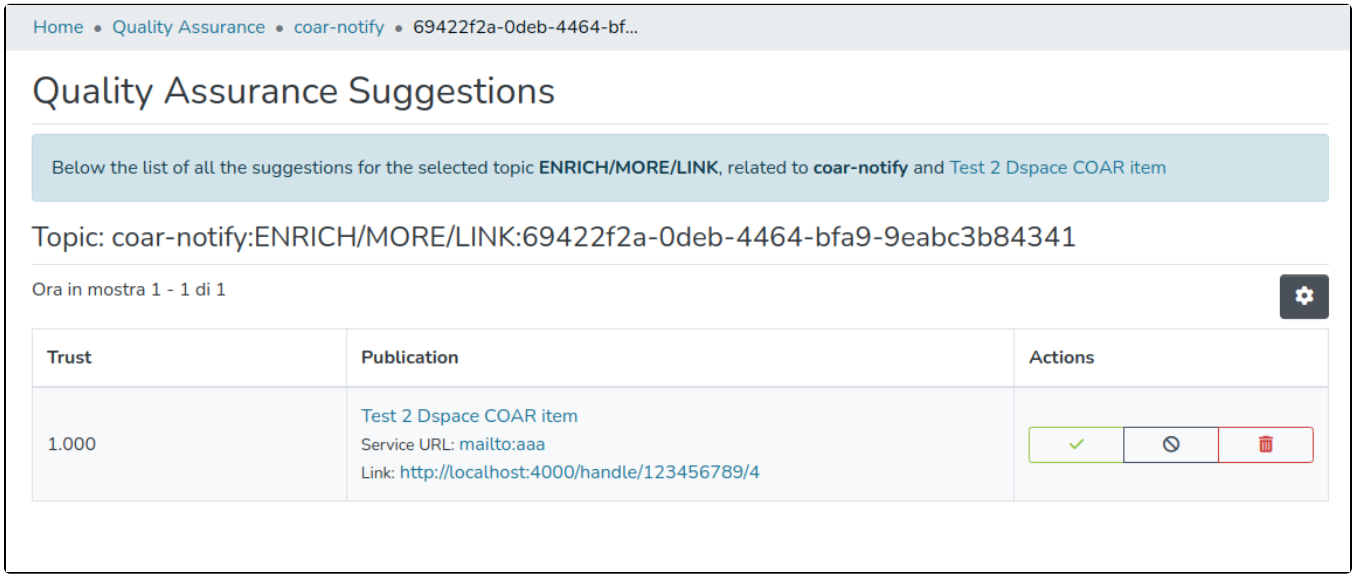
Start a submission and fill the related item field (Configure it if not present) (DSpace 1)



The system receives the notification (DSpace 2)



Approve the generated QA Event (DSpace 2)



Check the Item for the newly added relation (DSpace 2)

Home • Publications • DataSets • Test 2 Dspace COAR item

Pubblicazione: Test 2 Dspace COAR item

Nessuna immagine disponibile

File

Screenshot from 2025-02-07 17-55-27.png (33.04 KB)

Data

2025-02

URI

https://dspace-coar.4science.cloud/handle/123456789/80

collections

DataSets

Dataset

http://localhost:4000/handle/123456789/4

Mostra tutti i metadati

Notifications

The Notifications include the possibility to claim publication suggested by external system, for example ORCID and the Quality assurance.

See more in the pages:

- Publication Claim
- Quality Assurance

Publication Claim

- Publication Claim
  - Enabling Publication Claim
  - Data source & Provider configuration
    - Data Source
    - Suggestion Provider
      - Current Scorers
    - Running the Suggestion Loader
    - Source providers OpenAIRE
    - Source providers OpenAlex
      - Key Properties
      - OpenAlex Publication Claim
    - SOLR Suggestion Core
  - Verify Suggestions
    - Researcher
      - Approve and Import
    - Admin
  - Repository Manager UI
  - Researcher UI
  - Processing the decisions
  - Rest Contract

Publication Claim

> *Scenario: A new researcher joins the institution and logins for the first time in the repository. The publication claim services found most of their publications in the OpenAIRE network and prompts for import. The researcher reviews the list, confirms the authorship and imports the publication saving a significant amount of (often publicly payed) time. Moreover, the authorship confirmation will come back later to OpenAIRE offering useful information about the data quality and potential enrichment. The same applies for publications authored by researchers in different institutes, having the data in multiple repositories makes the data more reliable and raises the chance to get more information and content from any of the authors.*

The goal of the Publication Claim service is to support such scenarios by simplifying the publication import process and enhancing metadata quality across ecosystems.

This feature is designed to be **provider-agnostic**, supporting integration with multiple external sources simultaneously.

Currently, the service supports:

- **OpenAIRE Research Graph**, via its Publication REST API.
- **OpenAlex**, using its public API.

Enabling Publication Claim

In order to use Publication Claim, you MUST first enable:

- Configurable Entities
- Researcher Profiles
- Quality Assurance

No additional settings are required, but this feature only works for users who have a Researcher Profile established, and when Quality Assurance is enabled.

Data source & Provider configuration

The publication claim service queries external APIs (e.g., OpenAIRE and OpenAlex) using researcher information available in the repository (e.g., names, identifiers). The results are processed through a configurable pipeline of Java classes that score and filter the publications based on various criteria. Records previously rejected by the researcher are automatically filtered out to prevent redundant suggestions.

Providers are configured in the Spring configuration file:
`dspace/config/spring/api/suggestions.xml`

Data Source

The OpenAIRE Publication REST API are used to retrieve publication that could be authored by researcher at the Institution. The OpenAIRE Publication REST API are queried using the names known by the repository for its researchers, the retrieve list is later reduced passing identified publications to a pipeline of JAVA classes that can promote or reject his inclusion in the suggestion list. Publications previously discarded by the researcher are automatically filter out avoiding to re-present the same publication again and again.

The suggestion providers are defined in the `dspace/config/spring/api/suggestions.xml` spring configuration file.

```
<util:map id="suggestionProviders" map-class="java.util.HashMap"
                 key-type="java.lang.String" value-type="org.dspace.app.suggestion.SuggestionProvider">
           <entry key="openaire" value-ref="OpenairePublicationLoader" />
        <entry key="openalex" value-ref="OpenAlexPublicationLoader"/>
       </util:map>

   <!-- OpenAIRE -->
   <bean id="OpenairePublicationLoader" class="org.dspace.app.suggestion.loader.PublicationLoader">
       <property name="sourceName" value="openaire" />
       <property name="primaryProvider" ref="openaireLiveImportDataProviderByAuthor" />
       <property name="otherProviders">
           <list>
               <ref bean="openaireLiveImportDataProviderByTitle"/>
           </list>
       </property>
       <property name="names">
           <list>
               <value>dc.title</value>
           </list>
       </property>
       <property name="pipeline">
           <list>
               <bean
                       class="org.dspace.app.suggestion.scorer.AuthorNamesScorer">
                   <property name="contributorMetadata">
                       <list>
                           <value>dc.contributor.author</value>
                       </list>
                   </property>
                   <property name="names">
                       <list>
                           <value>dc.title</value>
                       </list>
                   </property>
               </bean>
               <bean
                       class="org.dspace.app.suggestion.scorer.DateScorer">
                   <property name="birthDateMetadata" value="person.birthDate" />
                   <property name="publicationDateMetadata" value="dc.date.issued" />
               </bean>
           </list>
       </property>
   </bean>


   <!-- OpenAlex -->
   <bean id="OpenAlexPublicationLoader" class="org.dspace.app.suggestion.openalex.OpenAlexPublicationLoader">
       <property name="sourceName" value="openalex"/>
       <property name="primaryProvider" ref="openalexPublicationByAuthorIdLiveImportDataProvider"/>
       <property name="otherProviders">
           <list>
               <ref bean="openalexPublicationLiveImportDataProvider"/>
           </list>
       </property>
       <property name="names">
           <list>
               <value>dc.identifier.openalex</value>
               <value>dc.title</value>
           </list>
       </property>
       <property name="pipeline">
           <list>
               <bean class="org.dspace.app.suggestion.scorer.AuthorNamesScorer">
                   <property name="contributorMetadata">
                       <list>
                           <value>dc.contributor.author</value>
                       </list>
                   </property>
                   <property name="names">
                       <list>
```

```
                        <value>dc.title</value>
                    </list>
                </property>
            </bean>
        </list>
    </property>
</bean>
```

Each `SuggestionProvider` is identified by a unique name used as a key in the `suggestionProviders` map defined in the Spring configuration file.

Suggestion Provider

The provider implementation is handled by the Java class `org.dspace.app.suggestion.loader.PublicationLoader`, and it is configured via the following properties:

- `primaryProvider`: Specifies which `ExternalDataProvider` to use for retrieving publication records (e.g., `openaireLiveImportDataProviderByAuthor`).
- `otherProviders`: A list of additional `ExternalDataProviders` that may return overlapping records. This is used to automatically remove from the suggestion list records that are imported manually by the researcher from these other providers.
- `names`: A list of metadata fields used to construct the search query against the external data provider.
- `pipeline`: Defines the list of scorers used to evaluate the results.

Current Scorers

- `AuthorNamesScorer`: Ensures the suggested publication truly belongs to the researcher, mitigating false positives due to name collisions. For example, a query for "Bollini Susanna" might incorrectly match works co-authored by "Andrea Bollini" and "Susanna Mornati".
- `DateScorer`: Validates the publication date against an expected activity range for the researcher. This range is calculated based on available metadata like birth date or graduation date, but can also be set manually in the Researcher Profile.

Running the Suggestion Loader

The DSpace script class `org.dspace.app.suggestion.runnable.PublicationLoaderRunnableCli` is responsible for querying all configured suggestion providers (e.g., OpenAIRE, OpenAlex) and storing the resulting publication suggestions in the dedicated **SOLR core** for further processing and display.

This script is **provider-agnostic**: it dynamically uses the list of `SuggestionProviders` configured in the Spring context. Therefore, as long as a provider (like OpenAIRE or OpenAlex) is defined and enabled, its results will be retrieved and processed automatically.

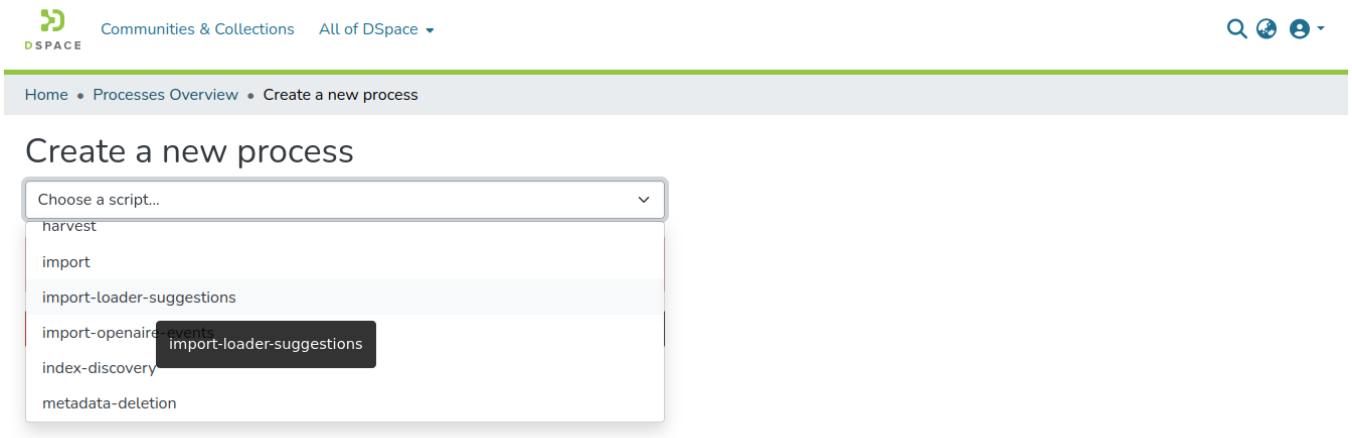
You can run the script from the command line or the Script UI.

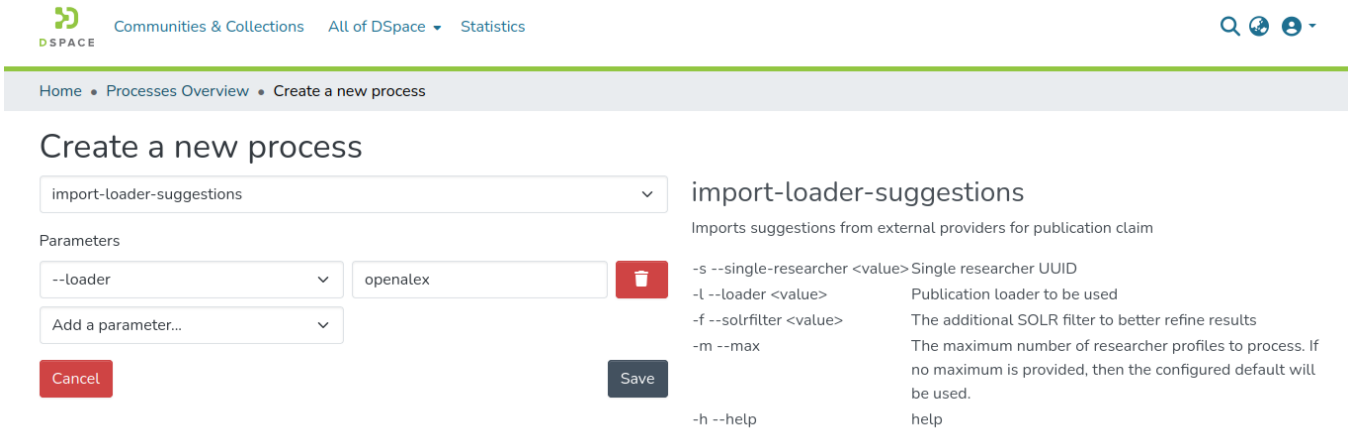
To run the loader from the dspace installation bin folder

```
./dspace import-loader-suggestions [-l suggestion-provider] [-s uuid-of-single-researcher]
```

without the `s` parameter the script will process all the researcher available in the system.

The script can be also run from the Script UI so that it is also available to repository manager that cannot be access the CLI.





Source providers OpenAIRE

Two external source providers, openAIRE Publications By Title and By Author have been defined according to the standard DSpace External Sources framework. It is activated in the `config/spring/api/external-services.xml` as follow

```
<bean id="openaireLiveImportDataProviderByAuthor" class="org.dspace.external.provider.impl.
LiveImportDataProvider">
        <property name="metadataSource" ref="openaireImportServiceByAuthor"/>
        <property name="sourceIdentifier" value="openaire"/>
        <property name="recordIdMetadata" value="dc.identifier.other"/>
        <property name="supportedEntityTypes">
            <list>
                <value>Publication</value>
            </list>
        </property>
    </bean>
    <bean id="openaireLiveImportDataProviderByTitle" class="org.dspace.external.provider.impl.
LiveImportDataProvider">
        <property name="metadataSource" ref="openaireImportServiceByTitle"/>
        <property name="sourceIdentifier" value="openaireTitle"/>
        <property name="recordIdMetadata" value="dc.identifier.other"/>
        <property name="supportedEntityTypes">
            <list>
                <value>Publication</value>
            </list>
        </property>
    </bean>
```

with the importer services defined via the Live Import Framework in `/dspace-api/src/main/resources/spring/spring-dspace-addon-import-services.xml` as follow

```
    <bean id="openaireImportServiceByAuthor"
          class="org.dspace.importer.external.openaire.service.OpenAireImportMetadataSourceServiceImpl" scope="
singleton">
        <property name="metadataFieldMapping" ref="openaireMetadataFieldMapping"/>
        <property name="queryParam" value="author"/>
    </bean>
    <bean id="openaireImportServiceByTitle"
          class="org.dspace.importer.external.openaire.service.OpenAireImportMetadataSourceServiceImpl" scope="
singleton">
        <property name="metadataFieldMapping" ref="openaireMetadataFieldMapping"/>
        <property name="queryParam" value="title"/>
    </bean>
    <bean id="openaireMetadataFieldMapping"
          class="org.dspace.importer.external.openaire.service.metadatamapping.OpenAireFieldMapping">
    </bean>
```

The mapping between the openAIRE Publications metadata and the dspace metadata is provided in the `config/spring/api/openaire-integration.xml` using the usual xpath approach of the DSpace Live Import Framework.

> *Having used the Live Import Framework internally to the loader to perform the query has had the side benefit to make available the publication data of the openAIRE Research Graph also to the direct import functionality of DSpace, so that the researcher can now query the openAIRE graph and import publication on demand.*

Source providers OpenAlex

The OpenAlex integration defines multiple **external source providers** using the standard DSpace External Sources framework. These providers enable live import of metadata for various entity types such as publications, people, journals, institutions, and more.

They are configured in the `config/spring/api/external-services.xml` file and follow the `LiveImportDataProvider` model.

Only two external providers are used in the context of publication claim:

```
<bean id="openalexPublicationLiveImportDataProvider"
      class="org.dspace.external.provider.impl.LiveImportDataProvider">
    <property name="metadataSource" ref="openalexImportPublicationService"/>
    <property name="sourceIdentifier" value="openalexPublication"/>
    <property name="recordIdMetadata" value="dc.identifier.openalex"/>
    <property name="supportedEntityTypes">
        <list>
            <value>Publication</value>
            <value>none</value>
        </list>
    </property>
</bean>

<bean id="openalexPublicationByAuthorIdLiveImportDataProvider"
      class="org.dspace.external.provider.impl.LiveImportDataProvider">
    <property name="metadataSource" ref="openalexImportPublicationByAuthorIdService"/>
    <property name="sourceIdentifier" value="openalexPublicationByAuthorId"/>
    <property name="recordIdMetadata" value="dc.identifier.openalex"/>
    <property name="supportedEntityTypes">
        <list>
            <value>Publication</value>
            <value>none</value>
        </list>
    </property>
</bean>
```

Key Properties

- `metadataSource`: The service responsible for retrieving and transforming data from OpenAlex.
- `sourceIdentifier`: A unique string used to refer to this source throughout the system.
- `recordIdMetadata`: The metadata field used to store the unique OpenAlex identifier (e.g., `dc.identifier.openalex`).
- `supportedEntityTypes`: Lists which DSpace entity types this provider supports (e.g., `Publication`, `Person`, `OrgUnit`).

Each provider bean is linked to a **metadata source service** (or "importer service") defined via the Live Import Framework. These services are configured in the file:

```
/dspace-api/src/main/resources/spring/spring-dspace-addon-import-services.xml
```

They are defined as follows:

```
<bean id="openalexImportPublicationService"
      class="org.dspace.importer.external.openalex.service.OpenAlexImportMetadataSourceServiceImpl">
    <property name="metadataFieldMapping" ref="openalexPublicationMetadataFieldMapping"/>
    <property name="url" value="${openalex.url.works}"/>
</bean>

<bean id="openalexImportPublicationByAuthorIdService"
      class="org.dspace.importer.external.openalex.service.OpenAlexImportMetadataSourceServiceImpl">
    <property name="metadataFieldMapping" ref="openalexPublicationMetadataFieldMapping"/>
    <property name="url" value="${openalex.url.works}?filter=authorships.author.id:"/>
</bean>
```

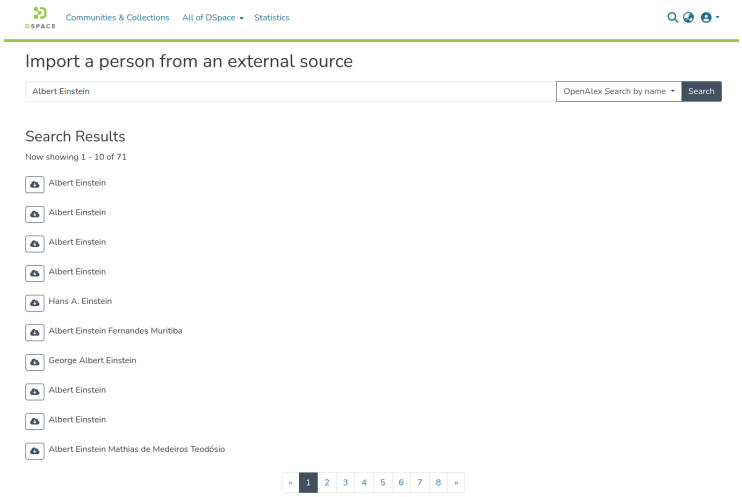
The mapping between the **OpenAlex metadata** and **DSpace metadata fields** is defined in the file:

```
config/spring/api/openalex-integration.xml
```

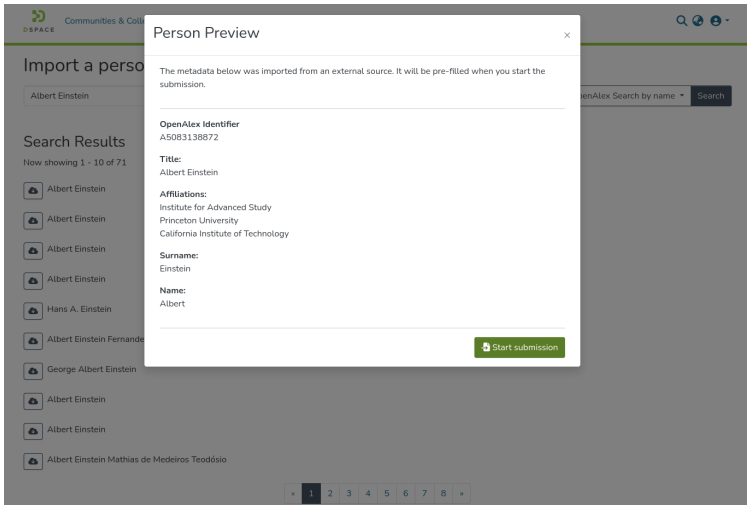
OpenAlex Publication Claim

NOTE: To use the publication claim feature with OpenAlex, the Researcher Profile (Person Entity) must include a `dc.identifier.openalex` metadata (e.g., `A1969205037`). It won't work if you set the `dc.identifier.openalex` with the full url (e.g., `` `https://openalex.org/A1969205037` ``).

One way to automatically set this `dc.identifier.openalex` field is by importing the author from OpenAlex using live import and then create the researcher profile.



The selected author contains the openalex id that will be mapped to `dc.identifier.openalex`.



SOLR Suggestion Core

The SOLR **suggestion** core has the following structure

```
<fields>
    <field name="source" type="string" indexed="true" stored="true" omitNorms="true" />
    <field name="suggestion_fullid" type="string" indexed="true" stored="true" omitNorms="true" />
    <field name="suggestion_id" type="string" indexed="true" stored="true" omitNorms="true" />
    <field name="target_id" type="string" indexed="true" stored="true" omitNorms="true" />
    <field name="title" type="string" indexed="true" stored="true" omitNorms="true" />
    <field name="date" type="string" indexed="true" stored="true" omitNorms="true" />
    <field name="display" type="string" indexed="true" stored="true" omitNorms="true" />
    <field name="contributors" type="string" indexed="true" stored="true" omitNorms="true" multiValued="true" />
    <field name="abstract" type="string" indexed="true" stored="true" omitNorms="true" />
    <field name="category" type="string" indexed="true" stored="true" omitNorms="true" multiValued="true"/>
    <field name="external-uri" type="string" indexed="true" stored="true" omitNorms="true" />
    <field name="processed" type="boolean" indexed="true" stored="true" omitNorms="true" />
    <field name="trust" type="double" indexed="true" stored="true" omitNorms="true" />
    <field name="evidences" type="string" indexed="false" stored="true" omitNorms="true" />
</fields>
<uniqueKey>suggestion_id</uniqueKey>
```

the `source` field would allow the reuse of such structure by other sources than openAIRE.

Three endpoints have been designed to expose the result of the processing to the DSpace UI and so to the Repository Managers and single researchers:

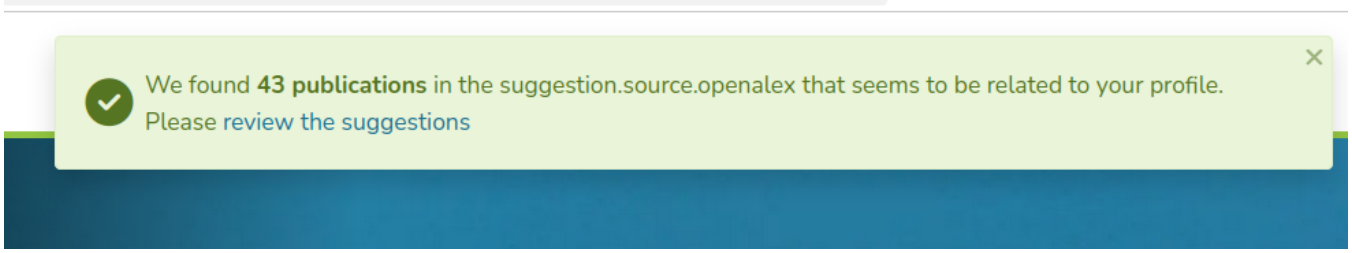
- `/api/integration/suggestionsources` to provide access to summary information about the available suggestion from each source (openaire, orcid, etc.)
- `/api/integration/suggestiontargets` to provide access to summary information about the available suggestions for a specific researcher
- `/api/integration/suggestions` to provide access to the detailed suggestions so that they can be reviewed and managed by the repository manager or the researcher to whom they related

The detailed REST contract for such endpoints are available on the 4Science Rest7Contract repository and embedded at the bottom of the page for easy reference.

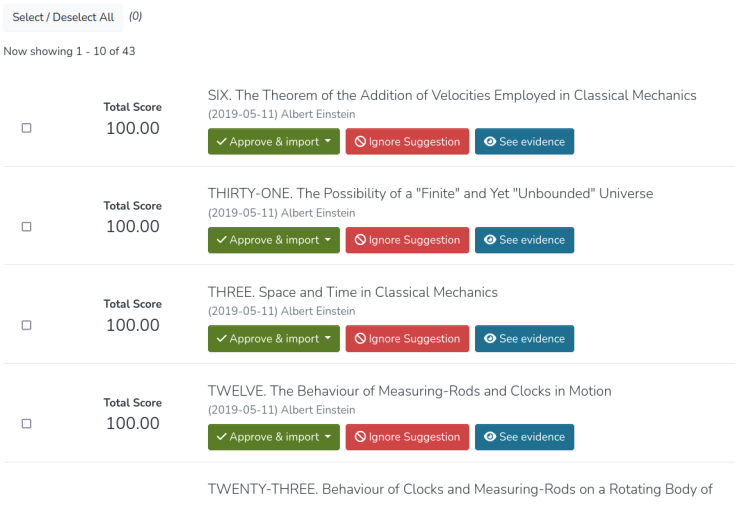
Verify Suggestions

Researcher

Log in as the researcher. You should see a suggestions pop-up in MyDSpace.



Click **"Review suggestions"** to see the list of suggested publications.

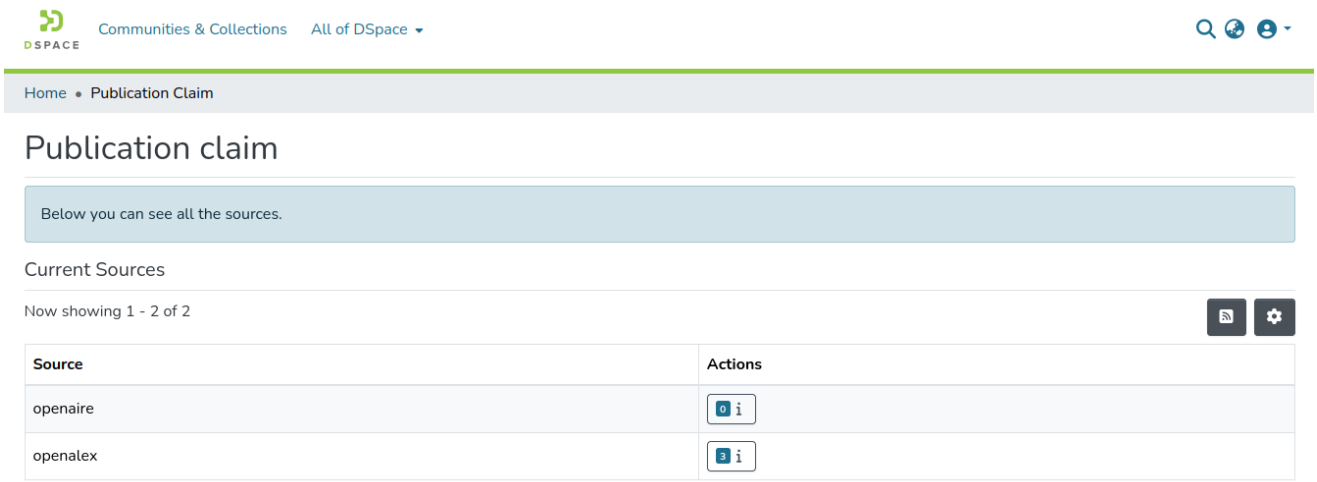


Approve and Import

Click **"Approve & Import"** to create a workspace item from a suggestion.

Admin

Admins can access all suggestions under **Notifications > Publication Claim**.

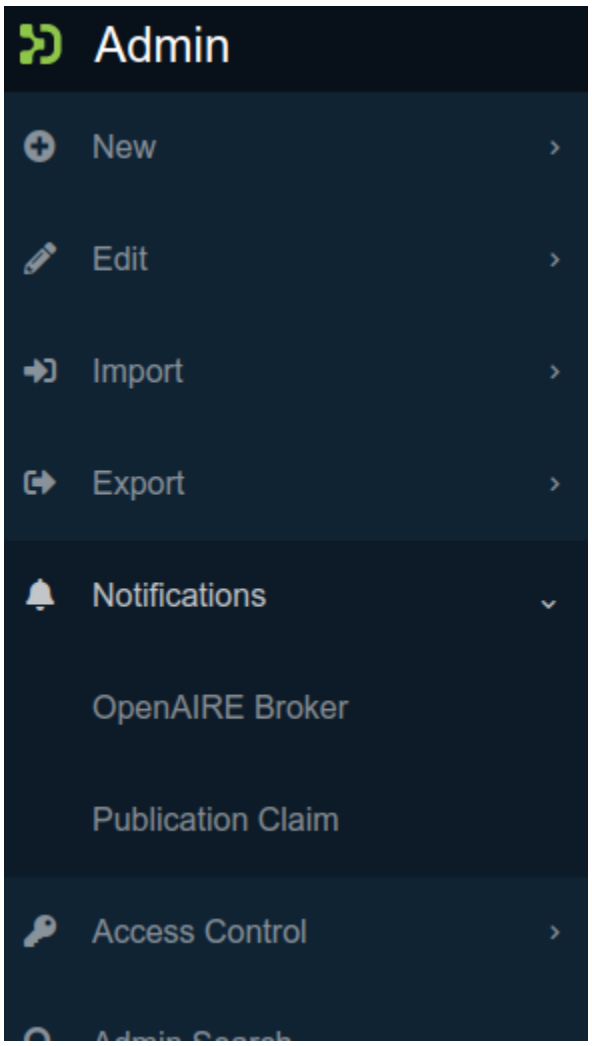


You can filter by suggestion source (e.g., OpenAlex) using the **Actions** menu.

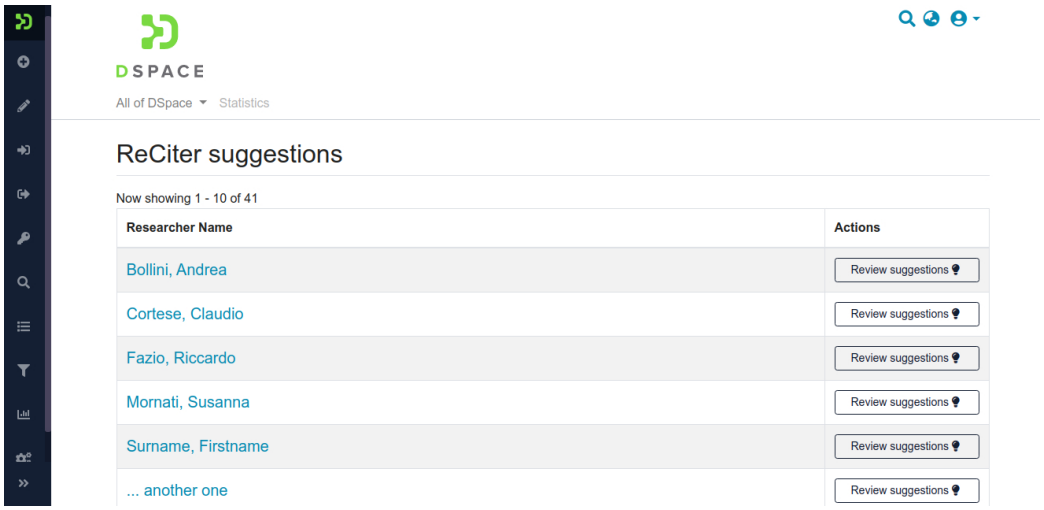


Repository Manager UI

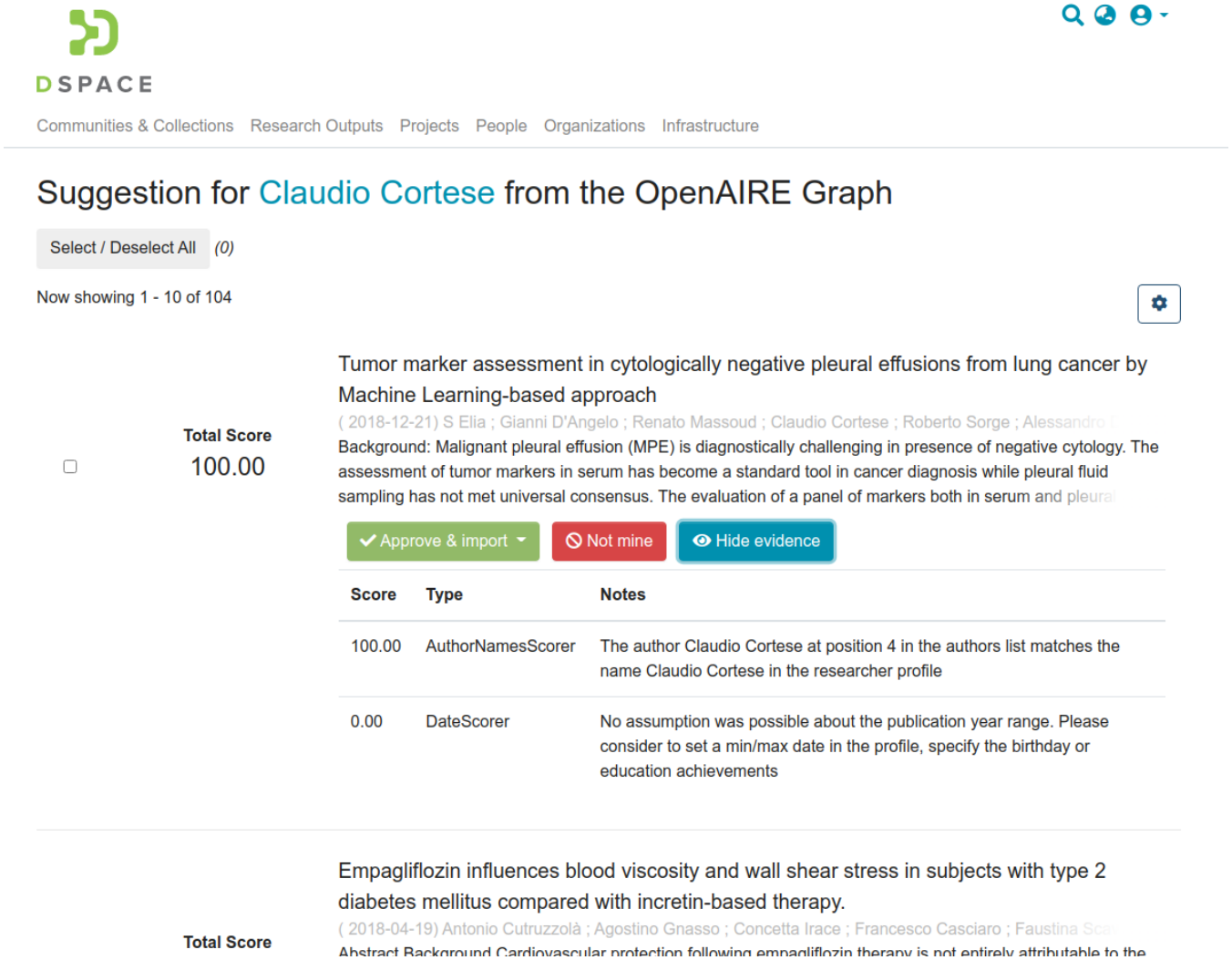
The resulting UI is accessible for the Repository Manager from the administrative menu. As entry point for the features a “Notifications” menu entry has been added to the DSpace administrative menu, from where the repository manager will be able to manage the suggestions got from the different sources.



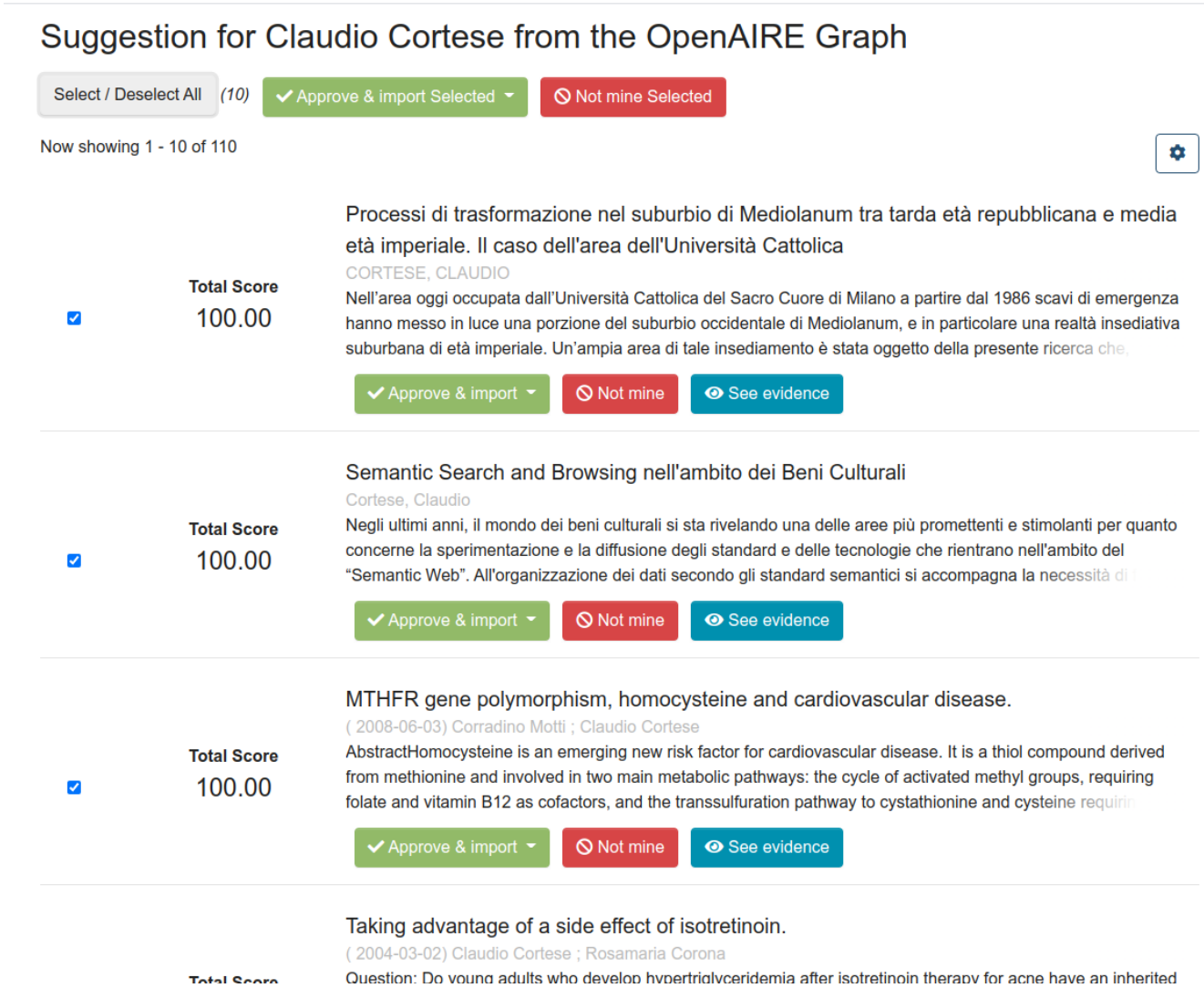
A list of local profiles with candidate publications will be shown so that the repository manager can review them directly or support the researcher:



For each candidate the available suggestions are shown, sorted by the evaluated total score (summing up all the processed evidences). Using the buttom see evidence is possible to get detailed information about the score



The suggested authorship of each article can be confirmed importing the data locally, or rejected. This operation can be performed individually but also simultaneously for all the selected suggestions, speeding up the process. The decision can also be guided by inspecting the matching evidences which are displayed for each suggestion by clicking on 'See evidence'



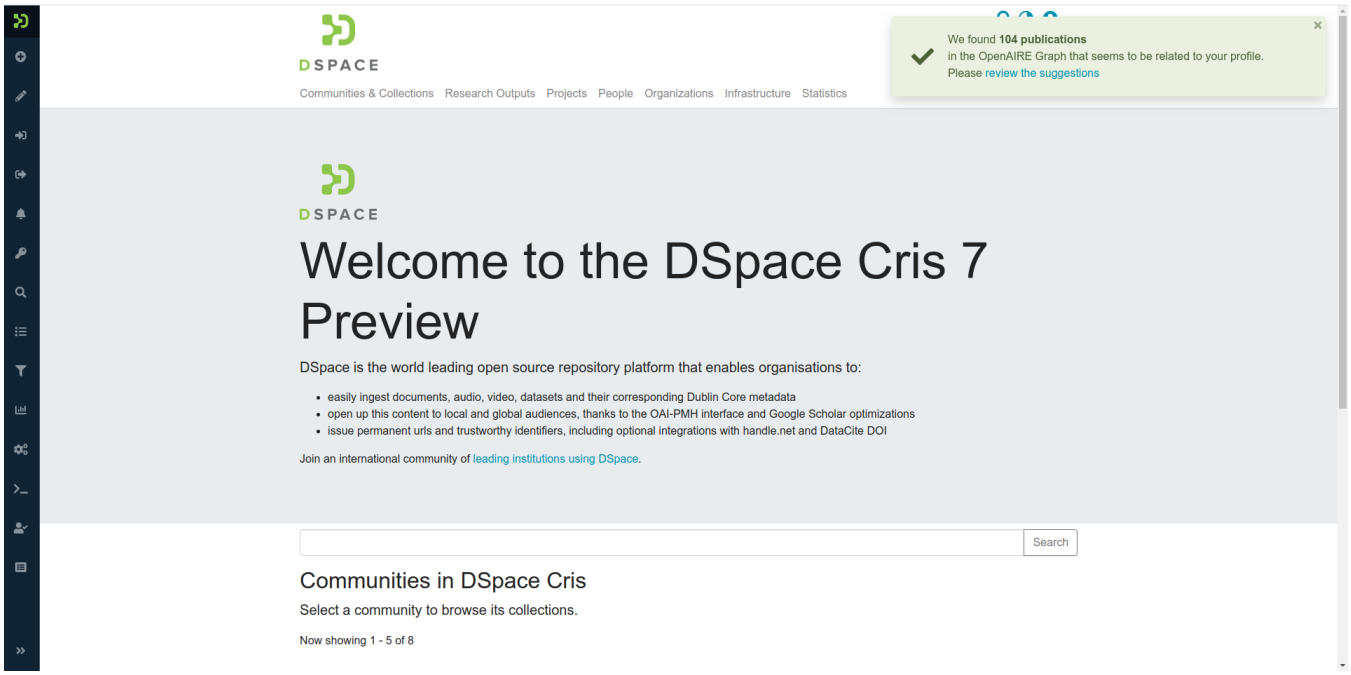
The suggestions list can be sorted by total score descending or ascending (highlighting the weakest candidates).



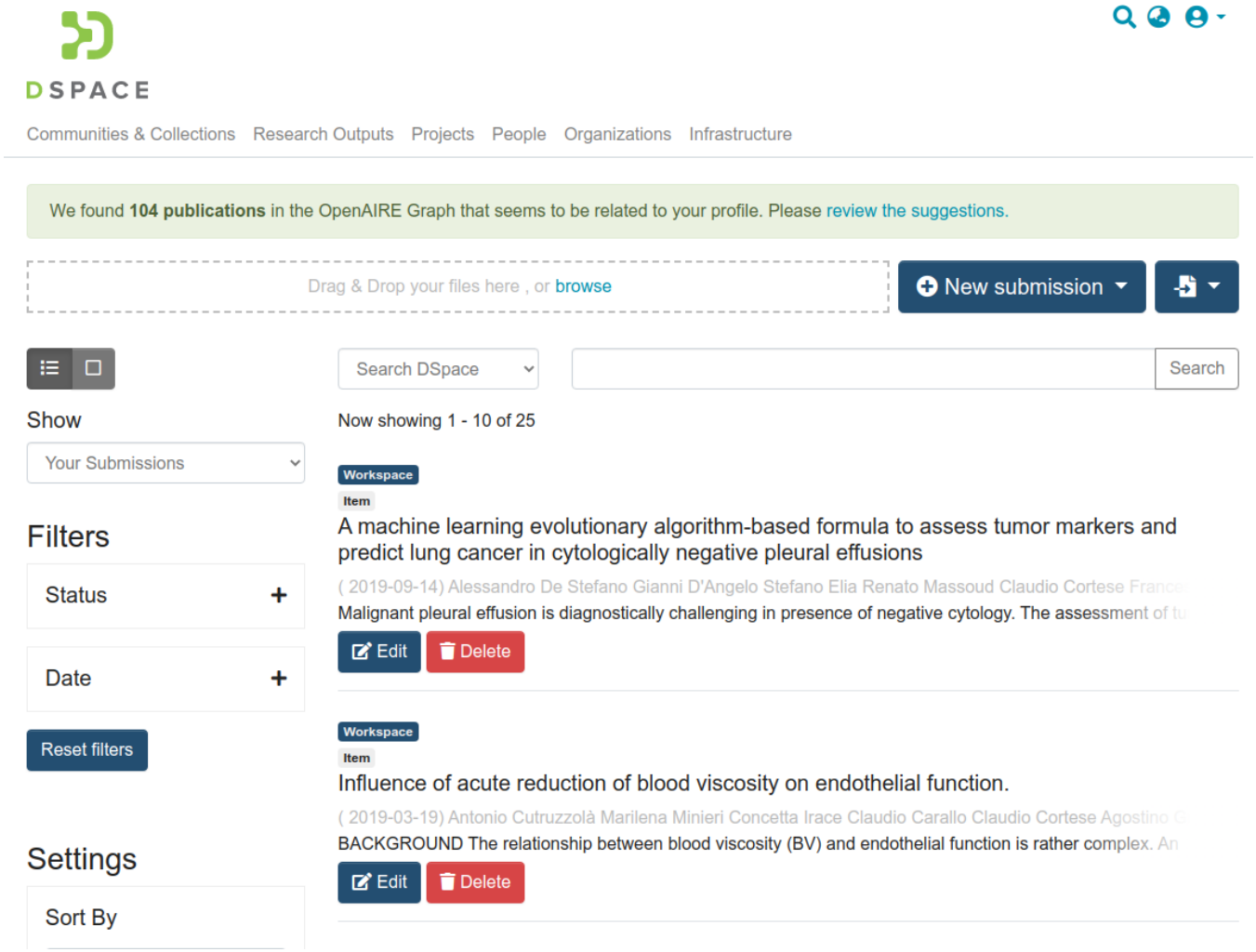
Researcher UI

> *This functionality requires to implement a mechanism to uniquely link user accounts with Person profiles. Such mechanism is implemented out-of-box in DSpace-CRIS. Where the link is not implemented, the Repository Manager UI can still be used.*

The single researcher is also allowed to directly review his suggestions. Upon login he is informed about the availability of suggestions from one or more providers



and can proceed to review the suggestions list in the same way than the Repository Manager, the notification message is also always available at the top of the mydspace



Processing the decisions

The backend is responsible to process the repository manager or researcher decisions taken over the received suggestions. The publication to be imported are processed according to the Import from External Sources normal data flow of DSpace 7. Upon import the suggestion document is removed from the SOLR core, in case of rejection the document is updated flagging it as *rejected* so that it will be not longer proposed to the user.

Rest Contract

Three endpoints have been designed to interact with the publication claim service. They are documented in our REST Contract.

- `/api/integration/suggestionsources` to provide access to summary information about the available suggestion from each source (openaire, orcid, etc.)
- `/api/integration/suggestiontargets` to provide access to summary information about the available suggestions for a specific researcher
- `/api/integration/suggestions` to provide access to the detailed suggestions so that they can be reviewed and managed by the repository manager or the researcher to whom they related

Quality Assurance

Quality Assurance Event

A QA event is an entity stored on the **qaevent** solr collection.


```
{
"source":"openaire",
"event_id":"a542103b9dda29afc320fb36116fc761",
"original_id":"oai:www.openstarts.units.it:123456789/1122",
"title":"The Impact of Longevity and Investment Risk on a Portfolio of Life Insurance Liabilities",

"topic":"ENRICH/MORE/PID",
"trust":1.0,
"message":"{\"pids[0].type\":\"doi\",\"pids[0].value\":\"10.1007/s13385-018-0175-5\"}",
"last_update":"2024-02-29T15:08:43.892Z",
"resource_uuid":"8572c238-18ed-42ed-a471-175acd5d1565"
}
```


A QA Event owns data related to an item archived in the repository: into it's *message* property it may have values for new metadata to be added to the Item.

The entity has properties such as: Source, Topic, Target (resource_uuid) and Message.

Enabling Quality Assurance

All the system described here is turned off and on with the configuration property: **qaevents.enabled**. It defaults to false. To enable this feature, you must set it to "true" in your local.cfg

```
qaevents.enabled
= true
```

Quality Assurance Source

A QA Source is the recognized authority through which the qa event has landed on the repository. Source names are stored into the configuration key: **qaevents.source** . It is defaulted as:

```
qaevents.sources = openaire,
DSpaceUsers, coar-notify
```

Every QA Event must have a recognized source: this means that at time of writing we recognize 3 possible sources.

Quality Assurance Topic

The QA Topic describes the type of event. As the topic is known - it is expected for the *message* content to have a certain format. All the topic managed by the *openaire* source are stored into the configuration key: **qaevents.openaire.import.topic**. it is defaulted as:

```
qaevents.openaire.import.topic =
ENRICH/MISSING/ABSTRACT
# add missing publication id
suggestion

qaevents.openaire.import.topic =
ENRICH/MISSING/PID
# add more publication id suggestion

qaevents.openaire.import.topic =
ENRICH/MORE/PID
# add missing project suggestion

qaevents.openaire.import.topic =
ENRICH/MISSING/PROJECT
# add more project suggestion

qaevents.openaire.import.topic =
ENRICH/MORE/PROJECT
# add more review

qaevents.openaire.import.topic =
ENRICH/MORE/REVIEW
# add more endorsement

qaevents.openaire.import.topic =
ENRICH/MORE/ENDORSEMENT
# add more release/relationship

qaevents.openaire.import.topic =
ENRICH/MORE/LINK
```

Quality Assurance Target

the target of a qa event is an item archived into the repository. The qa event solr document owns the item system uuid into the *resource_uuid* property.

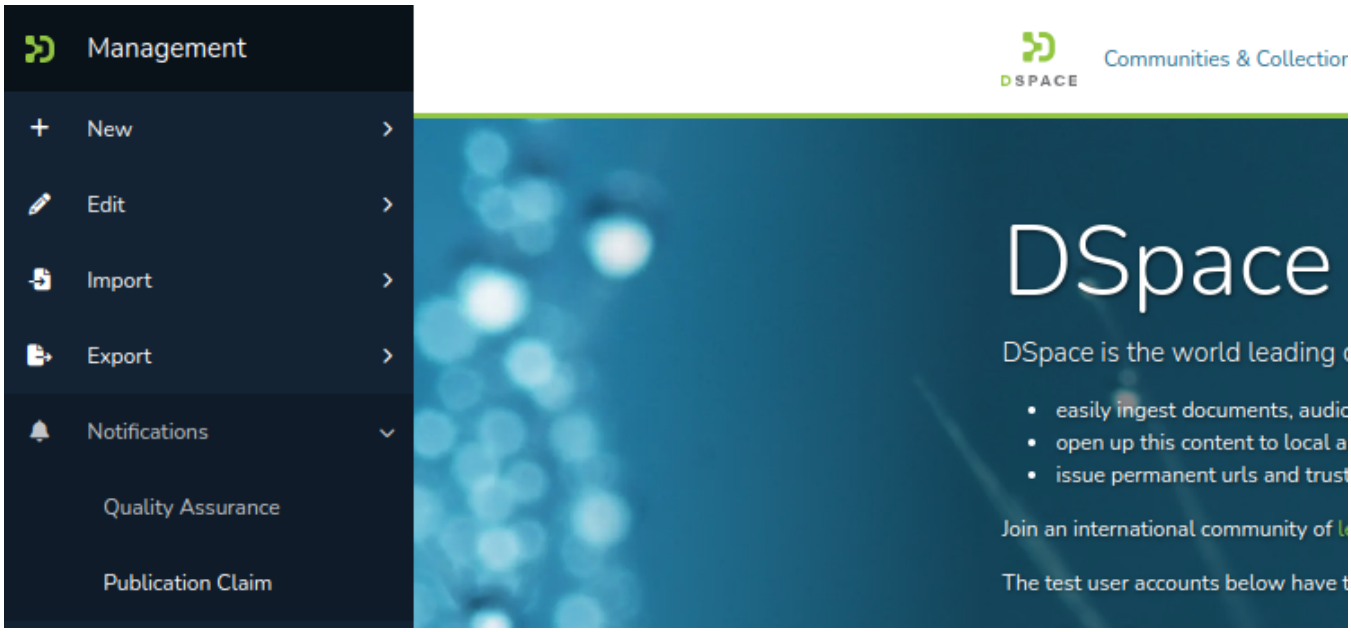
Quality Assurance Message

the message of a qa event is a json-formatted string that contains all the new values related to the item.

Quality Assurance Management

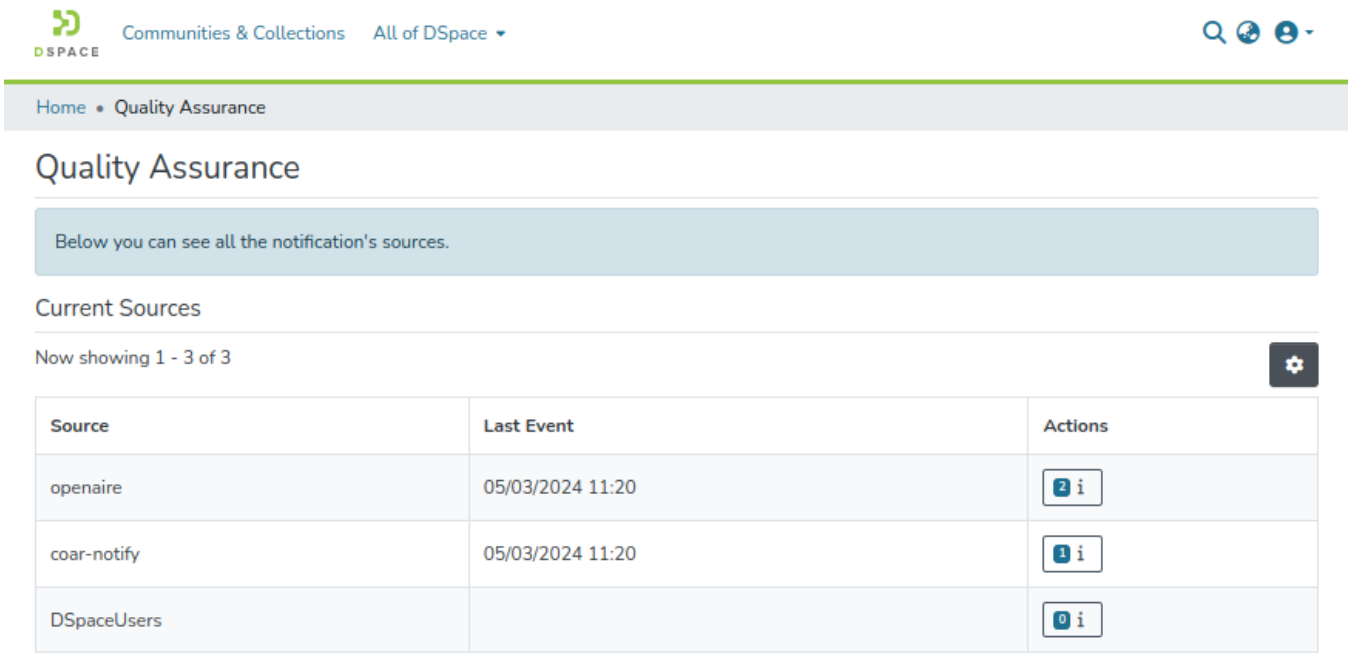
A qa events lands on the repository according to its source. for example: Openaire has an import batch for qaevents in json format to be loaded, Coar-Notify creates new qa event solr document when certain ldn messages are received.

QA Events are visible at Menu -< Notifications Quality Assurance



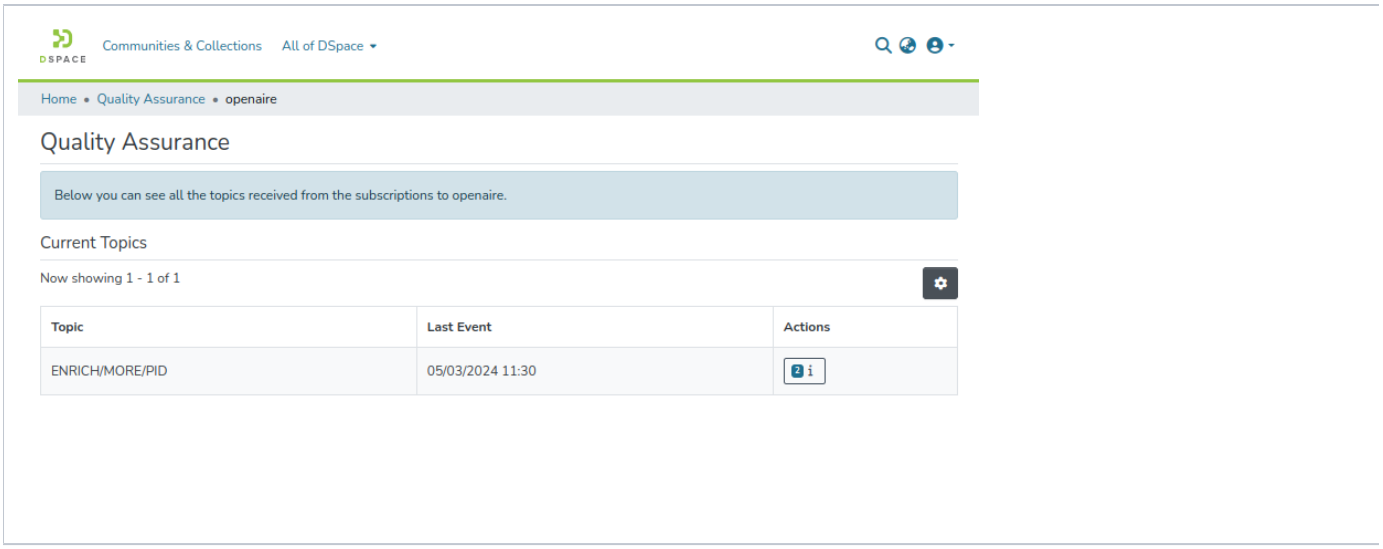
This is the main page of the Quality Assurance.

The first column shows the Sources the user can see.
The last column shows the counter of the events the user can see.



By clicking at the counter the source page is shown. Here there's the list of all the qa events the user can see,

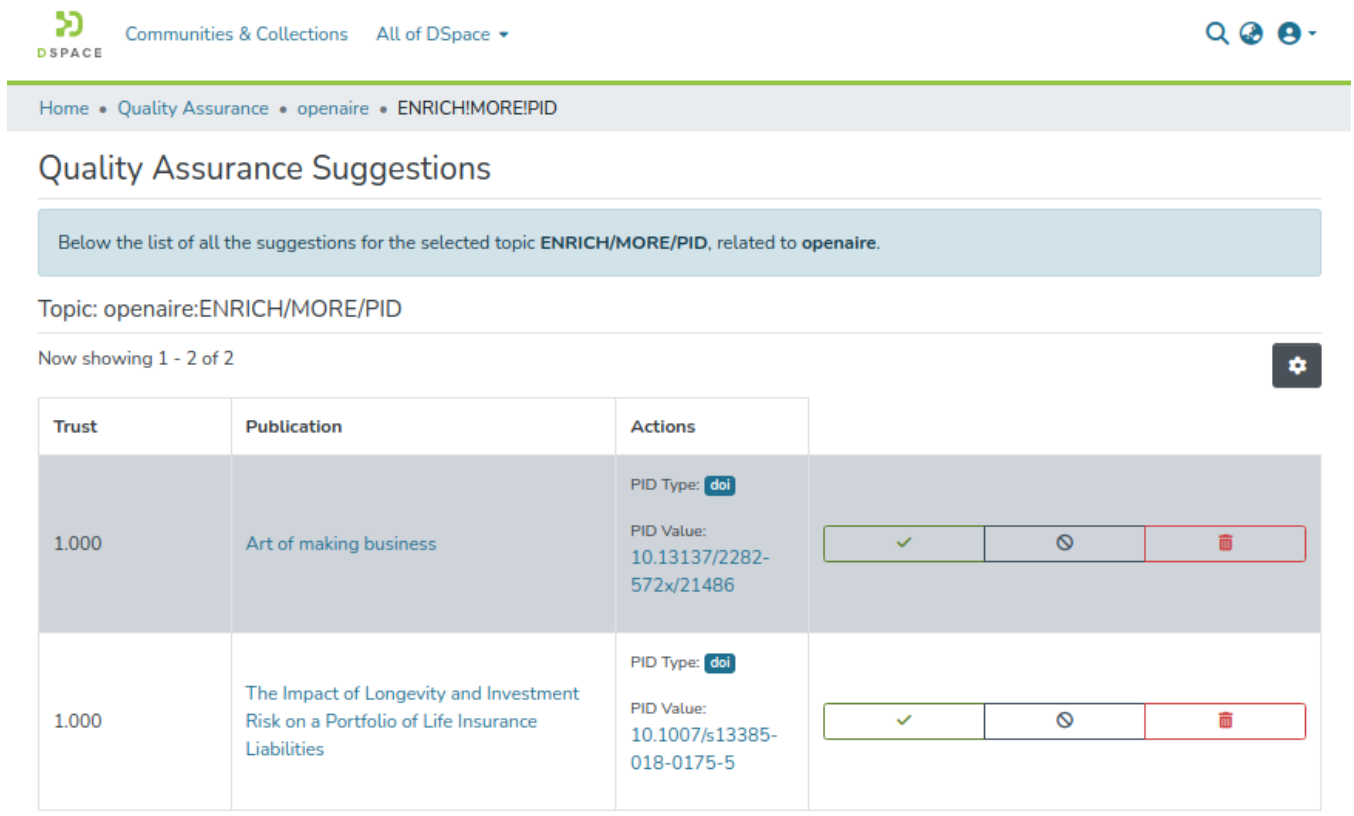
grouped by their topic



Every qa event can be accepted, discarded or rejected.

When accepted an action is triggered. There's a configured correspondence between the topic of the event and an action to be performed.

When discarded or rejected the solr document of the qa event is deleted.



Processing the decision

Every decision been made from this panel (about qa events) is reported on the outside through an http post call. It is directed to all the receivers configured at the configuration key: **qaevents.** + *source* + **.acknowledge-url**

it contains a simple json as payload as:

```
{
"eventId":"a542103b9dda29afc320fb36116fc761",
"status":"accepted|discarded|rejected"
}
```

On the Repository side what is performed is encapsulated in a JAVA class specialized to deal with a specific TOPIC. The `/config/spring/api/qaevents.xml` spring configuration file map each TOPIC to a specific implementation

```
<bean id="org.dspace.qaevent.service.QAEventActionService" class="org.dspace.qaevent.service.impl.
QAEventActionServiceImpl">
        <property name="topicsToActions">
            <map>
                <!--The key are the TOPIC, the value must be a valid implementation of the
                    org.dspace.qaevent.QAEventAction interface -->
               <entry value-ref="ProjectLinkedEntityAction">
                  <key><util:constant static-field="org.dspace.qaevent.QANotifyPatterns.TOPIC_ENRICH_MORE_PROJECT"
/></key>
               </entry>
               <entry value-ref="ProjectLinkedEntityAction">
                  <key><util:constant static-field="org.dspace.qaevent.QANotifyPatterns.
TOPIC_ENRICH_MISSING_PROJECT"/></key>
               </entry>
               <entry value-ref="AbstractMetadataAction">
                  <key><util:constant static-field="org.dspace.qaevent.QANotifyPatterns.
TOPIC_ENRICH_MISSING_ABSTRACT"/></key>
               </entry>
               <entry value-ref="AddReviewMetadataAction">
                  <key><util:constant static-field="org.dspace.qaevent.QANotifyPatterns.TOPIC_ENRICH_MORE_REVIEW"
/></key>
               </entry>
               <entry value-ref="AddEndorsedMetadataAction">
                  <key><util:constant static-field="org.dspace.qaevent.QANotifyPatterns.
TOPIC_ENRICH_MORE_ENDORSEMENT"/></key>
               </entry>
               <entry value-ref="PIDMetadataAction">
                  <key><util:constant static-field="org.dspace.qaevent.QANotifyPatterns.TOPIC_ENRICH_MORE_PID"/><
/key>
               </entry>
               <entry value-ref="PIDMetadataAction">
                  <key><util:constant static-field="org.dspace.qaevent.QANotifyPatterns.TOPIC_ENRICH_MISSING_PID"
/></key>
               </entry>
               <entry value-ref="AddLinkMetadataAction">
                  <key><util:constant static-field="org.dspace.qaevent.QANotifyPatterns.TOPIC_ENRICH_MORE_LINK"/><
/key>
               </entry>
            </map>
        </property>
    </bean>
```

Each implementation allows to configure additional parameters to deal with the event as needed, ranging from the simple definition of the metadata to use to save the information as in the case of the Abstract related events

```
    <bean id="AbstractMetadataAction" class="org.dspace.qaevent.action.QAOpenaireSimpleMetadataAction">
       <property name="metadata" value="dc.description.abstract" />
    </bean>
```

to a dynamic mapping used for SUBJECT and PID related events

```
    <!-- Add a new identifier to the given item, using the defined types mapping -->
    <bean id="PIDMetadataAction" class="org.dspace.qaevent.action.QAOpenaireMetadataMapAction">
        <property name="types">
            <map>
   <!--The key are the type of identifier (or subject) reported in the message, the value is the metadata in
       the linked entity where the information should be stored -->
              <entry key="default" value="dc.identifier.other" />
              <!-- <entry key="doi" value="dc.identifier.doi" />
              <entry key="pmid" value="dc.identifier.pmid" /> -->
            </map>
        </property>
    </bean>
```

to the definition of the metadata used in linked entity for Project related events

```
    <!-- This action bind the publication to the project, otherwise if the project has not been specified,
     create a new project with the available data and then bind it to the publication -->
    <bean id="ProjectLinkedEntityAction" class="org.dspace.qaevent.action.QAEntityOpenaireMetadataAction">
       <!-- which metadata will hold the relation between the publication and the project -->
       <property name="relation" value="isPublicationOfProject" />
       <!-- the type of local entity used to store the project details -->
       <property name="entityType" value="Project" />
       <property name="entityMetadata">
           <map>
   <!--The key are the json path of qa message, the value is the metadata in
       the linked entity where the information should be stored -->
              <!--  <entry key="acronym" value="" /> -->
              <entry key="code" value="dc.identifier" />
              <!--  <entry key="funder" value="oairecerif.funder" /> -->
              <entry key="title" value="dc.title" />
              <!--  <entry key="fundingProgram" value="oairecerif.fundingProgram" /> -->
              <!--  <entry key="openaireId" value="oairecerif.funding.identifier" /> -->
           </map>
       </property>
    </bean>
```

Quality Assurance Security

QA Events are filtered by a security system. The system is defined on the spring context (see qaevents.xml in specific) on the QAEVentSecurityService bean. The default security system is set as only DSpace administrators can see and manage all sources. A security system can be assigned to none, one or more qa sources. The default one is used for Openaire source. Two more security system are defined at time of writing: one for each managed source: DSpaceUsers and coar-notify. Security filters help to hide qa events to the logged user. IE coar-notify security filter allow users to see qa events only to administrators and to targeted-item submitters.

COAR Notify Integration

Coar-Notify is a source managed for Quality Assurance Events. It has been introduced with the COAR Notify integration, mainly due to the integration of LDN Message exchange.

The processing of incoming LDN Messages is configured and it may create new QA Events solr documents - surely with the **source="coar-notify"**.

For more information about COAR Notify, see **COAR Notify**.

Configuration

When an incoming LDN Message is processed a QA E vent is built. The event message content is taken from the content of the corresponding LDN message. QA Event creation

Security

The QA security applied to the coar-notify source is the "submitterQASecurity" - which shows all events to the administrators and to the submitters.

OpenAIRE Integration

Data Correction

> *Scenario : A repository manager of a repository indexed in OpenAIRE can subscribe the event for Missed/More PIDs and Project links in the Content Provider Dashboard using “a repository callback” as notification mechanism instead of the current email alert. They login in the repository and see the list of events received, among others one publication that has a PMID that was unknown to the repository and a link to a project. They click on the “accept the suggestion” button and the new information is stored in the local record. OpenAIRE could “flag” the data as confirmed.*

The goal of the Data Correction service is to support the scenario above.

As the OpenAIRE Content Provider Dashboard doesn't allow yet to create a subscription setting up a callback mechanism, we agree with the OpenAIRE team to read the data generated by openAIRE's Notification Broker Service from a JSON file postponing to the last phase of the project the discussion and the implementation about the delivery mechanism (polling new versions from a stable URL, receive it as payload of a repository URL, etc.).

Data source

The JSON file contains an array of JSON Events, where each event has the following structure

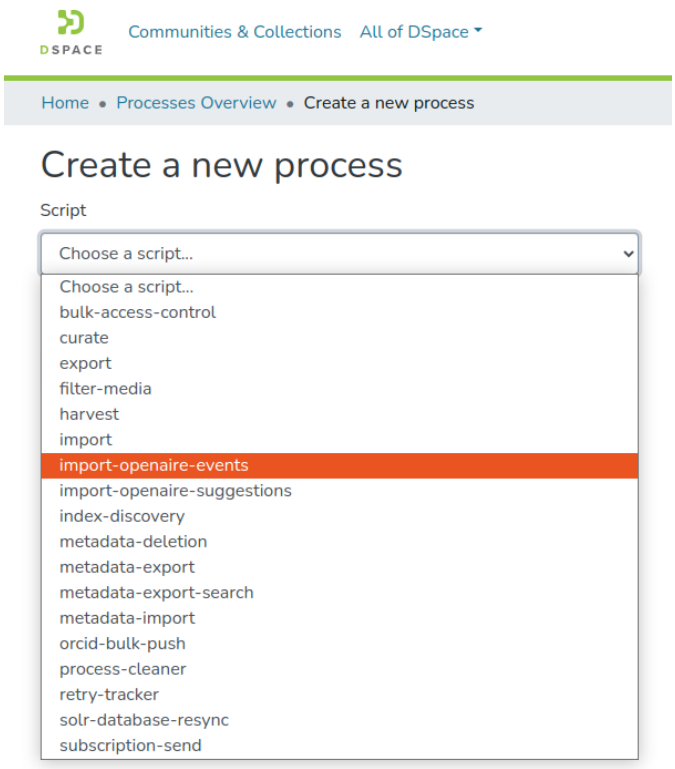
```
    {
        "originalId": "oai:www.openstarts.units.it:10077/21838",
        "title": "Egypt, crossroad of translations and literary interweavings (3rd-6th centuries). A
reconsideration of earlier Coptic literature",
        "topic": "ENRICH/MORE/PROJECT",
        "trust": 1.0,
        "message": {
            "projects[0].acronym": "PAThs",
            "projects[0].code": "687567",
            "projects[0].funder": "EC",
            "projects[0].fundingProgram": "H2020",
            "projects[0].jurisdiction": "EU",
            "projects[0].openaireId": "40|corda__h2020::6e32f5eb912688f2424c68b851483ea4",
            "projects[0].title": "Tracking Papyrus and Parchment Paths: An Archaeological Atlas of Coptic
Literature.\nLiterary Texts in their Geographical Context: Production, Copying, Usage, Dissemination and Storage"
        }
    }
```

please note that the message sub-object depends on the event TOPIC. A more complete set of sample events can be seen here: qaevents-sample.json

The java class `org.dspace.app.qaevent.qaeventsRunnableCli` provides a convenient method to process this json file loading the data in a dedicated new DSpace SOLR Core named **qaevent**, to use it run from the dspace installation bin folder

```
./dspace import-qaevents -f <path-to-the-json-file>
```

the same script is also available via the administrative runnable process UI



The `config/modules/qaevents.cfg` file allows to configure witch Topic should be processed, indeed some Topics could have no configured action on the repository

```
qaevents.openaire.import.topic = ENRICH/MISSING/ABSTRACT
qaevents.openaire.import.topic = ENRICH/MORE/PID
qaevents.openaire.import.topic = ...
```

and a list of URLs to acknowledge the decision made by the Repository Manager via the DSpace UI

```
qaevents.openaire.acknowledge-url = https://httpdump.io/...
```

Such configuration file is also expected in future to hold settings related to the delivery mechanism (such as the URL from where the json file can be download, the credentials to use, etc.)

The qaevent core has the following structure

```
<fields>
    <field name="event_id" type="string" indexed="true" stored="true" omitNorms="true" />
    <field name="original_id" type="string" indexed="true" stored="true" omitNorms="true" />
    <field name="title" type="string" indexed="true" stored="true" omitNorms="true" />
    <field name="topic" type="string" indexed="true" stored="true" omitNorms="true" />
    <field name="trust" type="double" indexed="true" stored="true" omitNorms="true" />
    <field name="message" type="string" indexed="true" stored="true" omitNorms="true" />
    <field name="resource_uuid" type="string" indexed="true" stored="true" omitNorms="true" />
    <field name="related_uuid" type="string" indexed="true" stored="true" omitNorms="true" />
    <field name="last_update" type="date" indexed="true" stored="true" omitNorms="true" />
</fields>
<uniqueKey>event_id</uniqueKey>
```

the `event_id` is currently generated on the repository side as an hash of the business information included in the event itself but it is envisioned that such information will be made available by openAIRE directly in the json file so that feedback from the Repository can linked back to the original event and further processed.

The `related_uuid` field contains the uuid of the related object that has been associated with the correction suggestion, this is the case for the PROJECT related TOPICS where a link between the publication and a project should be established. In the case the suggested project can be found in the system, the `related_uuid` field will hold its internal identifier otherwise the user will be allowed to created on the fly a new item also for the project and connect it to the publication item with a single click.

Two REST endpoints have been developed to expose the data so collected

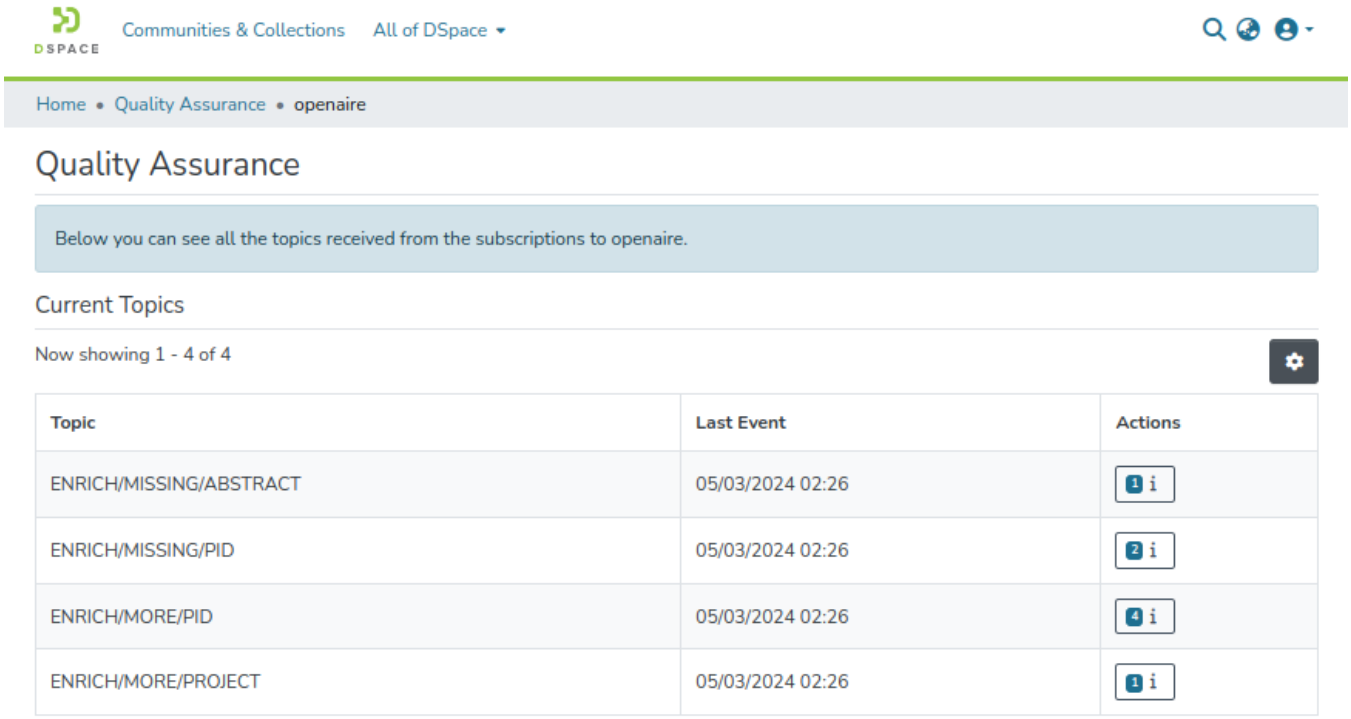
- `/api/integration/qualityassurancetopic` to provide access to summary information about the available topic and number of events to deal with
- `/api/integration/qualityassuranceevent` to provide access to the detailed events so that they can be reviewed and managed by the repository manager

The detailed REST contract for such endpoints are available on the 4Science Rest7Contract repository and embedded at the bottom of the page for easy reference.

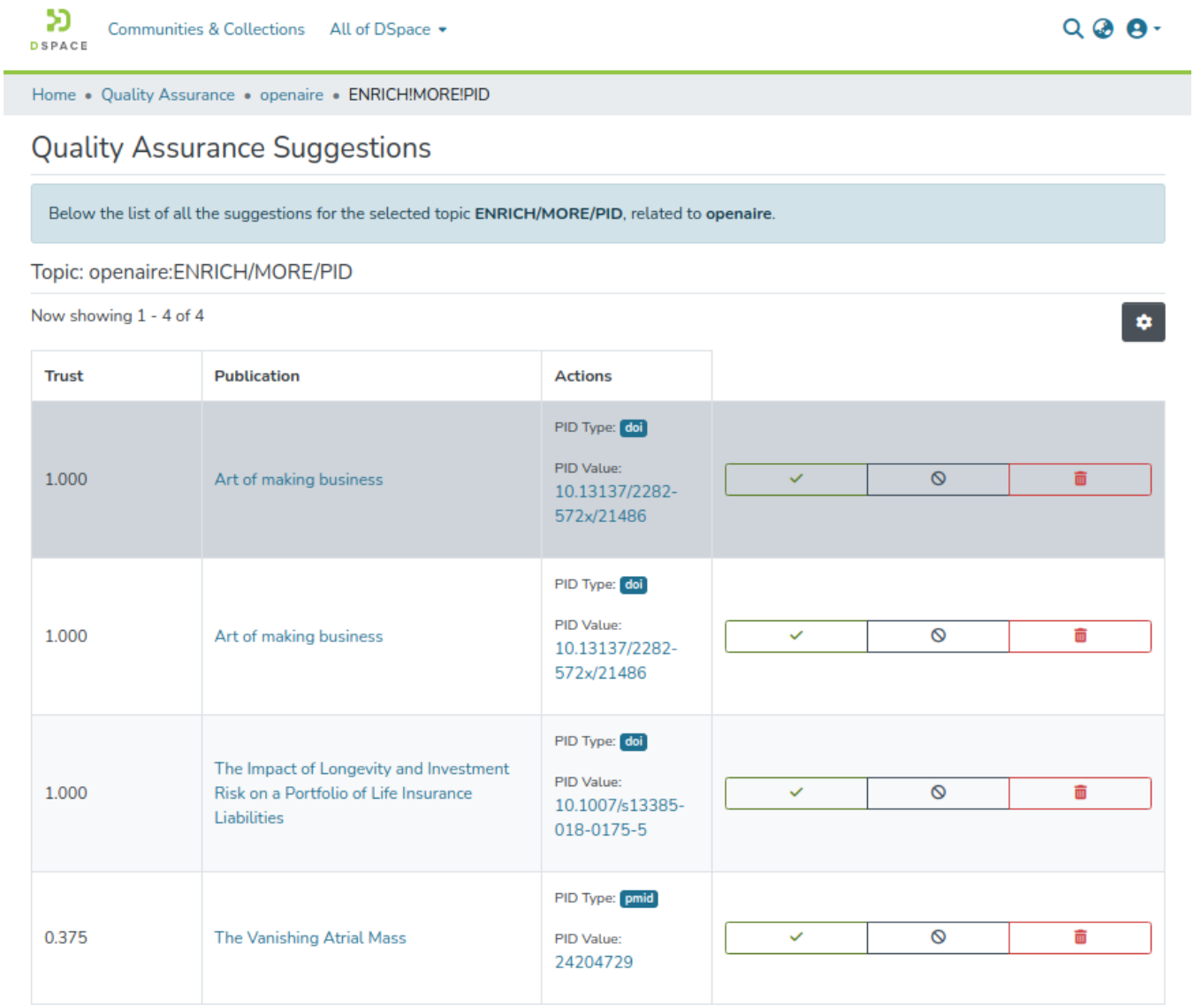
Repository Manager UI

The resulting UI is accessible from the administrative menu - if the configuration key is true: **qaevents.enabled** (it can be found onto *qaevents.cfg* file and is defaulted as **false**). As entry point for the features a “Notifications” menu entry has been added to the DSpace administrative menu, from where the repository manager will be able to manage the OpenAIRE subscription and access the details of received events.

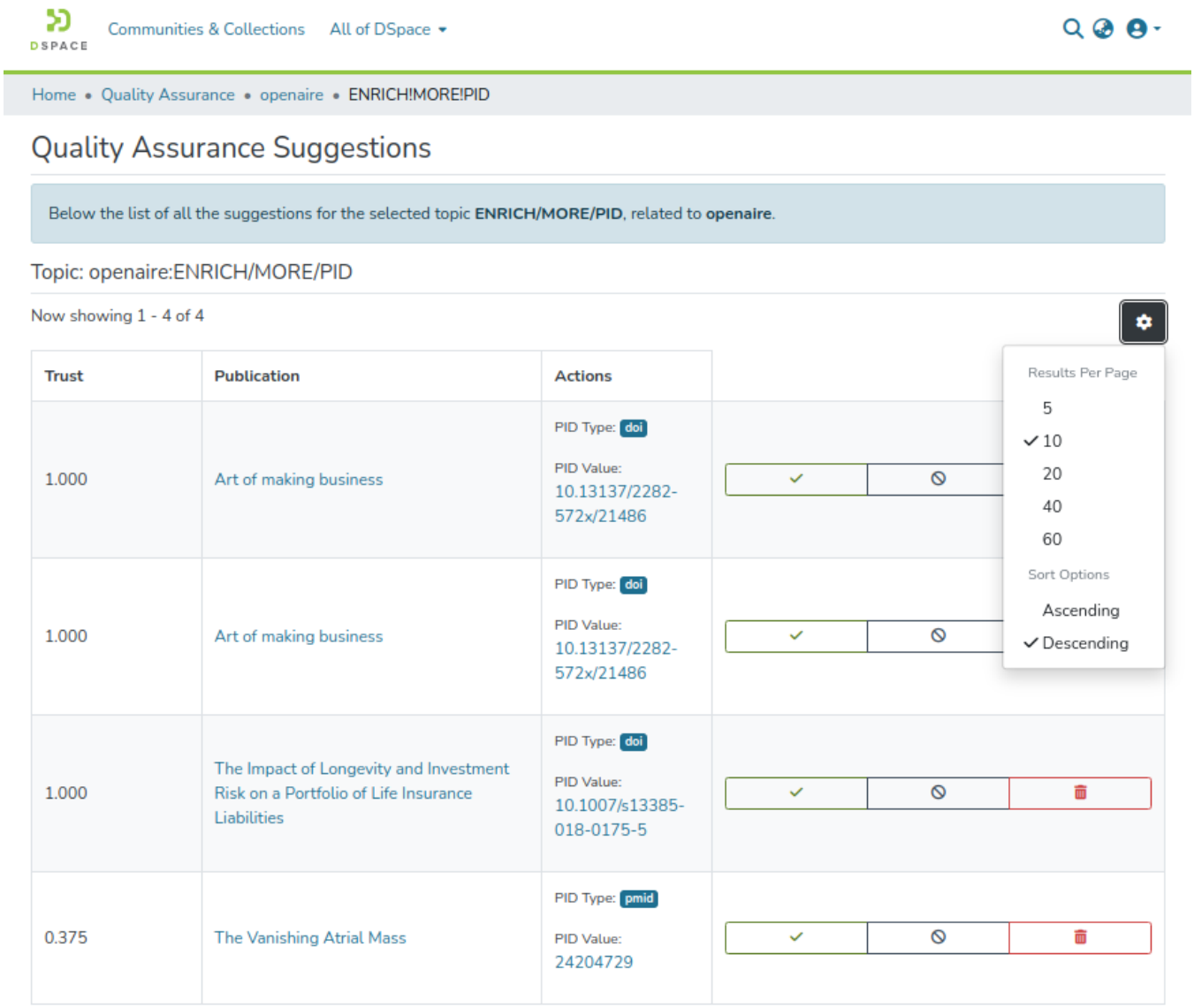
The main page list the topics found in the events loaded in the system



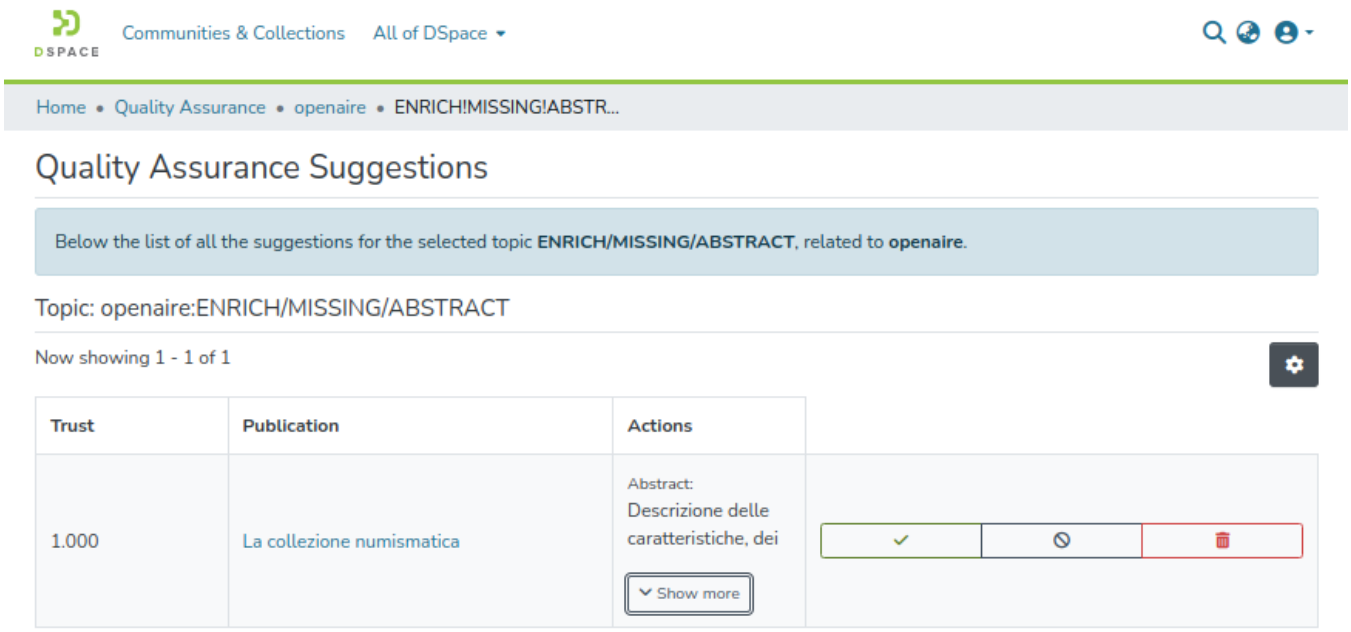
By default the system sort the events within a topic by trust descending (most accurate correction first)



but it is also possible to revert the direction

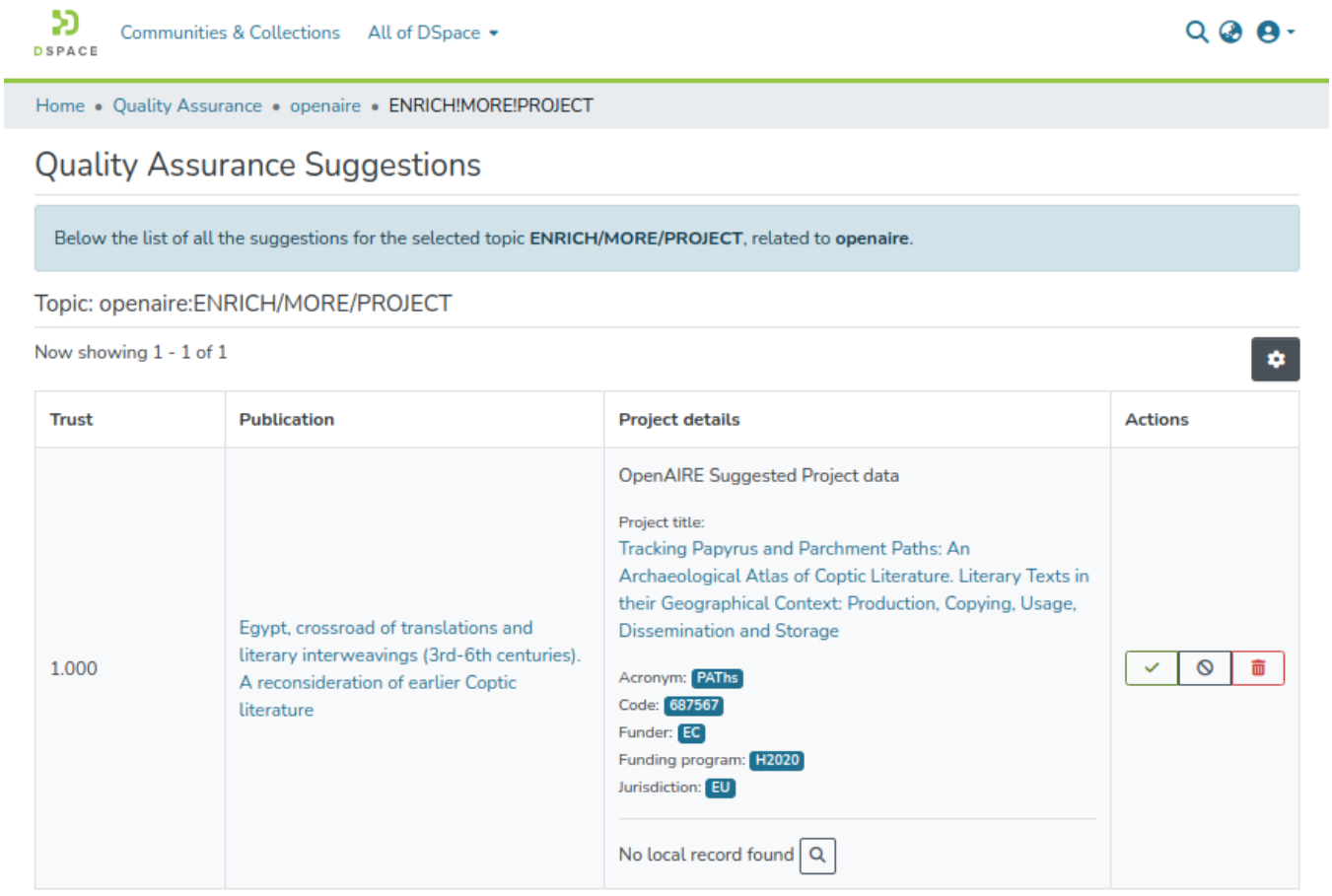


In the detail view of events in a specific topic links always open in a new tab so that the repository manager can quickly check the details without loosing the overview

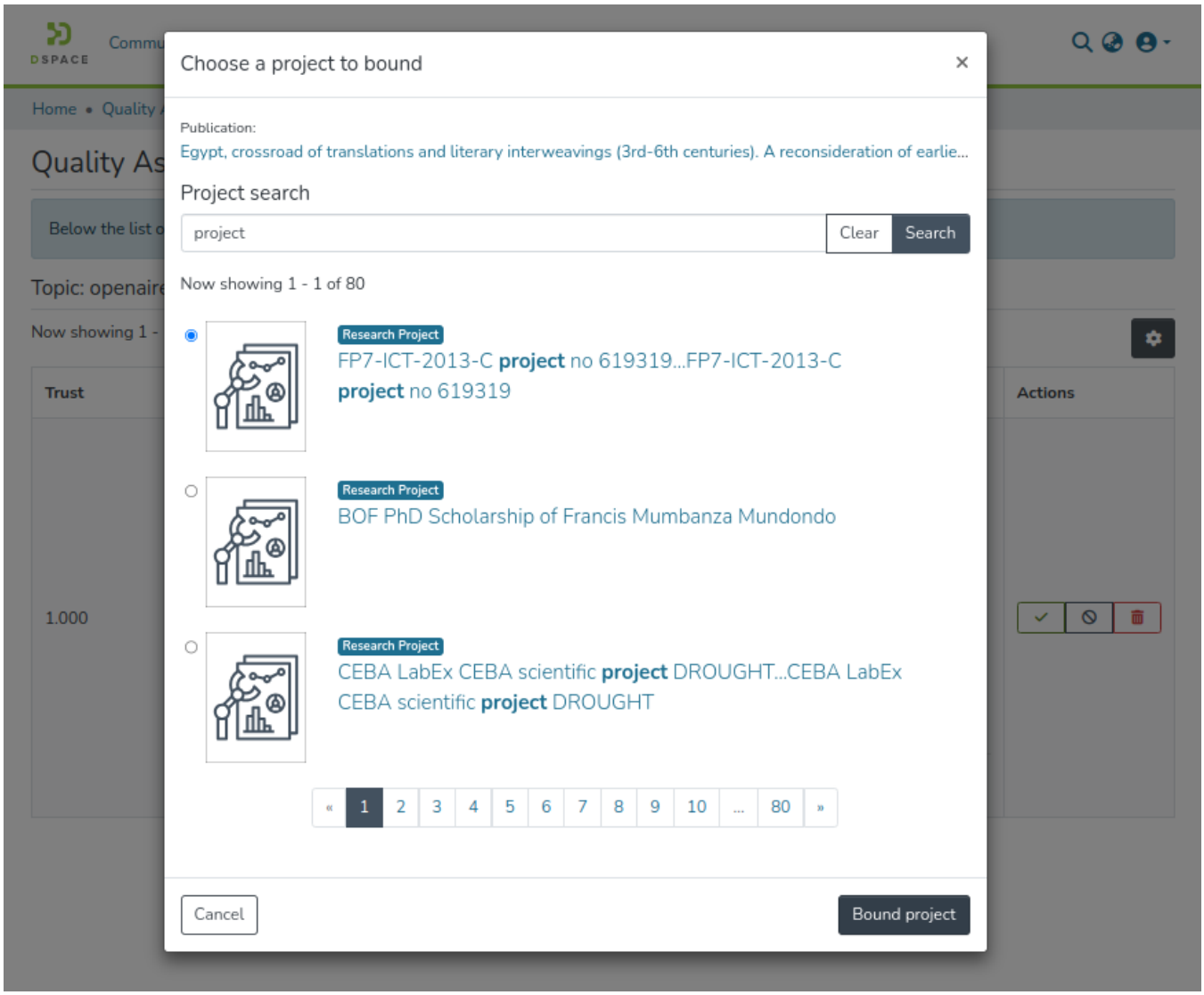


Below a screen of possible missing abstract events, where the repository manager will be able to check the current local publication record clicking on the title and scroll the abstract reported by OpenAIRE. Accepting the suggestion, the local record will be enriched with this extra information. The Ignore suggestions button is instead intent to be used to discard a notification without flagging it as wrong. This is important because the OpenAIRE Graph process the data from the repository not in real-time so it can happen that a local record has been updated recently with information not yet known to OpenAIRE. In such scenarios it could be possible that the repository manager prefers to keep the local version but this should be not reported to OpenAIRE a wrong suggestion as this feedback can be used to improve the OpenAIRE guessing capabilities. In contrast a wrong suggestion should be rejected so that OpenAIRE can learn from that.

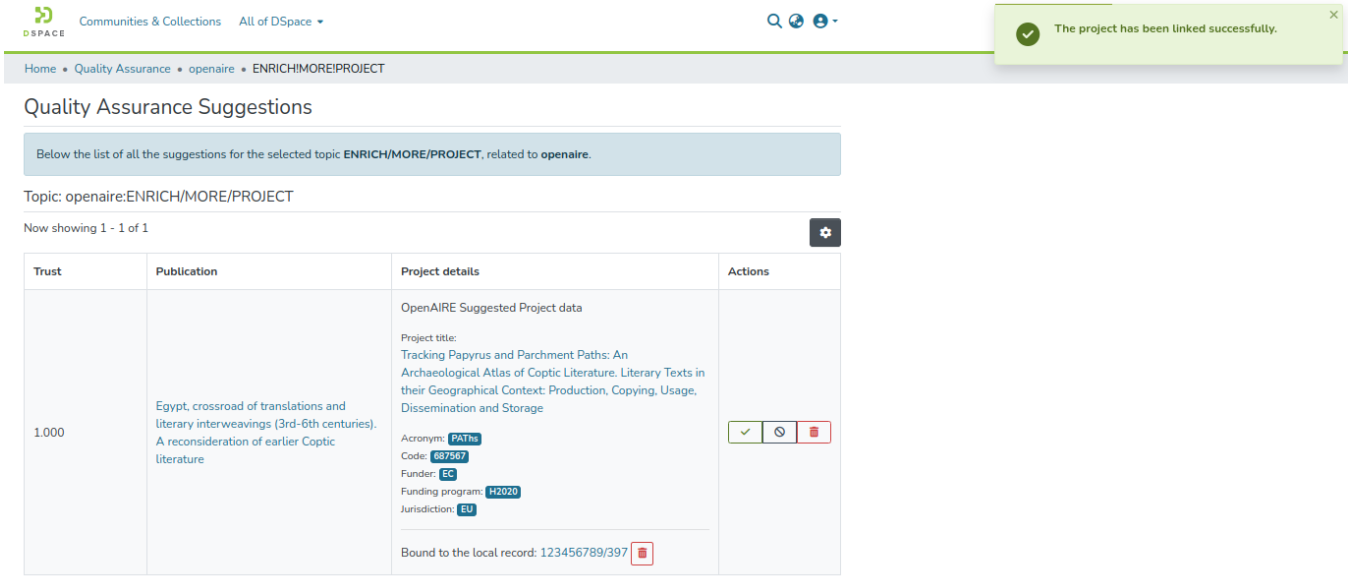
For PROJECT related events, alternative additional actions are needed. This is usually the case for information that is related to linked entities that can be tracked on the local repository as flat metadata (in such case the “abstract approach UI” will be used) or as individual entity. In this later case the below screen applies:



The system will attempt to identify a local record for the information reported by OpenAIRE (the project) and will offer to the repository manager the option to manually lookup the record or fix the automatic match



if the related project is found in the system the repository manager can proceed to accept the correction linking the publication to the local copy of the project otherwise it is possible to import and connect the project in one click as shown in the first project related screen above



For PID related events, the system offers where available (doi, handle, pmid, pmc, arXiv, NCID, urn/url) the resolution of the identifier to a details page

Quality Assurance Suggestions

Below the list of all the suggestions for the selected topic **ENRICH/MORE/PID**, related to **openaire**.

Topic: openaire:ENRICH/MORE/PID

Now showing 1 - 4 of 4

Trust	Publication	Actions
1.000	Art of making business	PID Type: doi PID Value: 10.13137/2282-572x/21486
1.000	Art of making business	PID Type: doi PID Value: 10.13137/2282-572x/21486
1.000	The Impact of Longevity and Investment Risk on a Portfolio of Life Insurance Liabilities	PID Type: doi PID Value: 10.1007/s13385-018-0175-5
0.375	The Vanishing Atrial Mass	PID Type: pmid PID Value: 24204729

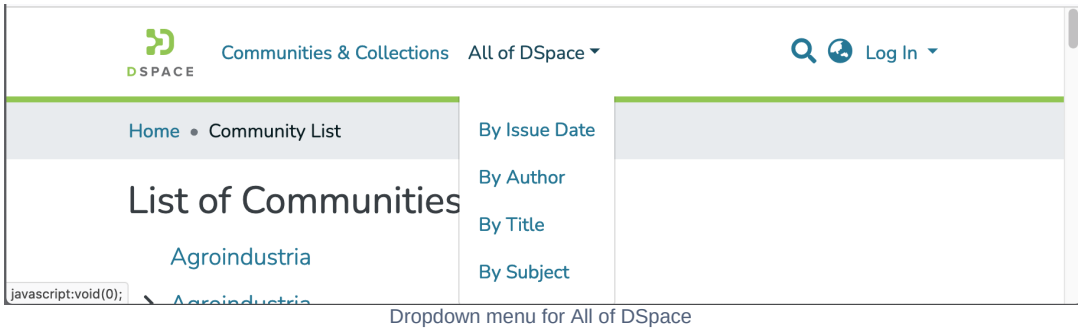
Menus

- Communities & Collections
- All of DSpace
  - By Issue Date
  - By Author
  - By Title
  - By Subject
- Statistics
- Search box
- Language
- Log In
  - Profile
  - MyDSpace

Communities & Collections

This is an alphabetical and hierarchical list of communities in the repository. Communities may contain sub-communities and collections. Use the left chevrons to expand the hierarchy.

All of DSpace



Dropdown menu for All of DSpace

Browse by Issue Date, Author, Title, or Subject. Use the gear icon to set the sorting and number of results per page.

By Issue Date

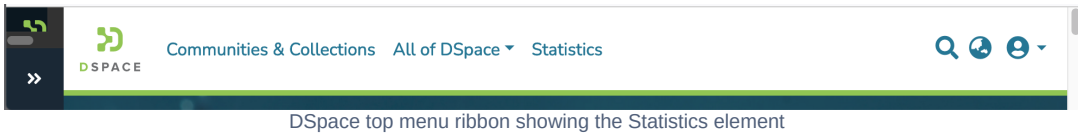
By Author

By Title

By Subject

Statistics

(Once logged in with correct access)



DSpace top menu ribbon showing the Statistics element

On the Home page, this lists the 10 items with most visits to the item page. (This does not count the downloads of the files contained in the item.)

On a Community page, this lists the total cumulative visits to the community's main page (not all the collections and items that community contains), visits to the community main page broken out by month for the past 7 months, and by top cumulative country and city views.

On a Collection page, this lists the total cumulative visits to the collection's main page (not all the items that collection contains), visits to the collection page broken out by month for the past 7 months, and by top cumulative country and city views.

On an Item page, this lists the total cumulative visits to the item page, (This does not count the downloads of the files contained in the item.), visits to that page broken out by month for the past 7 months, and by top cumulative country and city views.

Search box

Use this search box to retrieve search results that can be further refined by search filters, such as date, subject, author, and item type.

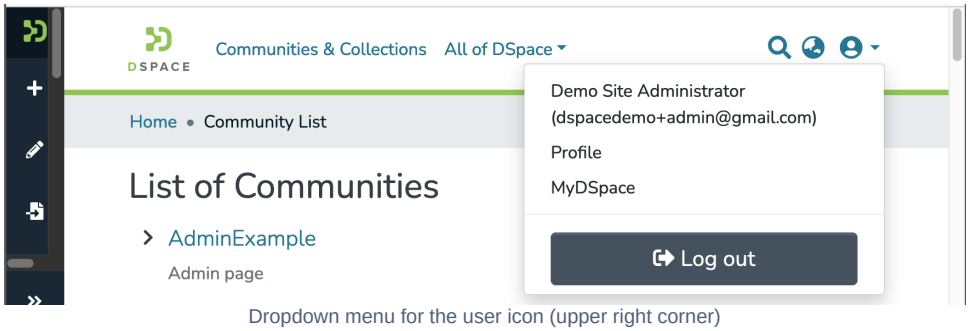
See Search - Advanced for tips for creating Boolean searches.

Language

Choose your preferred supported language for the DSpace system. Repository files and metadata will be in their source language.

Log In

(Once logged in)



Dropdown menu for the user icon (upper right corner)

Profile

If enabled in the repository, one can set up a Researcher Profile.

One can reset their password and view your group memberships.

MyDSpace

View or edit your submissions:

- Items that you are preparing (Status=Workspace),
- Items that are in the approval process (Status=Workflow),
- Items that have been approved and completed (Status=Archived).

Registry management

Info for repository managers, covering topics such as metadata registry management and format registry management.

- Metadata Registry Management

Metadata Registry Management

The metadata registry maintains a list of various metadata schemas in DSpace. These metadata schemas consist of different metadata elements. However, DSpace requires the qualified Dublin Core schema.

- Audience
- Create Metadata Registry
- Metadata Registry Management
- Delete Metadata Schema

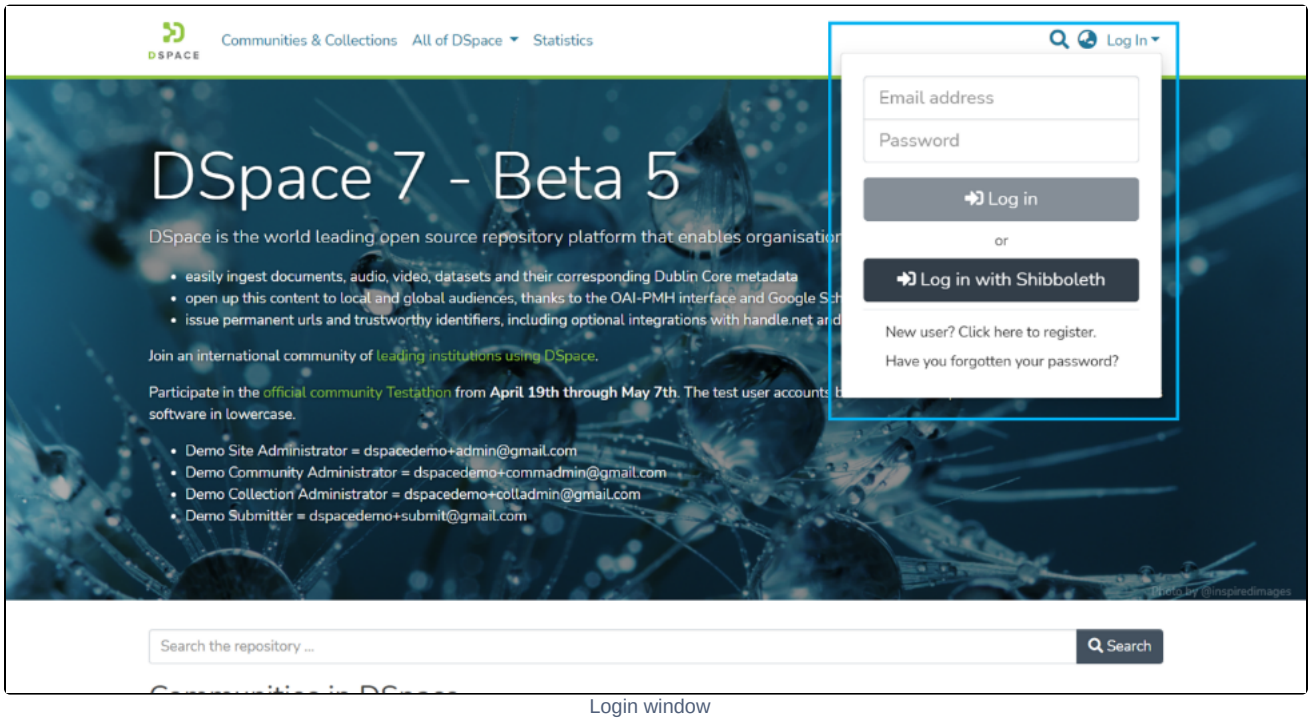
Audience

1. Repository Administrator
2. Community Administrator

Create Metadata Registry

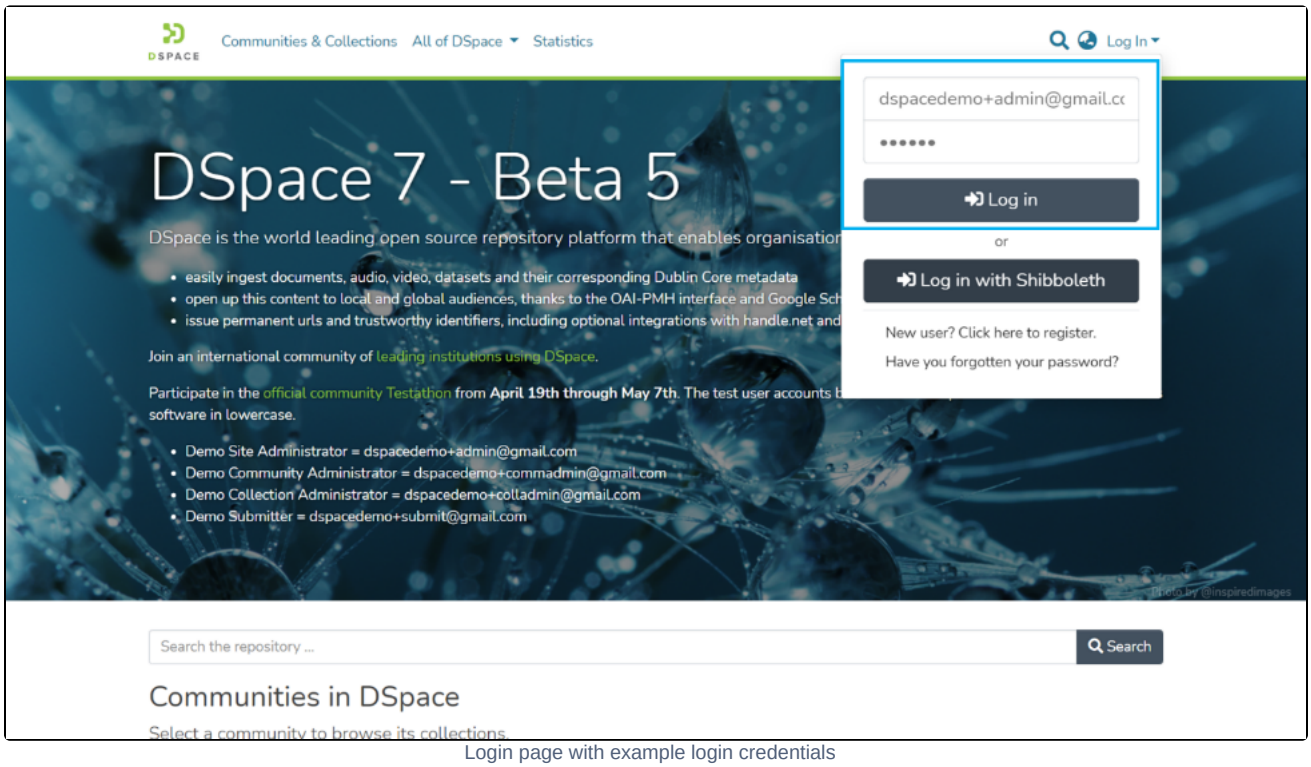
This process is for System and Community Administrator user profiles.

Step 1: Click on the “Log In” link appearing at the top right corner of the DSpace home page, and the pop-up will open, as illustrated in the below screen.



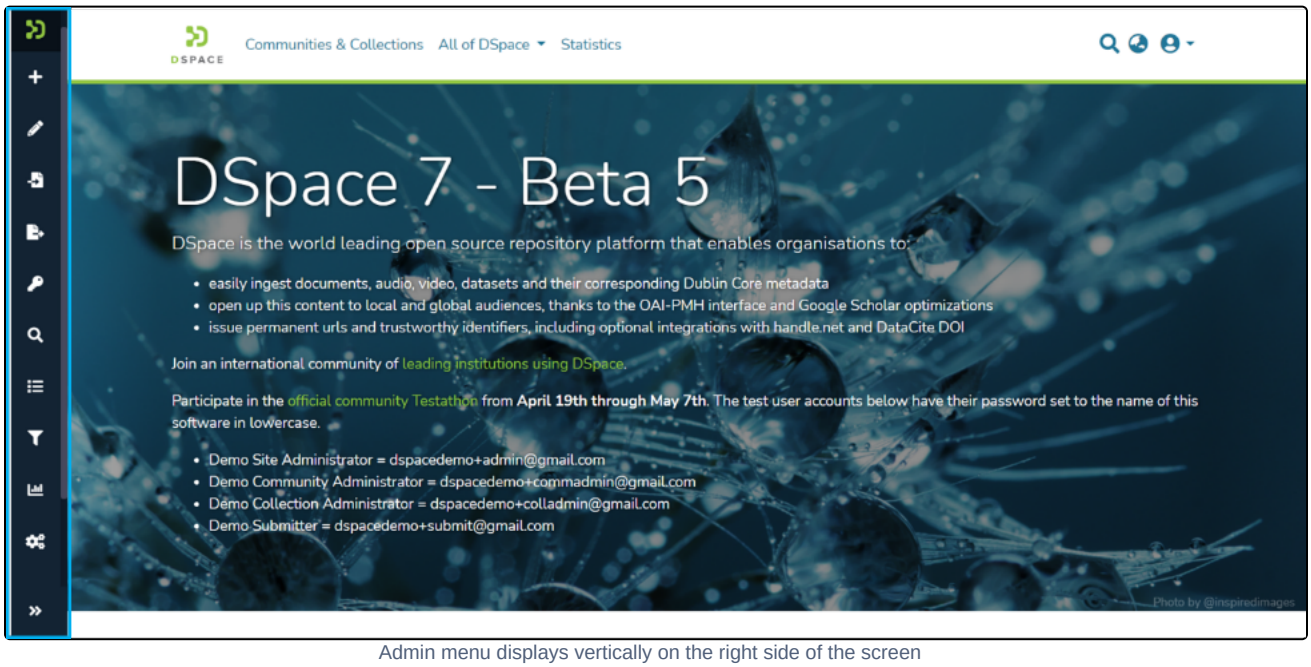
Login window

Step 2: Enter your user id and password and click on the login button for logging in to DSpace.



Login page with example login credentials

Step 3: Users with administrative rights will see the admin menu on the left-hand side of the screen, as shown in the below illustration.



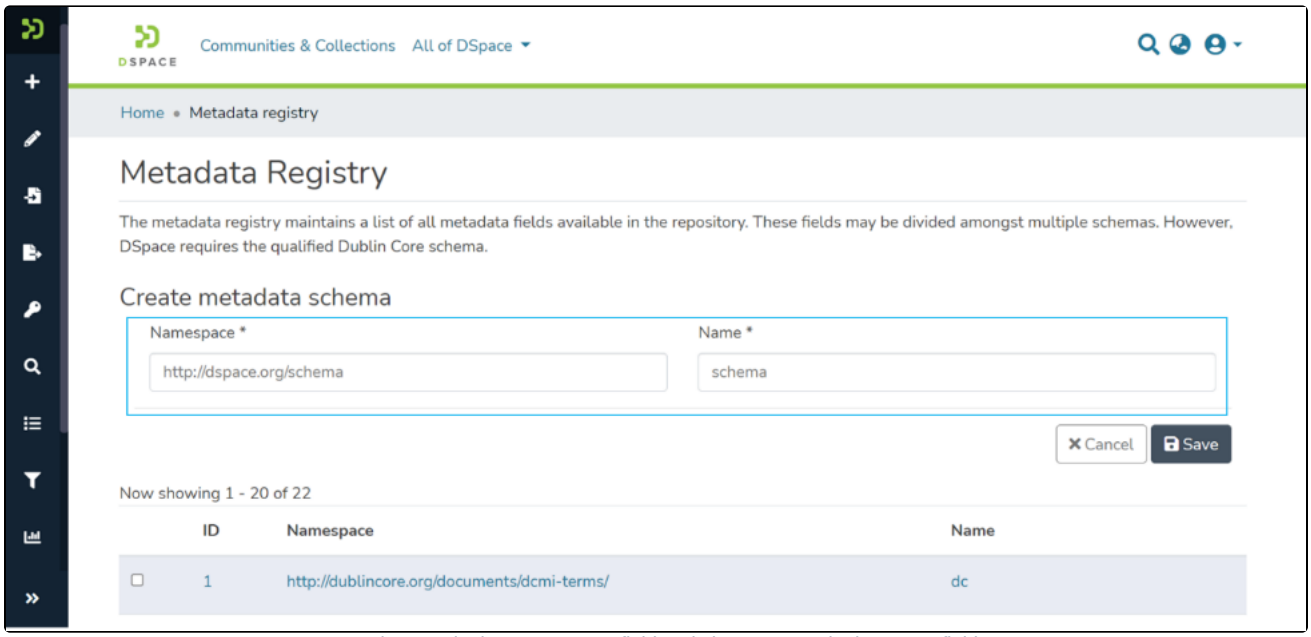
Admin menu displays vertically on the right side of the screen

Step 4: Roll over your cursor over the administration menu and click on Registries. Click on Metadata to go to the Metadata Registries.



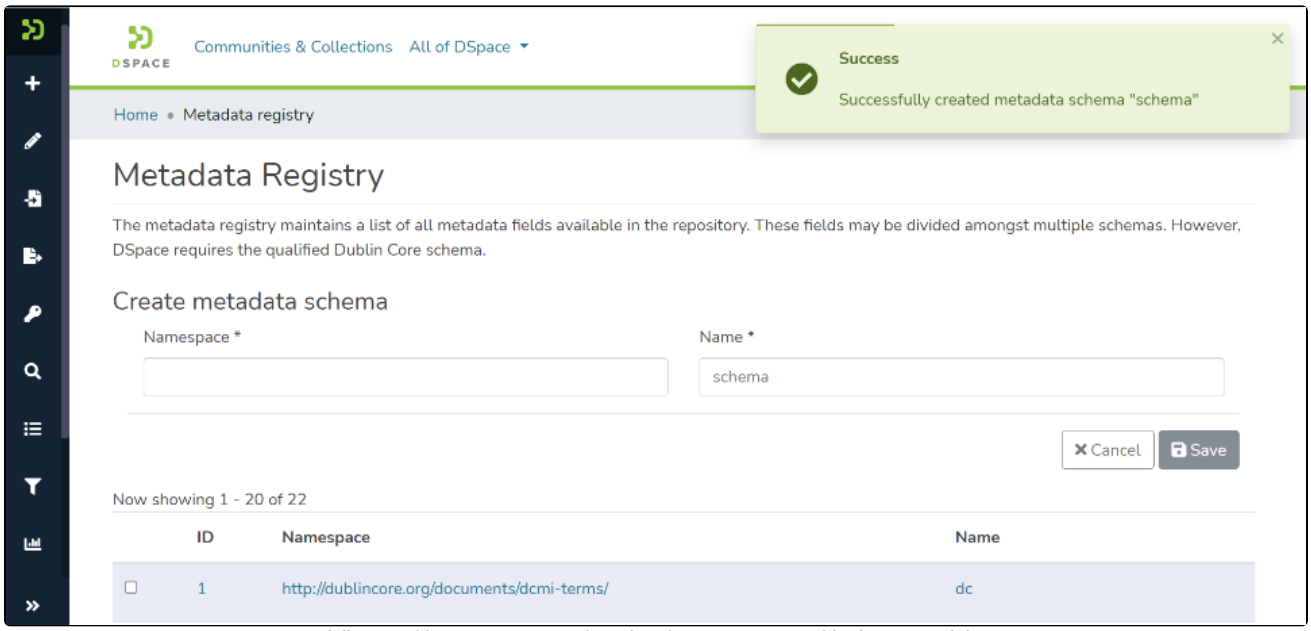
Under Registries click on Metadata

Step 5: Enter the value of your choice in the ‘Namespace’ field and the short value in the ‘Name’ field.



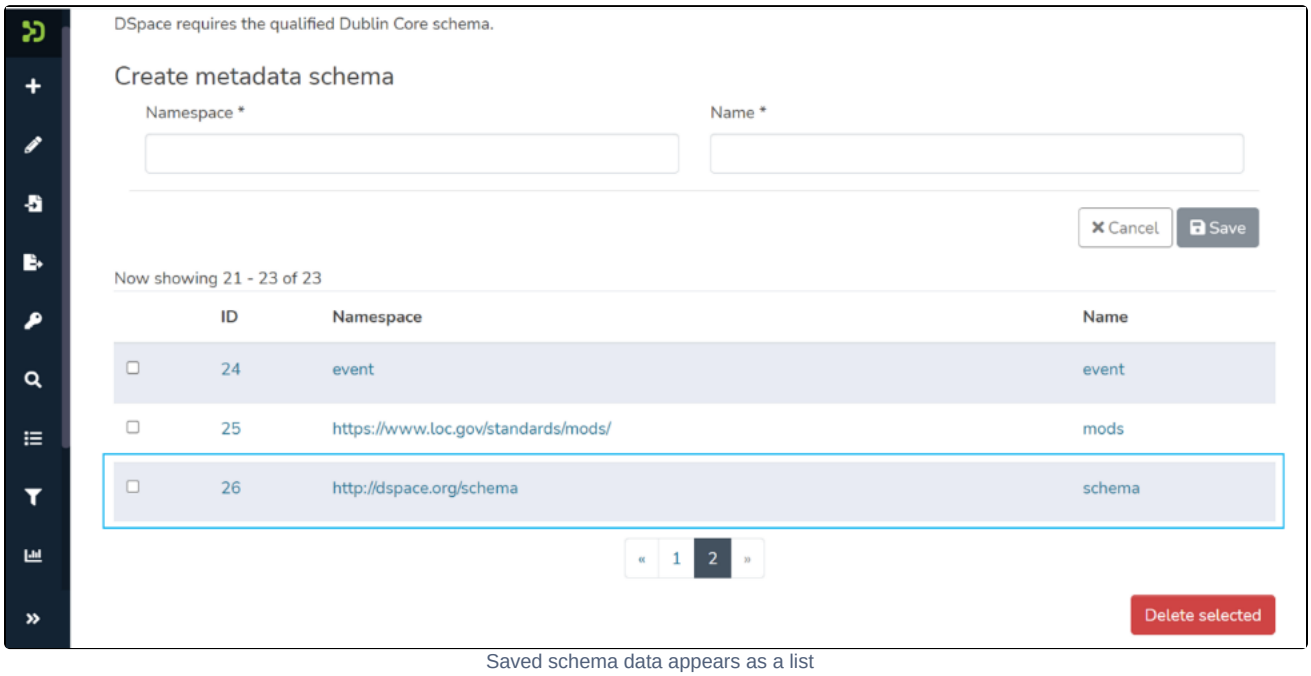
Enter the URL in the Namespace field and shorter Name in the Name field

Step 6: Click on the Save button to create a Metadata schema. A success prompt will appear upon metadata schema creation in the DSpace.



Successfully saved banner appears when the changes are saved in the upper right corner.

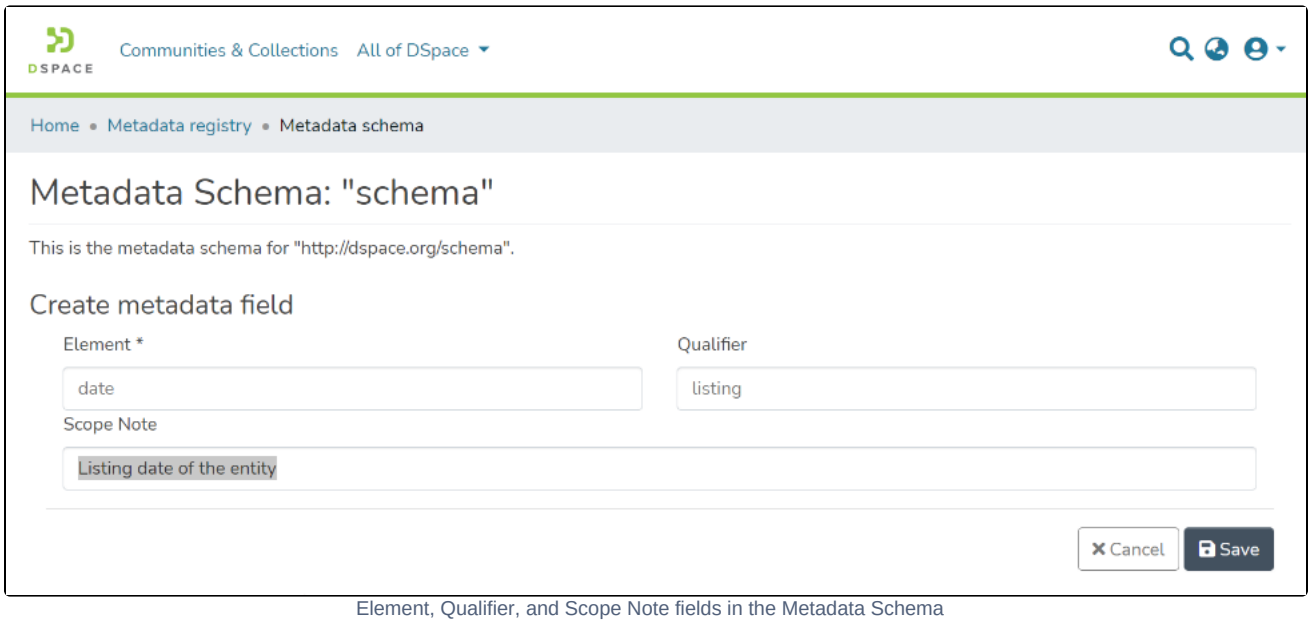
Step 7: Scroll down the list to see the schema added. Click on the namespace or name value to access the metadata schema.



Saved schema data appears as a list

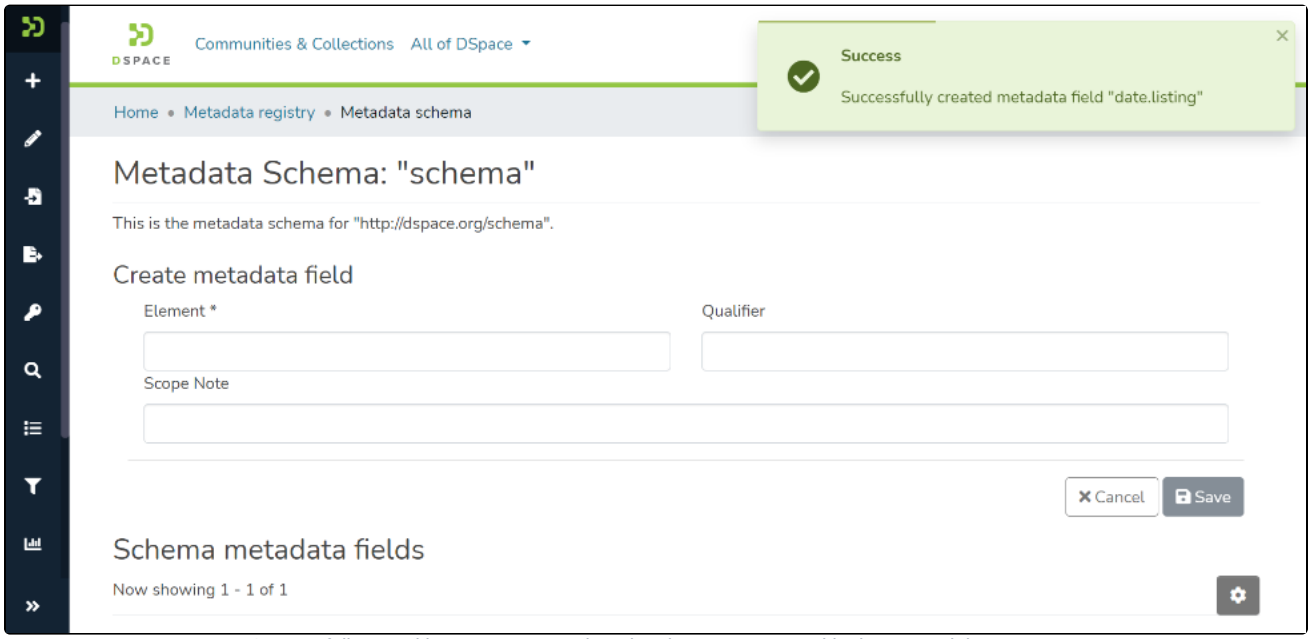
Step 8: Enter desired values in ‘Element,’ ‘Qualifier,’ and ‘scope note.’

1. Element is the primary value in the metadata schema. For example, suppose the date is required to be added as a metadata element. In that case, the element’s value can be the date.
2. Qualifier – Use the Qualifier to segregate the metadata element further. For example, suppose you want to add multiple dates under an element. In that case, users can add various qualifiers for each date type. For example, there can be a content listing date, content de-listing date, and other types of dates.
3. Scope Note – Users can enter the definition of the metadata element created to benefit other users.



Element, Qualifier, and Scope Note fields in the Metadata Schema

Step 9: Click on the Save button to add an element to the selected metadata schema. You will see a confirmation prompt upon the successful addition of the element.

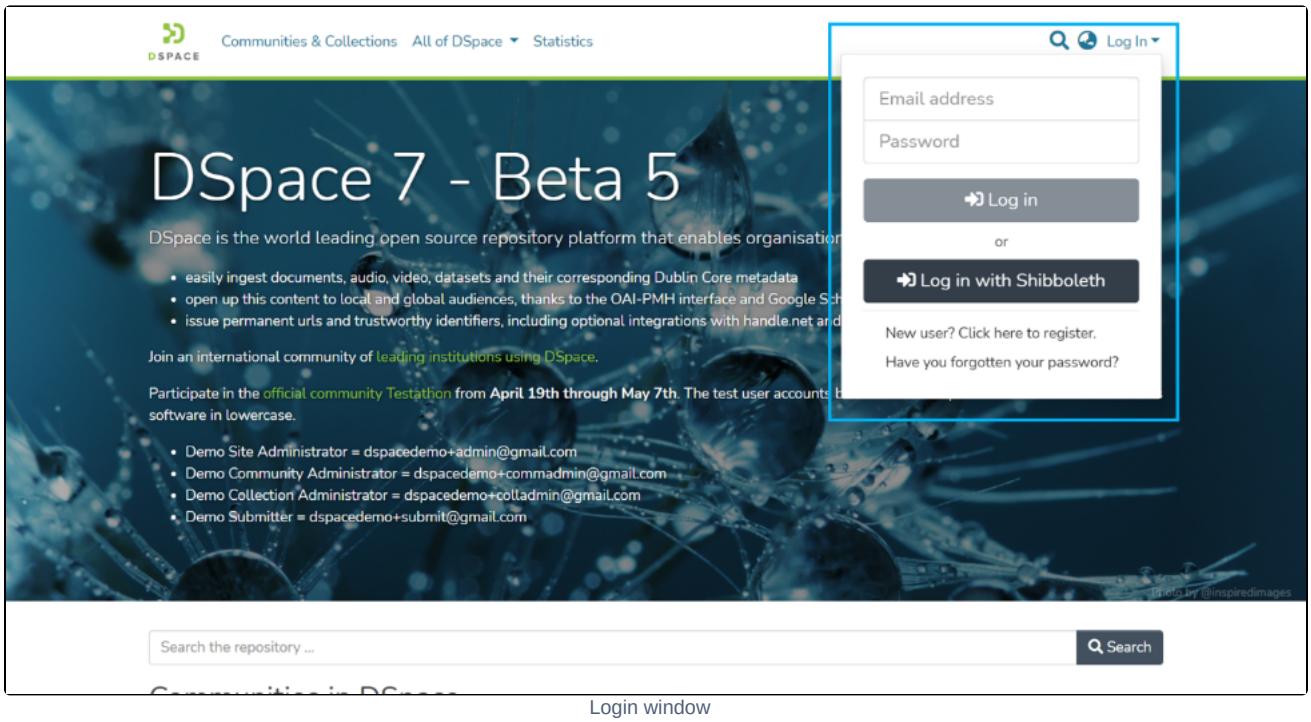


Successfully saved banner appears when the changes are saved in the upper right corner.

Metadata Registry Management

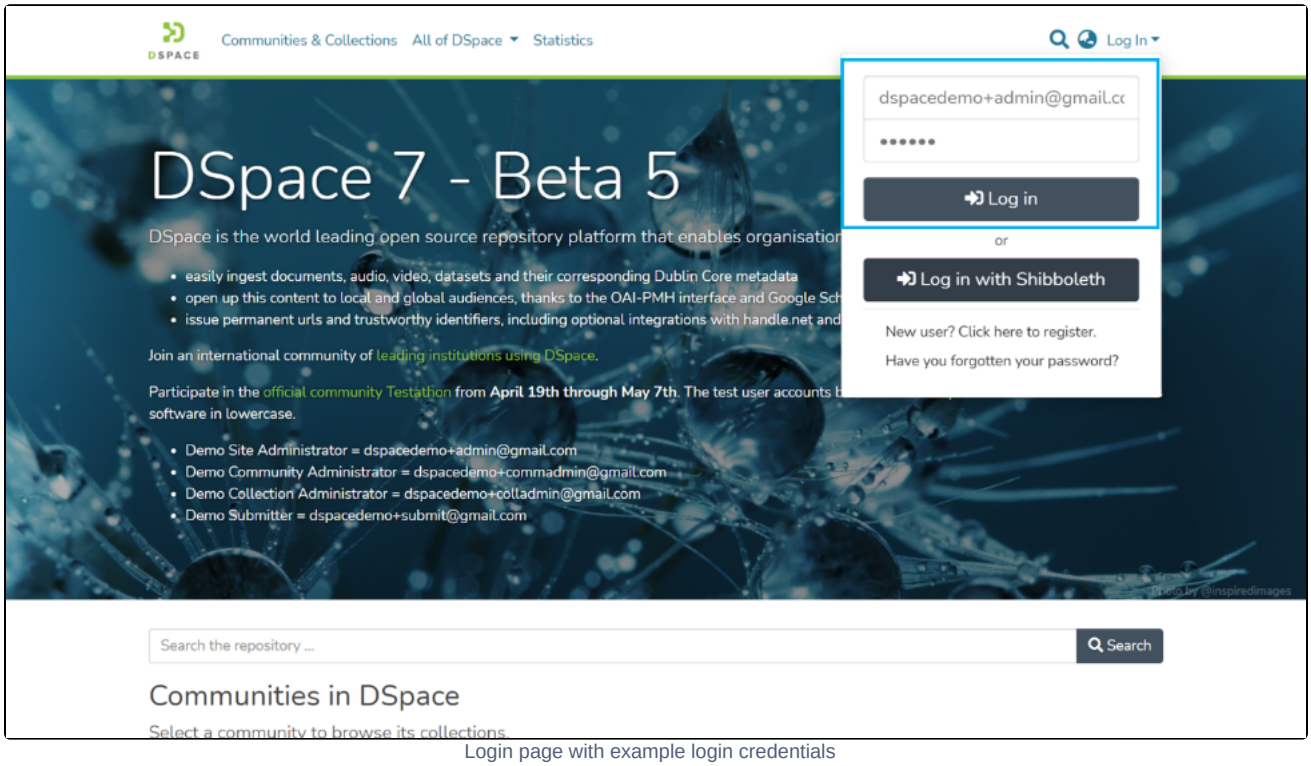
This process is for System and Community Administrator user profiles. Under metadata registry management, users can update or delete metadata elements or metadata registries.

Step 1: Go to the home page of DSpace and click on the “Log In” link appearing at the top right corner of the screen, and the pop-up will open, as illustrated in the below screen.



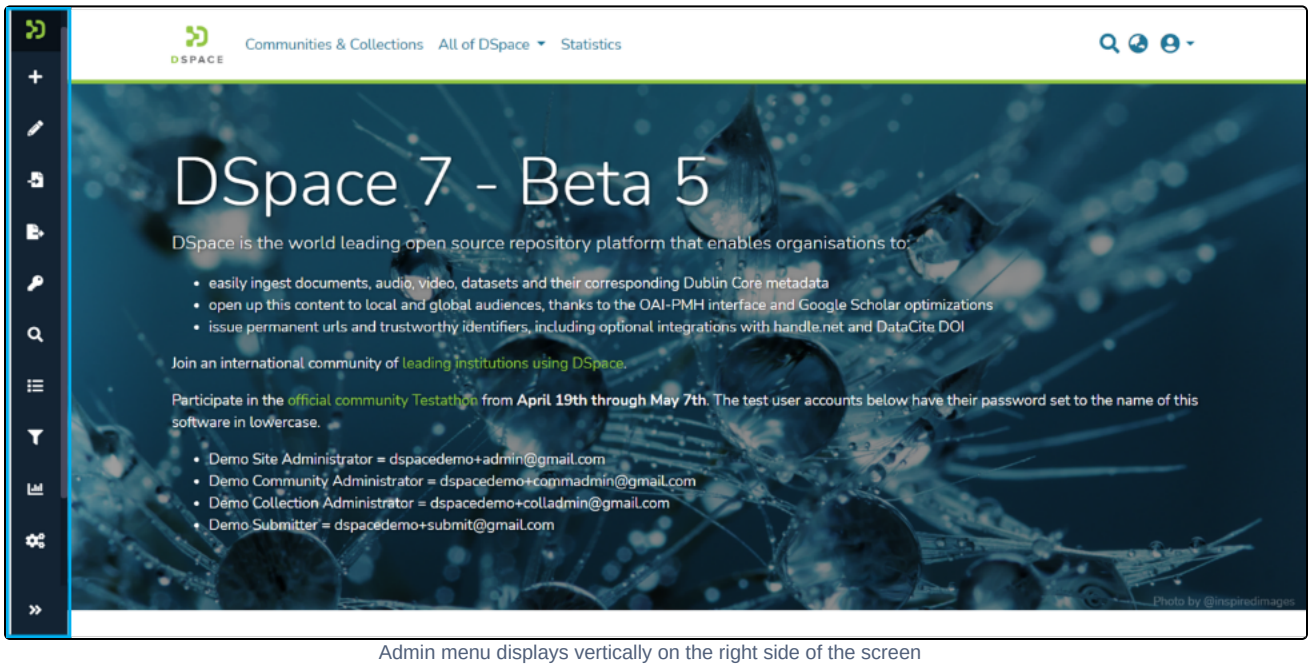
Login window

Step 2: Enter your user id and password and click on the login button for logging in to DSpace.



Login page with example login credentials

Step 3: Users with administrative rights will see the admin menu on the left-hand side of the screen, as shown in the below illustration.



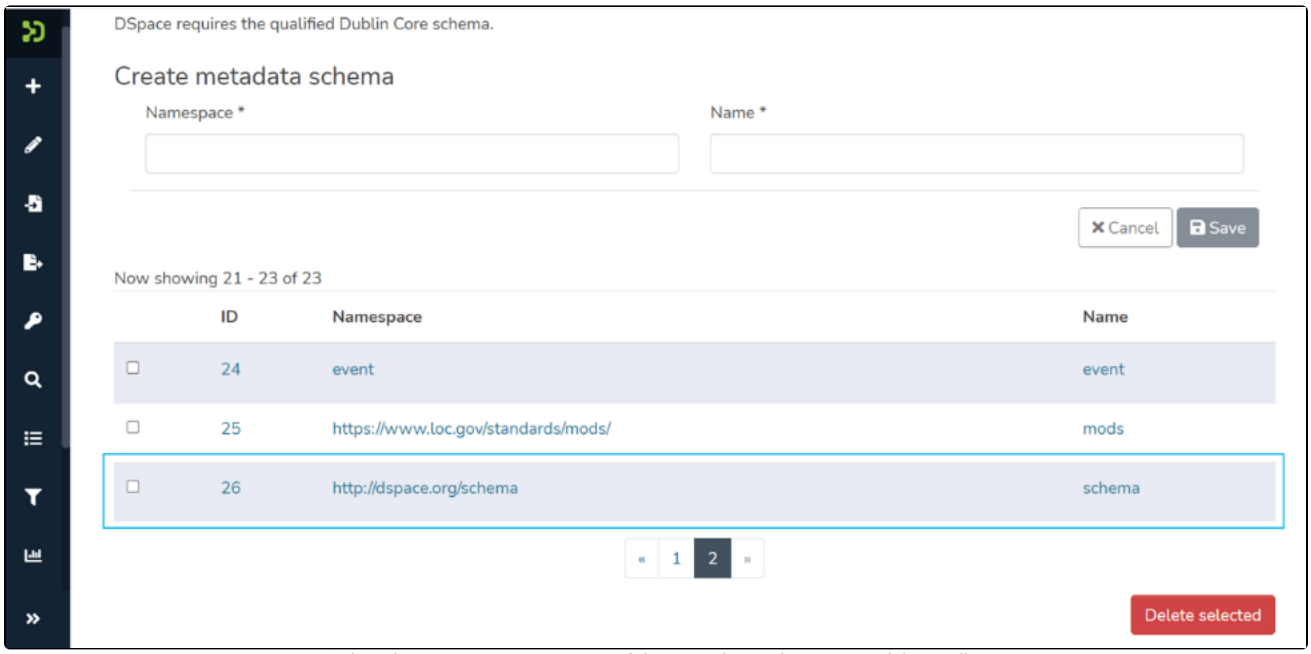
Admin menu displays vertically on the right side of the screen

Step 4: Roll over your cursor over the administration menu and click on Registries. Click on Metadata to go to the Metadata Registries.



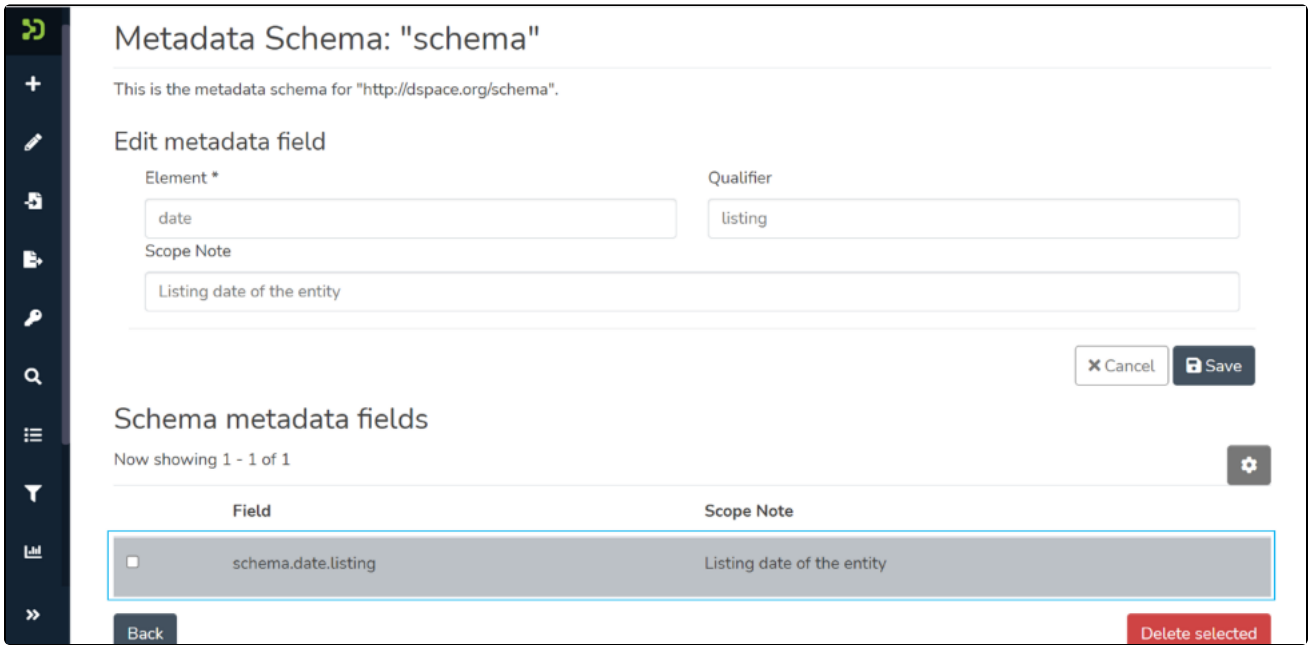
Click on Registries and Metadata

Step 5: Click on the namespace or name of the Metadata schema you want to edit.



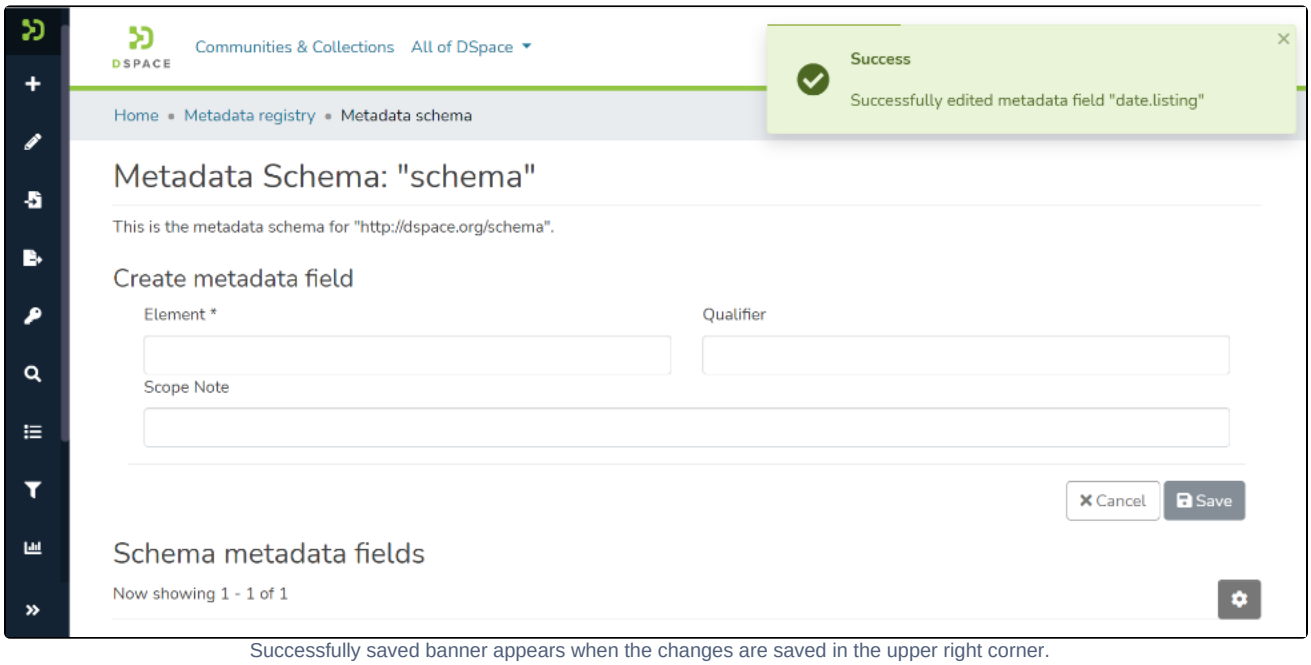
Select the Namespace or Name of the Metadata schema you wish to edit

Step 6: Click on the metadata element you want to edit. Upon clicking the target element, its values will be populated in the corresponding fields under the Edit Metadata fields section, as shown below.



Edit metadata schema

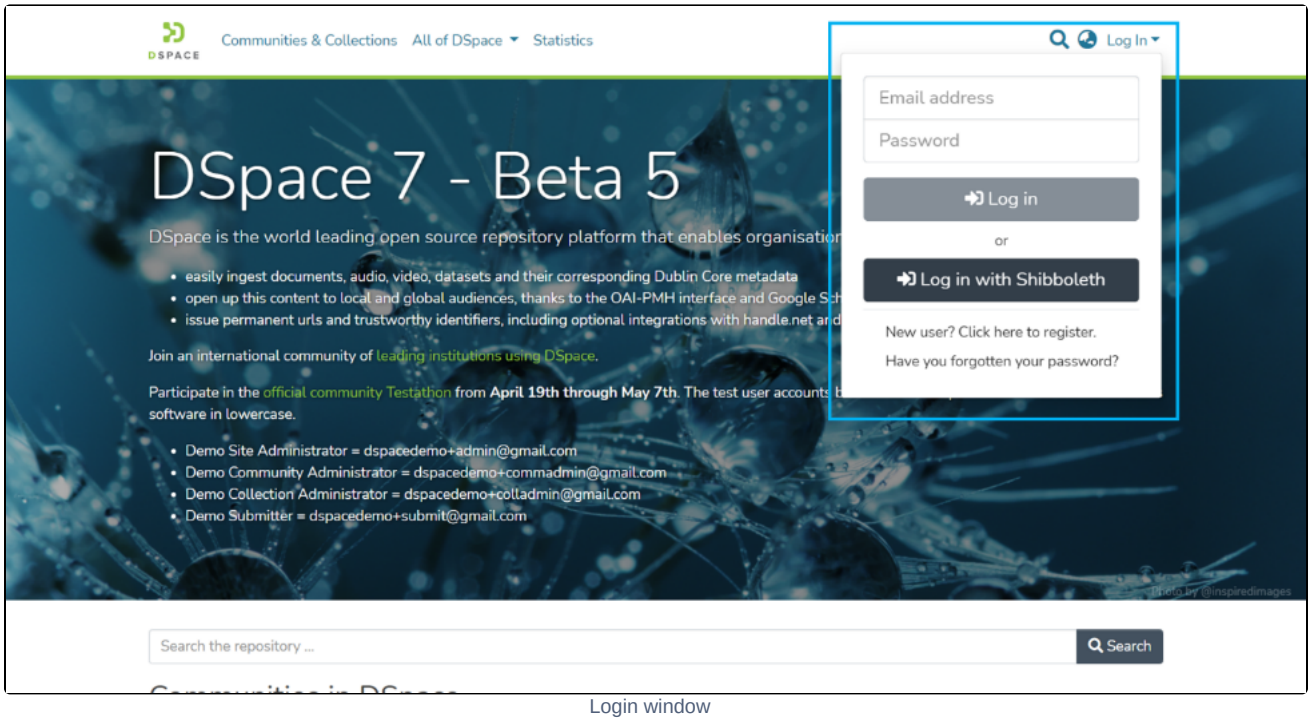
Step 7: Update the value under the target field(s) and click on the ‘Save’ button to save them. A success prompt will appear upon successful update. In addition, an updated metadata element will appear in the metadata schema.



Successfully saved banner appears when the changes are saved in the upper right corner.

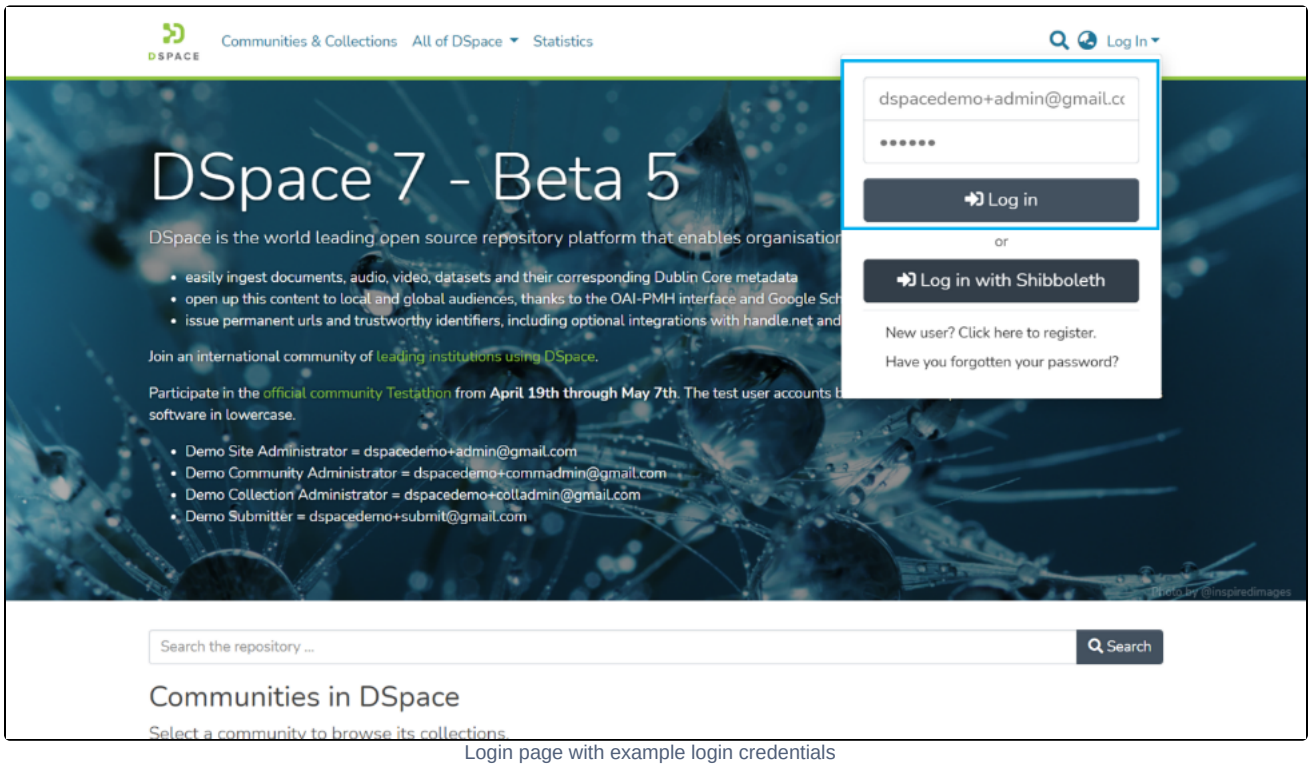
Delete Metadata Schema

Step 1: Go to the home page of DSpace and click on the “Log In” link appearing at the top right corner of the screen, and the pop-up will open, as illustrated in the below screen.



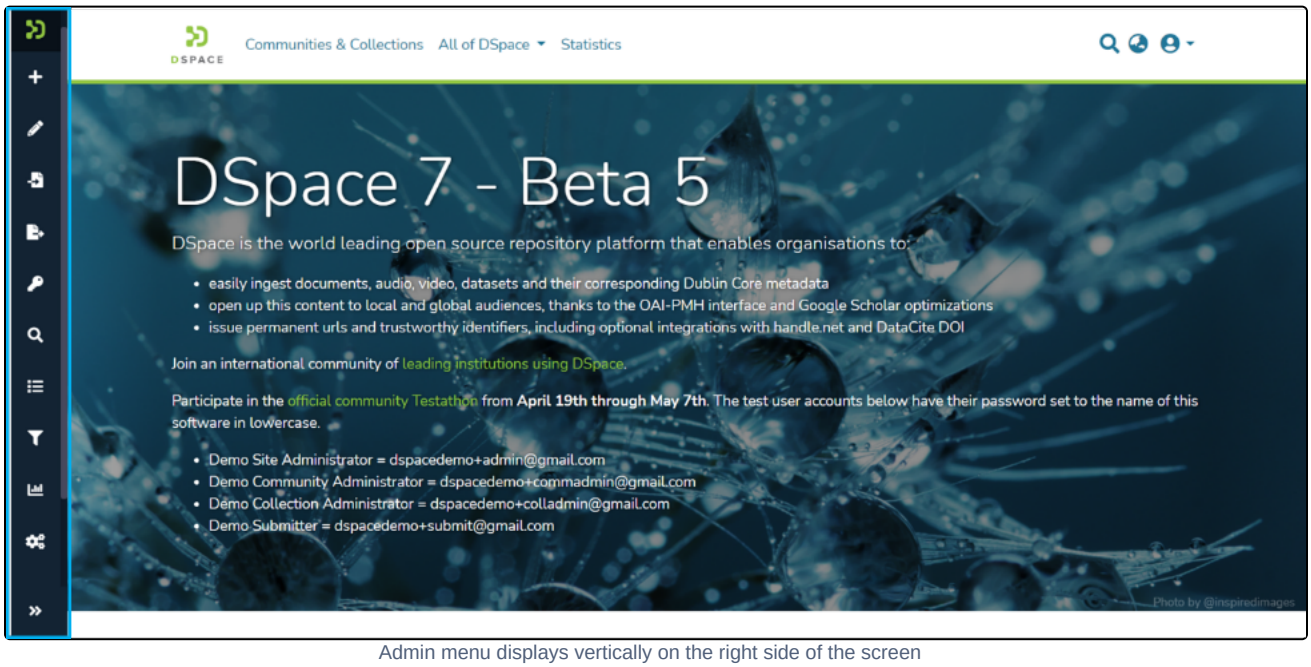
Login window

Step 2: Enter your user id and password and click on the login button for logging in to DSpace.



Login page with example login credentials

Step 3: Users with administrative rights will see the admin menu on the left-hand side of the screen, as shown in the below illustration.



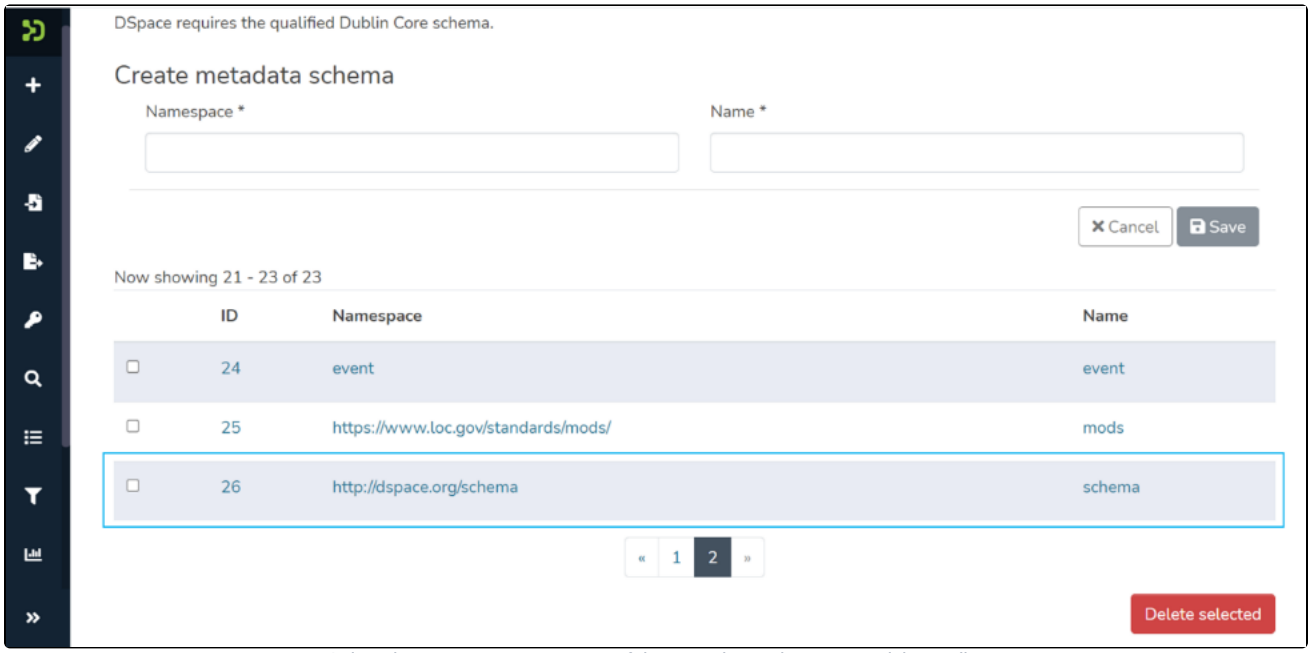
Admin menu displays vertically on the right side of the screen

Step 4: Roll over your cursor over the administration menu and click on Registries. Click on Metadata to go to the Metadata Registries.



Click on Registries and Metadata

Step 5: Click on the namespace or name of the Metadata schema under which you want to delete an element.



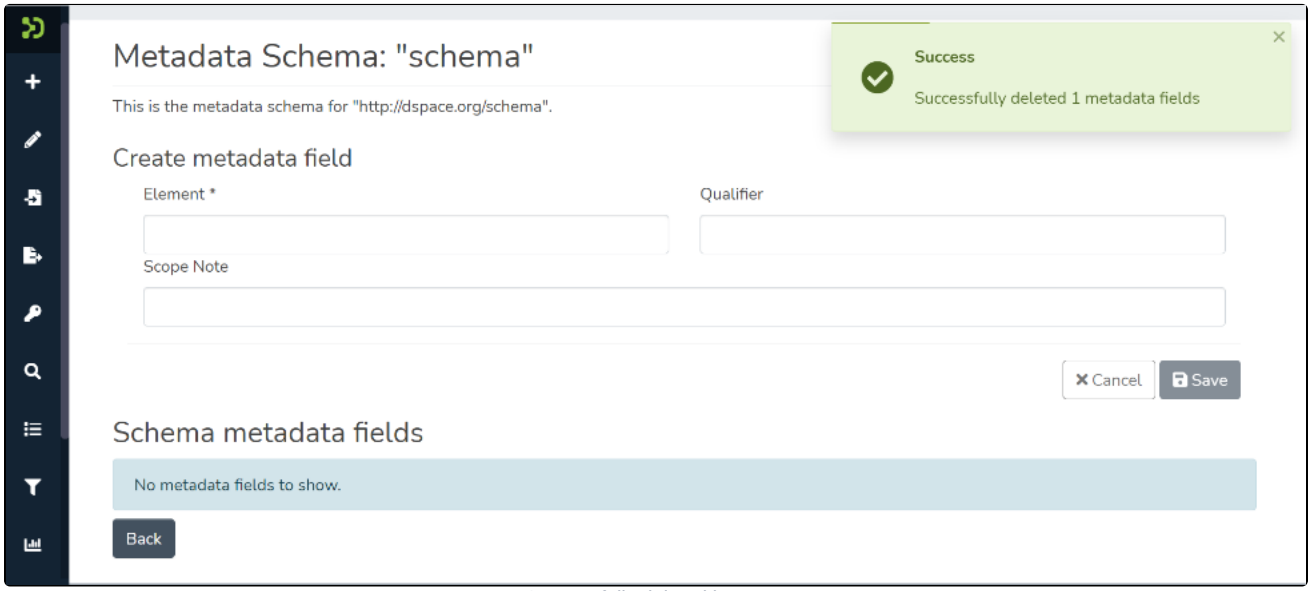
Select the Namespace or Name of the Metadata schema you wish to edit

Step 6: Click on the checkbox appearing on the left of the metadata element you want to delete.



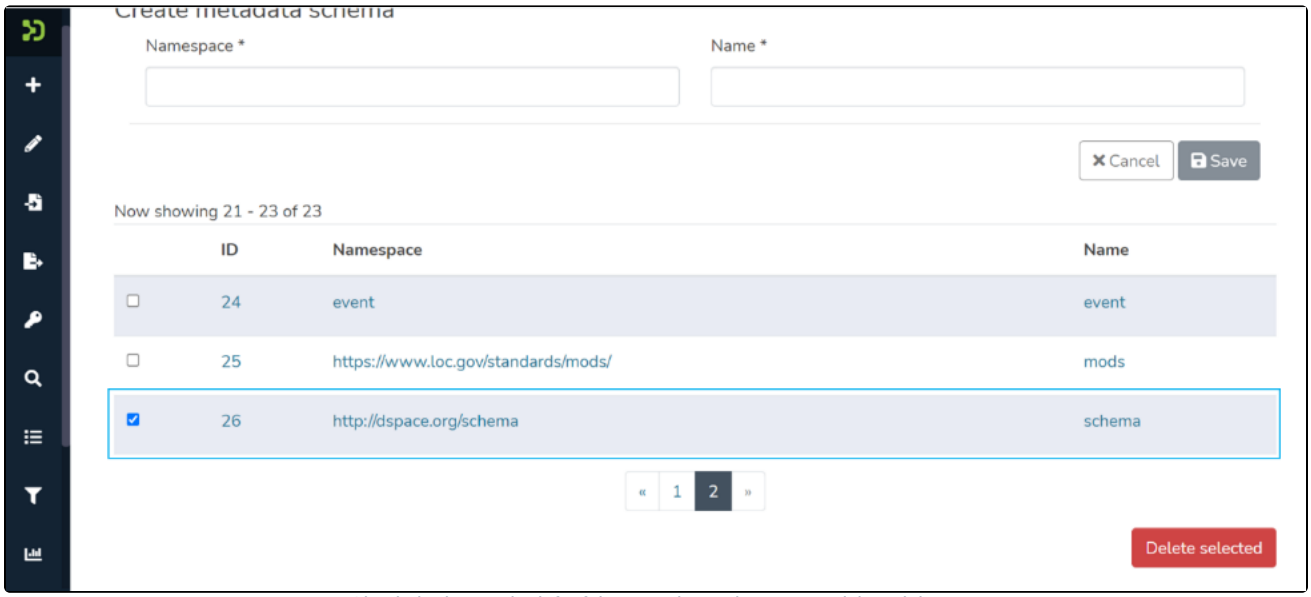
Check the box to the left of the metadata schema you wish to delete

Step 7: Click on the ‘Delete selected’ button appearing in red at the bottom of the page. Upon successful deletion of the element, you will see a prompt confirming deletion of the field.



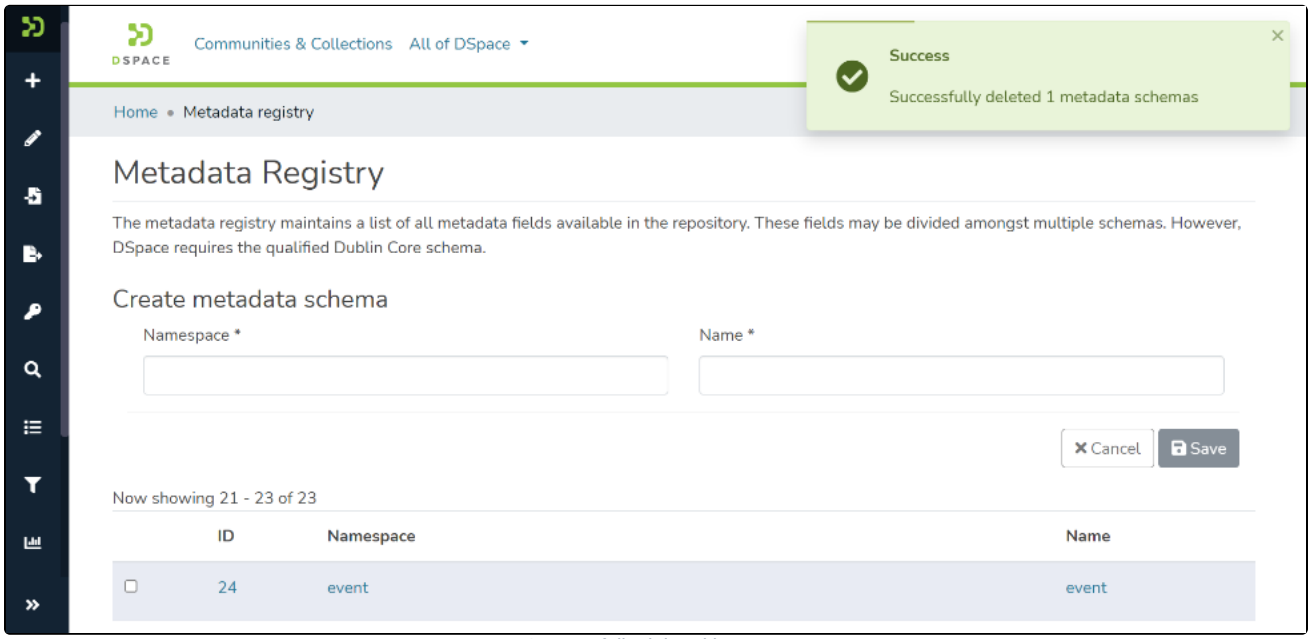
Successfully deleted banner

Step 8: To delete the entire metadata schema, click on the checkbox appearing on the left of the target metadata schema(s).



Check the box to the left of the metadata schema you wish to delete

Step 9: Click on the ‘Delete selected’ button appearing in red at the bottom of the page. Upon successful deletion of the element, you will see a prompt confirming deletion of the field.



Successfully deleted banner

Request-a-copy

When an attachment/bitstream(s) are restricted from open access, users can use the Request-a-Copy option to request a content submitter grant access to the restricted content. The process has been divided into two parts i.e.

- Approve an access request
- Create a request

Approve an access request

- Target Audience
- Overview
- Request creation process

Target Audience

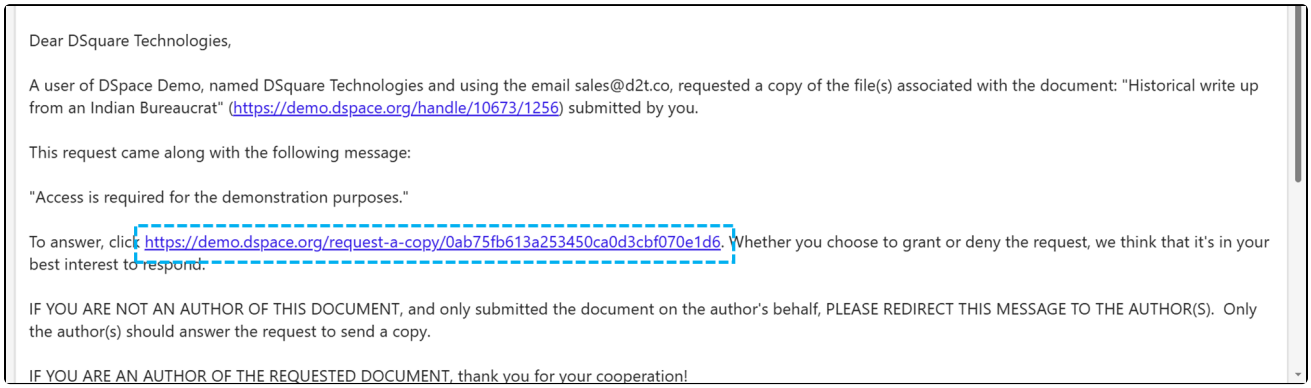
- Content submitters

Overview

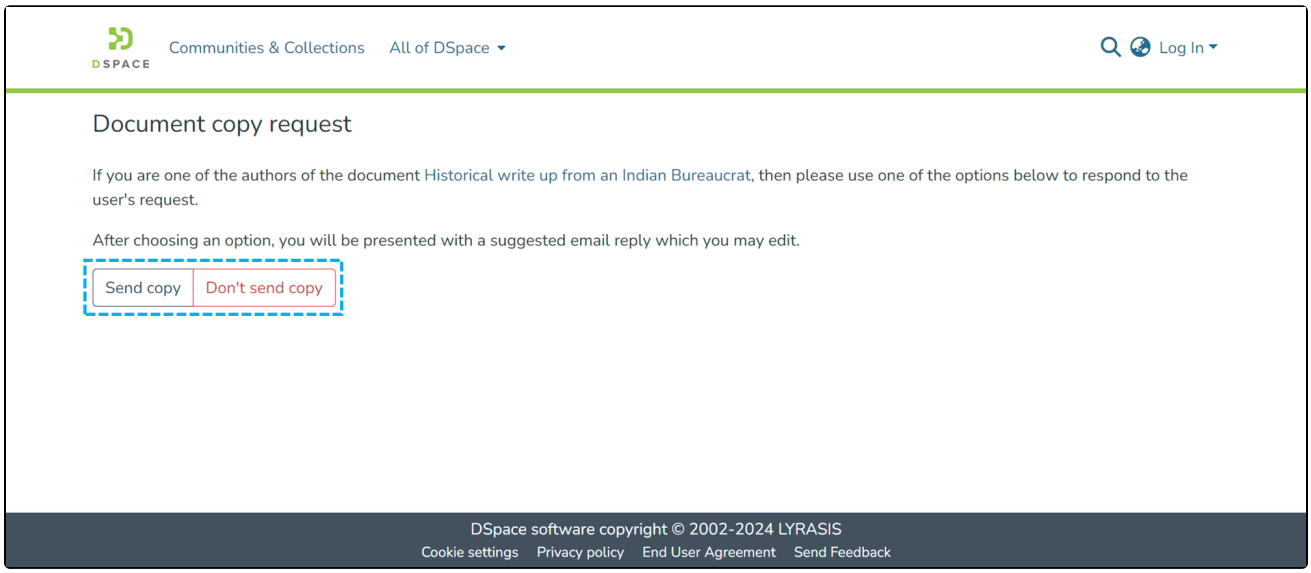
Users marked as content submitters receive access approval requests through e-mails and they can determine, whether to grant or deny the access request. The steps to do so are outlined below.

Request creation process

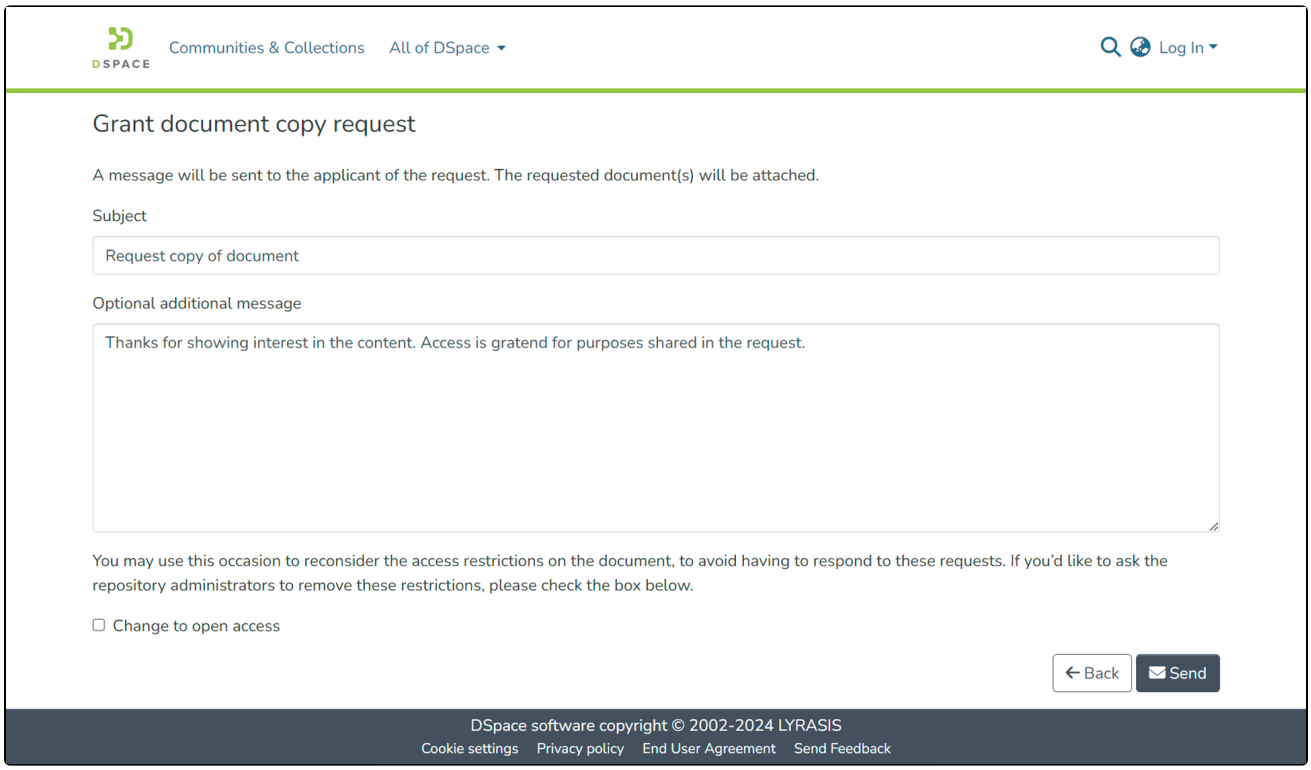
Step 1: Content submitters receive the access request on registered e-mail, as demonstrated in the below screenshot. Users need to click the highlighted link to proceed with the next steps.



Step 2: User can click one of the two options i.e. "Send Copy" to approve the request, or click "Don't send copy" to reject.



Step 3: On the following screen, users can add a subject for the response along with the message describing acceptance or rejection of the request, and complete the process by clicking the "Send" button.


DSPACE
Communities & Collections
All of DSpace
Log In
Grant document copy request
A message will be sent to the applicant of the request. The requested document(s) will be attached.
Subject
Request copy of document
Optional additional message
Thanks for showing interest in the content. Access is gratend for purposes shared in the request.
You may use this occasion to reconsider the access restrictions on the document, to avoid having to respond to these requests. If you'd like to ask the repository administrators to remove these restrictions, please check the box below.
Change to open access
Back
Send
DSpace software copyright © 2002-2024 LYRASIS
Cookie settings
Privacy policy
End User Agreement
Send Feedback

Create a request

- Target Audience
- Overview
- Request creation process

Target Audience

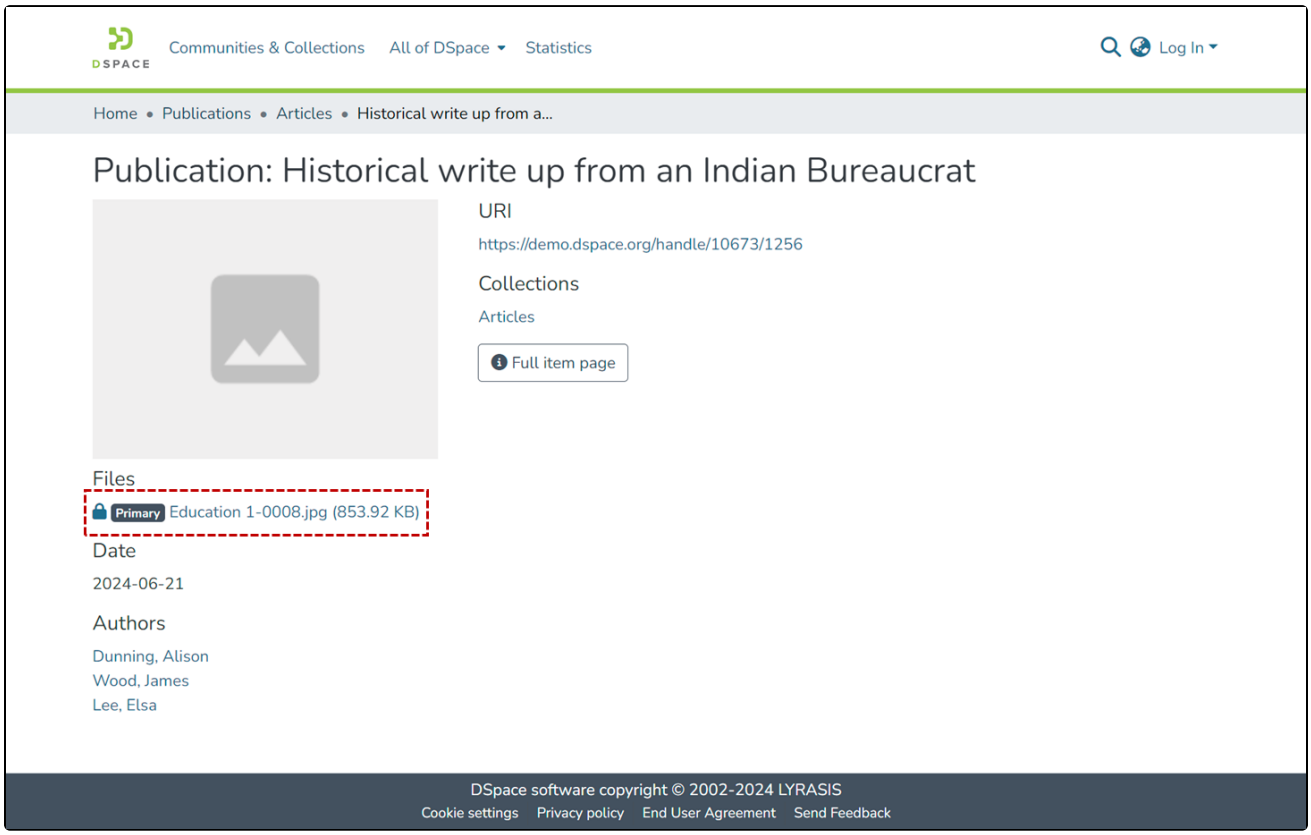
- Content consumers

Overview

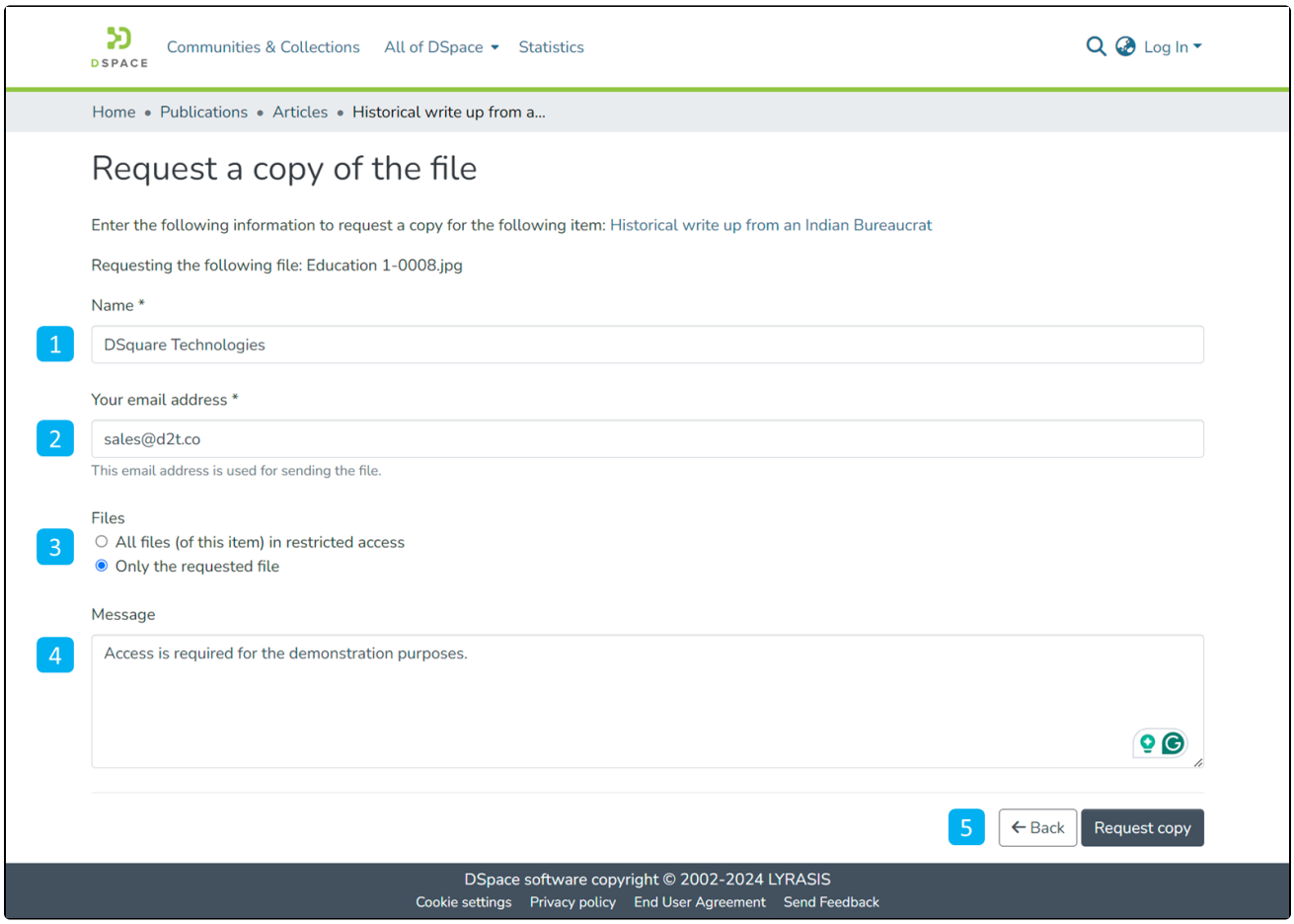
Content contributors can restrict access to attachments/bitstreams uploaded with an item. When this occurs, users can still access the attachments by utilizing the request-a-copy option as outlined below.

Request creation process

Step 1: Users should locate the desired item and navigate to its detail page. This can be done through various methods such as browsing, searching, or using taxonomy navigation. If the bitstream is restricted, a lock icon will appear next to the bitstream link, as highlighted in the screen below.

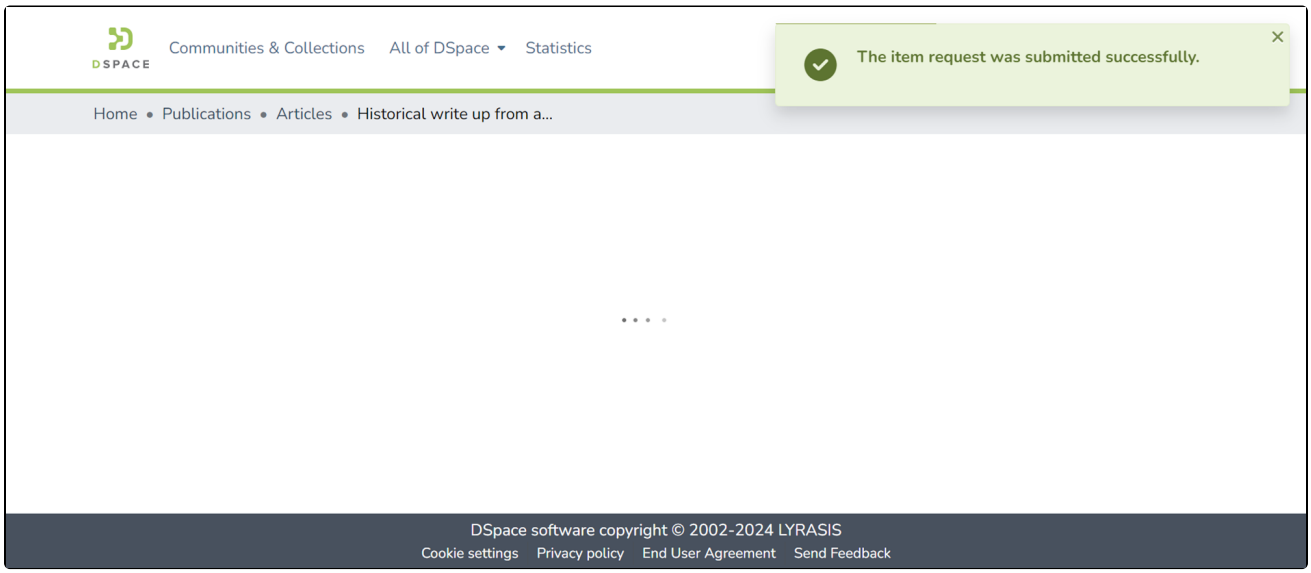


Step 2: Click the link to go to the Request-a-copy page and enter the required details. Inserting adequate details will help content contributors to understand the need and make an informed decision. Each element on the page is explained below

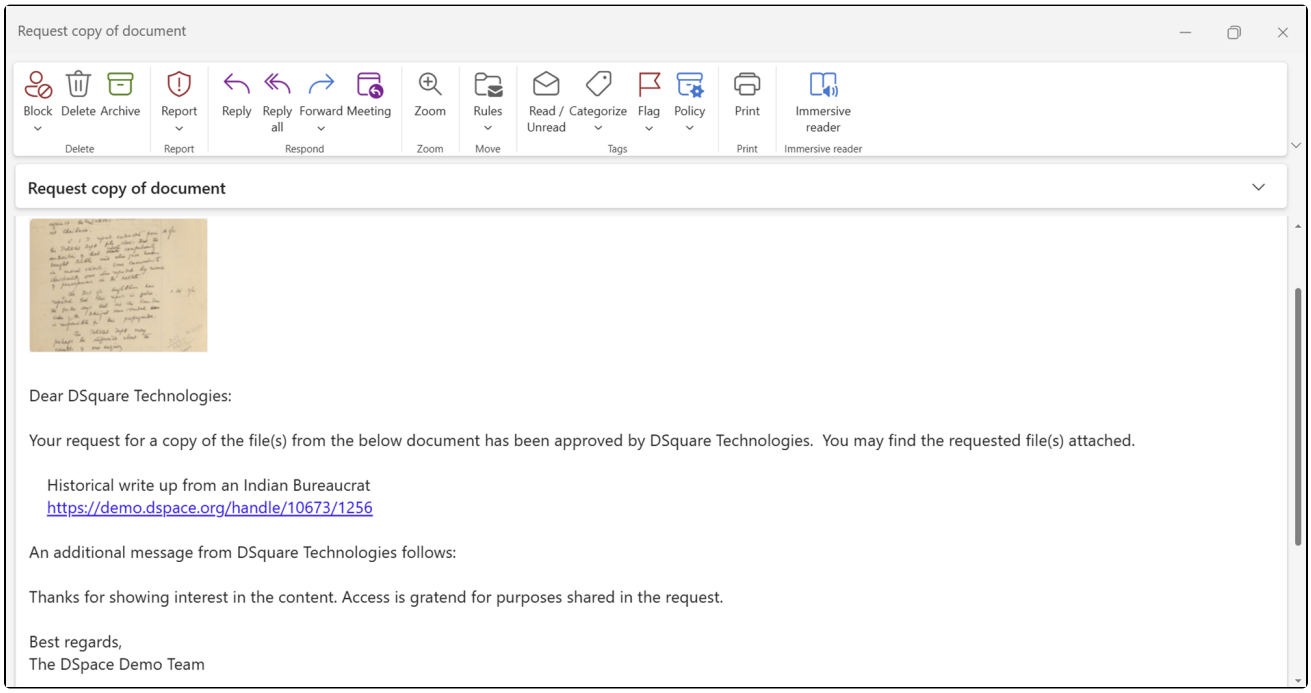


1. Name: Enter file the requester's name
2. E-Mail Address: Enter the correct e-mail ID here. A response to the request will be sent on the e-mail ID entered.
3. Files needing access: The requester can select "Only the requested file" for accessing a target bitstream, or select, "All files (of this item) in restricted access" to access all attachments.
4. Message: Enter the reason for accessing the file(s). A well-explained reason is recommended.
5. Action buttons: Action buttons help users to complete the action.

Step 3: After completing the request-a-copy form, click the "Request copy" button to complete the action. A success prompt should appear upon successful completion of the request.



Step 4: Users will receive an e-mail confirming a response to the access request made. Users can click the link to access the attachment from the mail.



Search - Advanced

Advanced Search (including boolean options) is already supported in the DSpace 7 search page. Boolean keywords can be used, and you can also specify to search within specific fields by name. Some examples:

- *Basic searching:* Searching `test power` will return results with both these words in them (this is equivalent to an "AND" boolean search). E.g. https://demo.dspace.org/search?query=test%20power
- *Boolean searching options*
  - Searching `test AND power` will return results with both these words in them. E.g. https://demo.dspace.org/search?query=test%20AND%20power
  - Searching `test OR power` will return results with either of these words in them. E.g. https://demo.dspace.org/search?query=test%20OR%20power
  - Searching `test NOT power` will return results with "test" but not including "power". E.g. https://demo.dspace.org/search?query=test%20NOT%20power
- *Phrase searching:* Searching `"test power"` (in quotes) will return results with the exact phrase "test power" in them. E.g. https://demo.dspace.org/search?query=%22test%20power%22
- *Searching within specific fields*
  - Searching `dc.title:test` will only return results where the `dc.title` includes "test". E.g. https://demo.dspace.org/search?query=dc.title:test
  - Searching `dc.title:test AND dc.description.abstract:green` will only return results where the `dc.title` field includes "test" and the `dc.description.abstract` field returns "green". E.g. https://demo.dspace.org/search?query=dc.title:test%20and%20dc.description.abstract:green
  - Searching `dc.subject:fin*` will only return results where one (or more) dc.subject fields start with "fin" (e.g. finance, financial, finish, etc), e.g. https://demo.dspace.org/search?query=dc.subject:fin*
- *Wildcard searching:*
  - Searching `test pow*` will return results including "test" and any word starting with "pow". E.g. https://demo.dspace.org/search?query=test%20pow*
  - Searching `dc.description.abstract:*` will return results that include the "dc.description.abstract" metadata field (with any value in it). E.g. https://demo.dspace.org/search?query=dc.description:*
- *Range searching:*
  - Searching `dc.date.issued:[1999 TO 2003]` will return results that have a "dc.date.issued" metadata field that has a date between 1999 and 2003 (inclusive). E.g. https://demo.dspace.org/search?query=dc.date.issued:%5B1999%20TO%202003%5D
  - Searching `dc.date.issued:[2010 TO *]` will return results that have a "dc.date.issued" metadata field that has a date after (or including) 2010 . E.g. https://demo.dspace.org/search?query=dc.date.issued:%5B2010%20TO%20*%5D
- *Special characters:* Some characters have special meaning in searches, e.g. colon (:), asterisk (*), boolean operators, etc. If you need to search for these characters exactly, surround them with double quotes.
  - Searching `"test:power"` will search for that string *exactly* (including the colon character). (NOTE: Without the quotes, DSpace would attempt to perform "Searching within specific fields" (see above) as the colon is a special character.)

DSpace supports all Solr search syntax options, as all searches in DSpace are sent directly to Solr. For more examples, see the Solr documentation for the "Specifying Terms for the Standard Query Parser".

User management

Documentation for repository managers.

- Add or Manage an E-Person
- Create or manage a user group

Add or Manage an E-Person

An E Person is a user in DSpace who can be assigned various rights to perform activities or manage content access in the repository.

This section provides details about various methods to create or update an E Person in DSpace. For example, DSpace allows users to self-register. In addition, users with administrative rights can create and update E Person in the system. Both methods are explained in the latter part of this document.

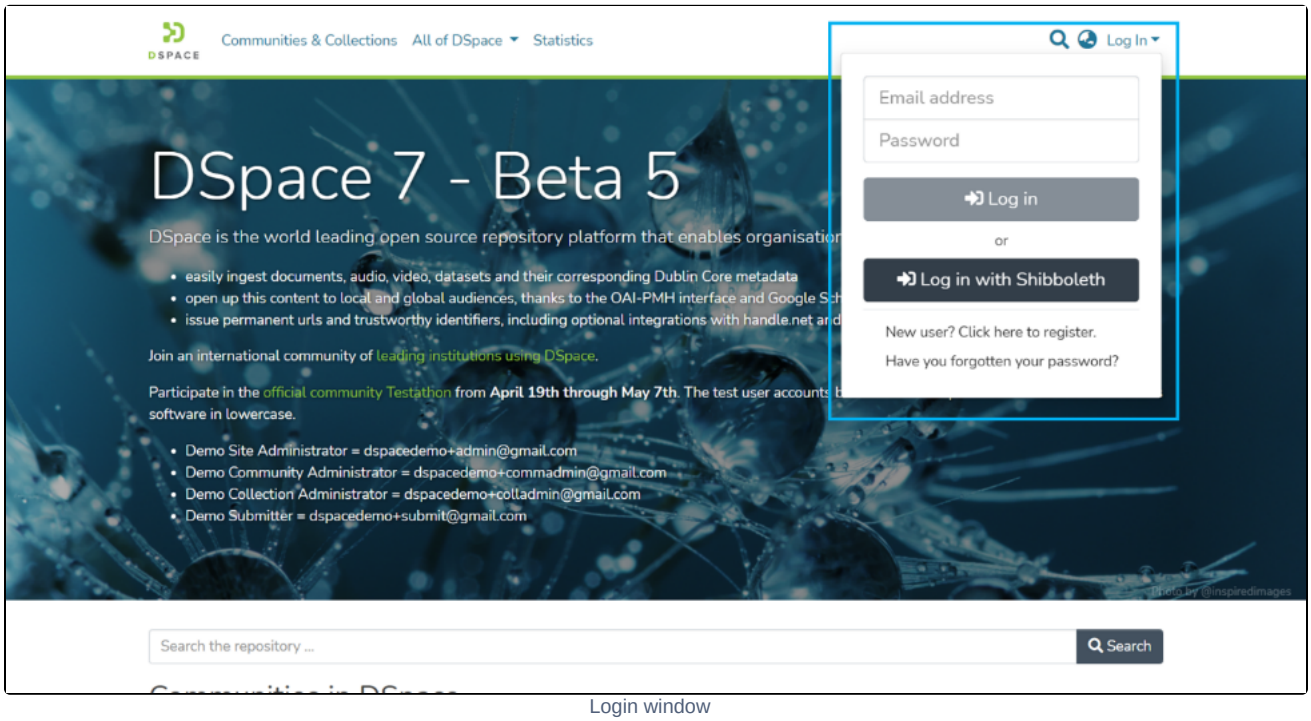
- Audience
- Add E Person – Self Registration
- Add E Person – Registration by Administrator
- Update an Eperson
    - Update an Eperson – Update details & Manage Log in
    - Update an Eperson – Manage user groups membership
    - Update an Eperson – Delete Eperson

Audience

1. Repository Administrator
2. Community Administrator
3. Collection Administrator
4. Base User

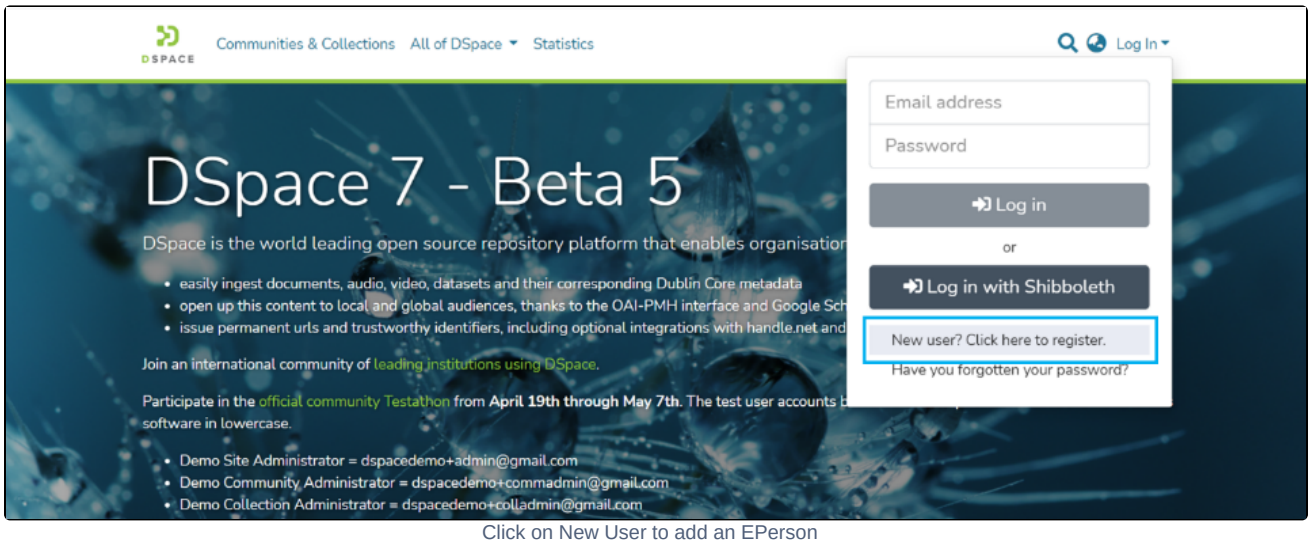
Add E Person – Self Registration

Step 1: Go to DSpace's home page and click on the "Log In" link appearing at the top right corner of the screen, and the pop-up will open, as illustrated below screen.



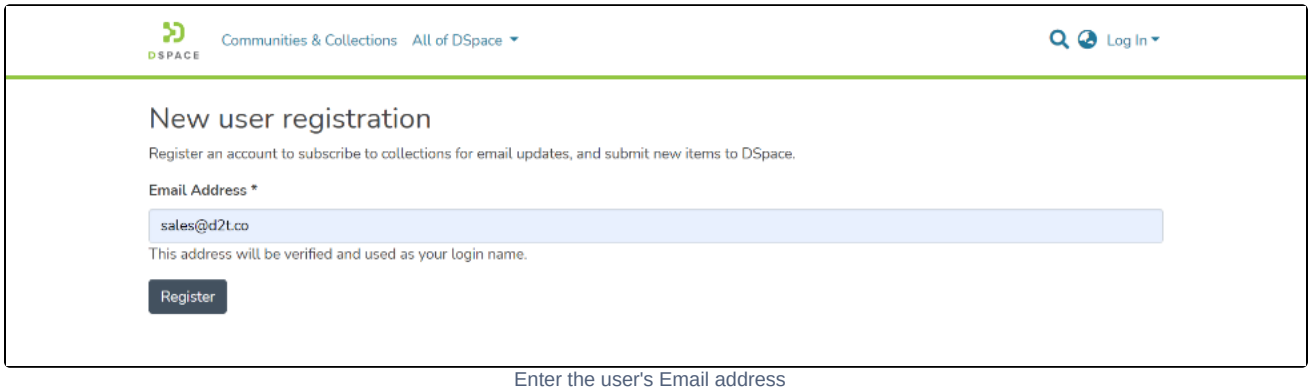
Login window

Step 2: Click on the New User to add an E Person in DSpace.



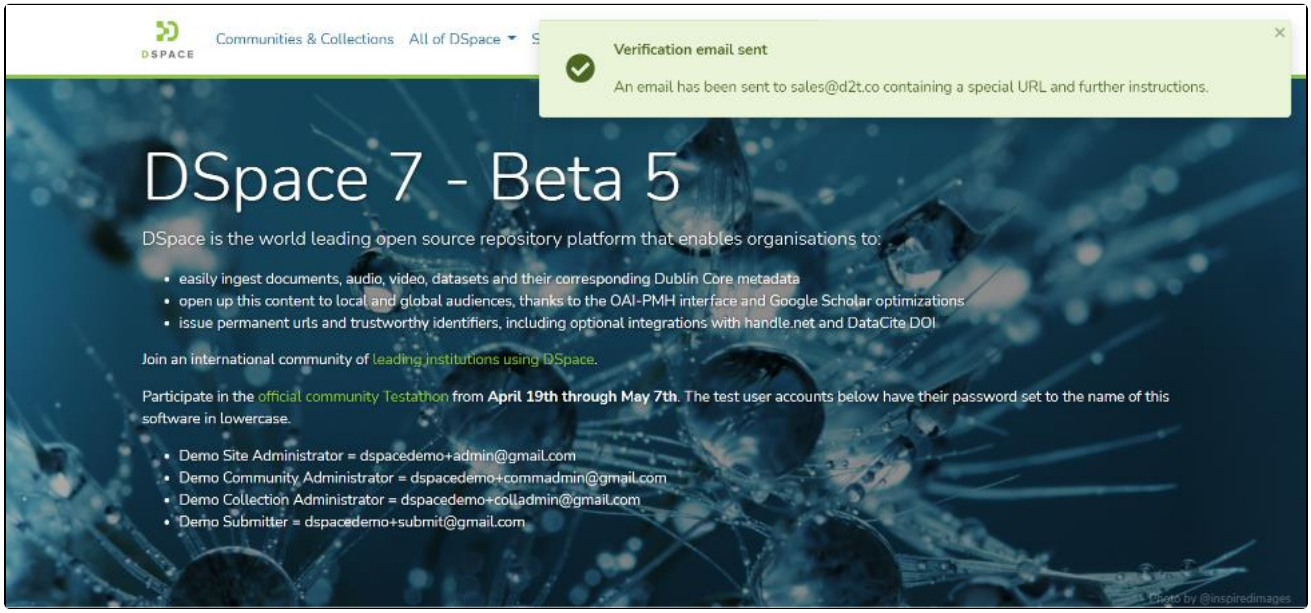
Click on New User to add an EPerson

Step 3: Enter the user's Email ID who needs to be registered as E Person in DSpace and click on the Register button.



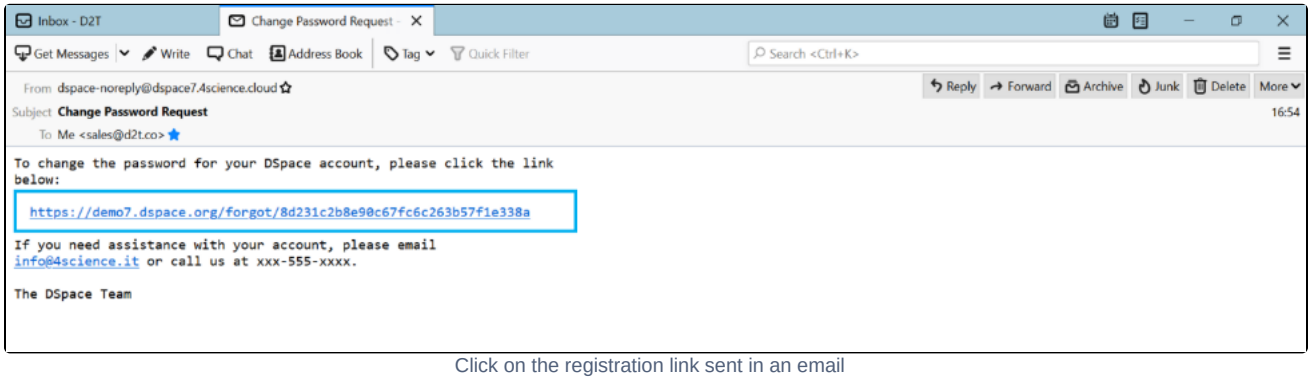
Enter the user's Email address

Step 4: After clicking the register button, an email will be sent to the user's email id, and the user will be redirected to the home page. This E Mail will go from the communication mail ID registered in the DSpace instance.



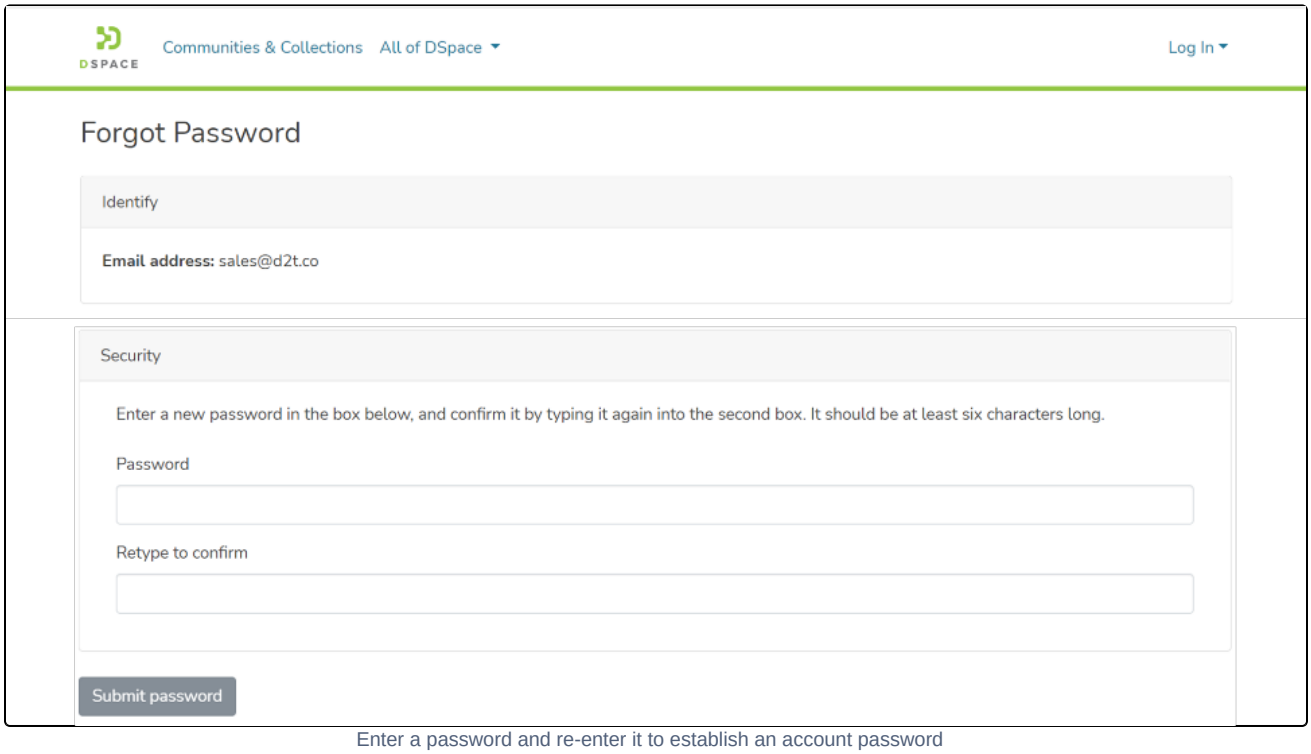
Verification email sent success banner

Step 5: Click on the unique registration link received in the email to continue with the registration process. Suppose your email client or server security settings do not show hyperlinks. In that case, you can copy the link and paste it into your browser.



Click on the registration link sent in an email

Step 6: Enter a password of your choice and re-enter the same password. Followed by this, click on the “Submit Password” button to complete the registration process.

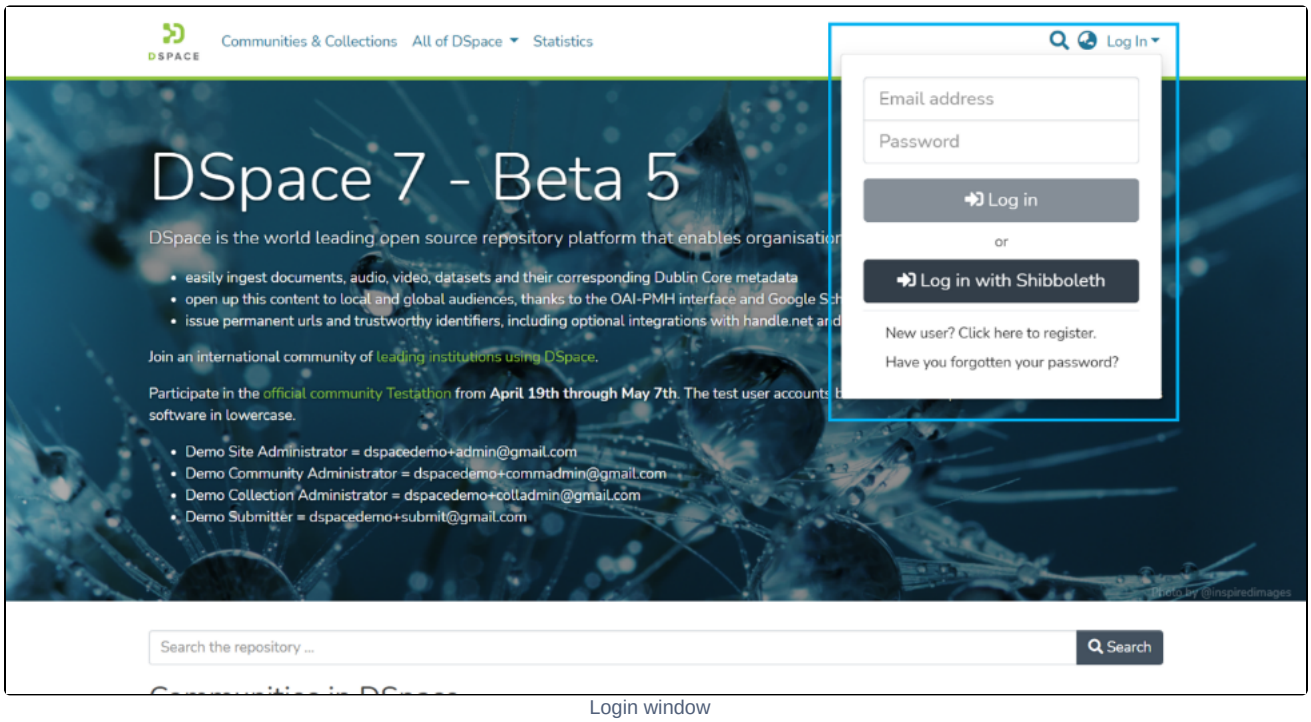


Enter a password and re-enter it to establish an account password

Add E Person – Registration by Administrator

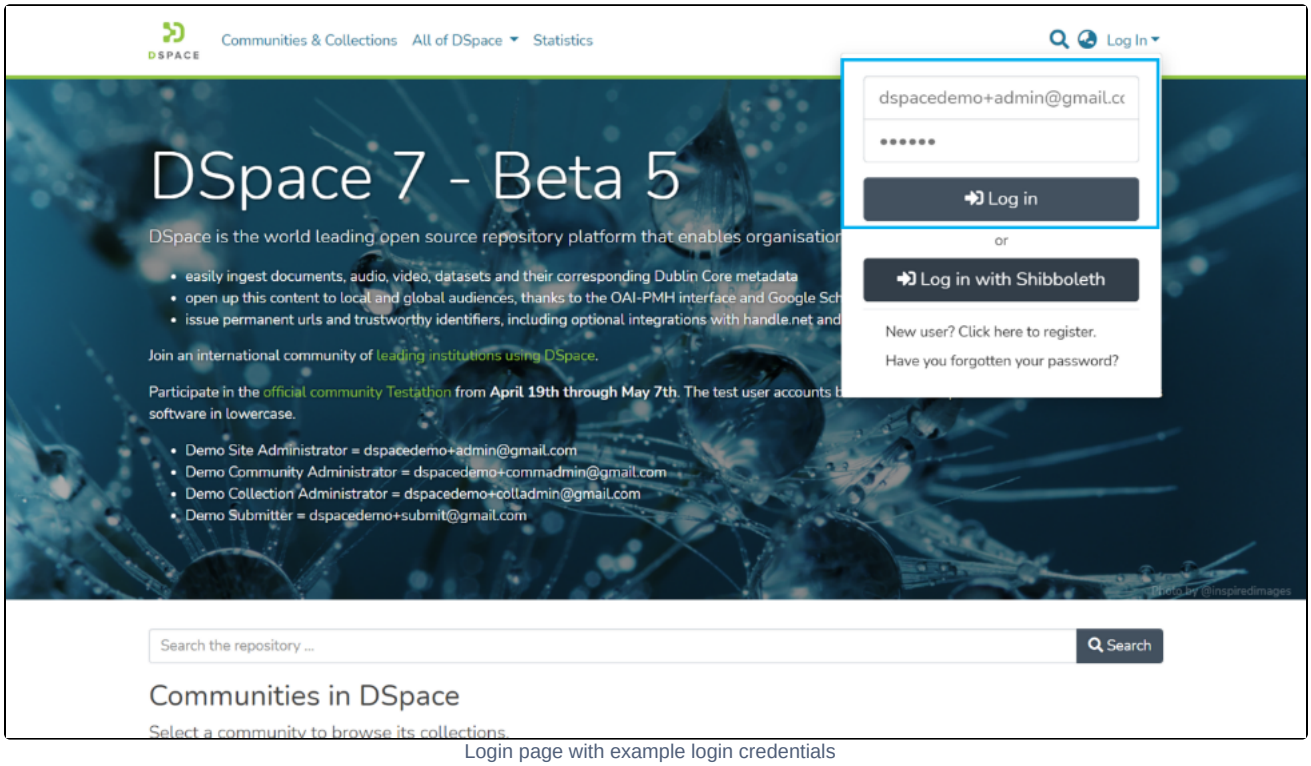
This process is for the user having System, Community, and Collection Administrator rights. Only users with these rights can manage E People.

Step 1: Go to DSpace’s home page and click on the “Log In” link appearing at the top right corner of the screen, and the pop-up will open, as illustrated below screen.



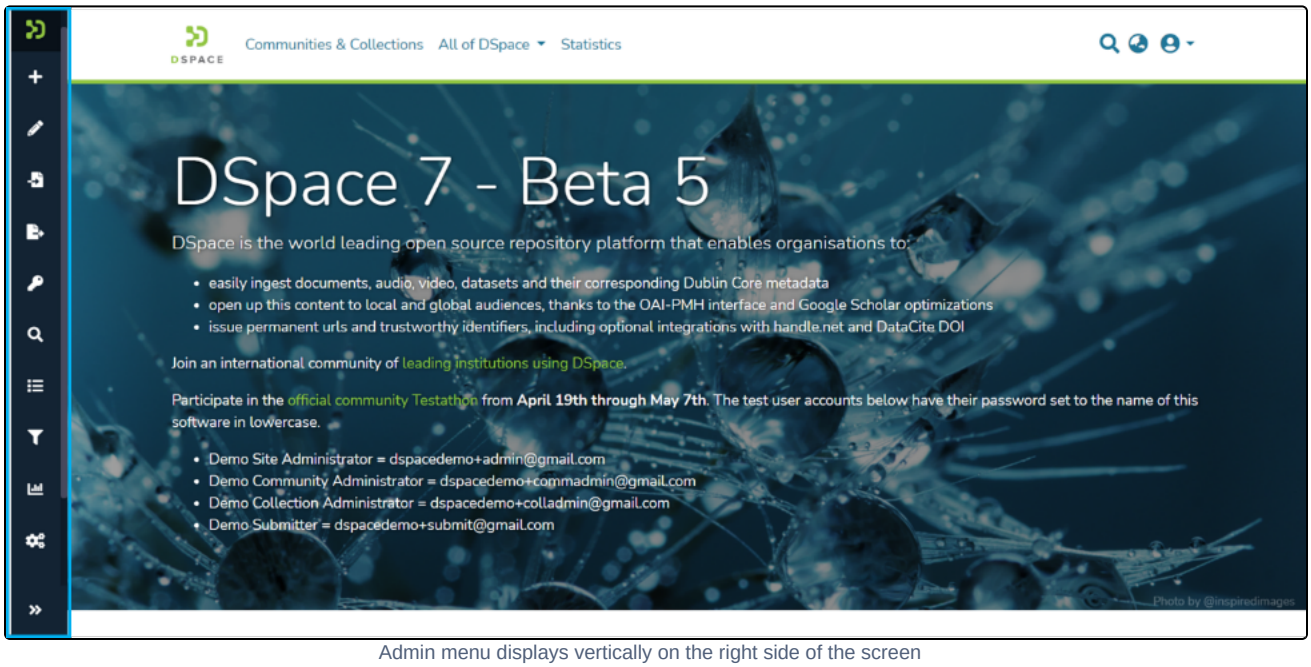
Login window

Step 2: Enter your user id and password and click on the login button for logging in to DSpace



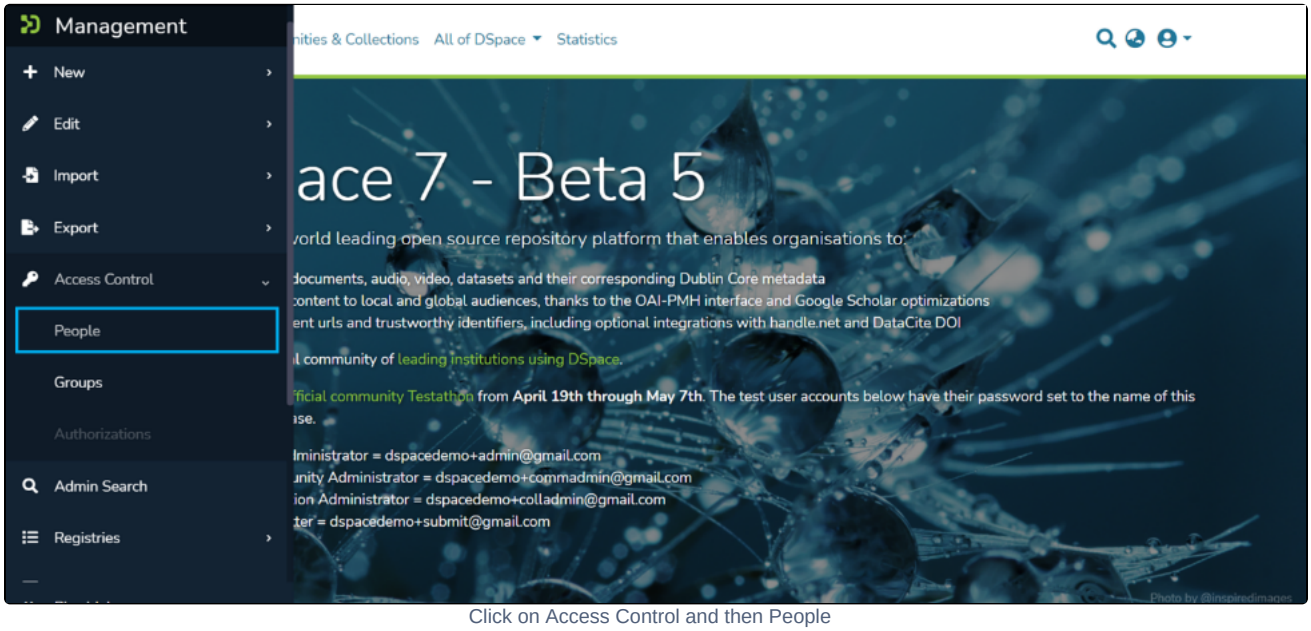
Login page with example login credentials

Step 3: Users with administrative rights will see the admin menu on the left-hand side of the screen, as shown below illustration.



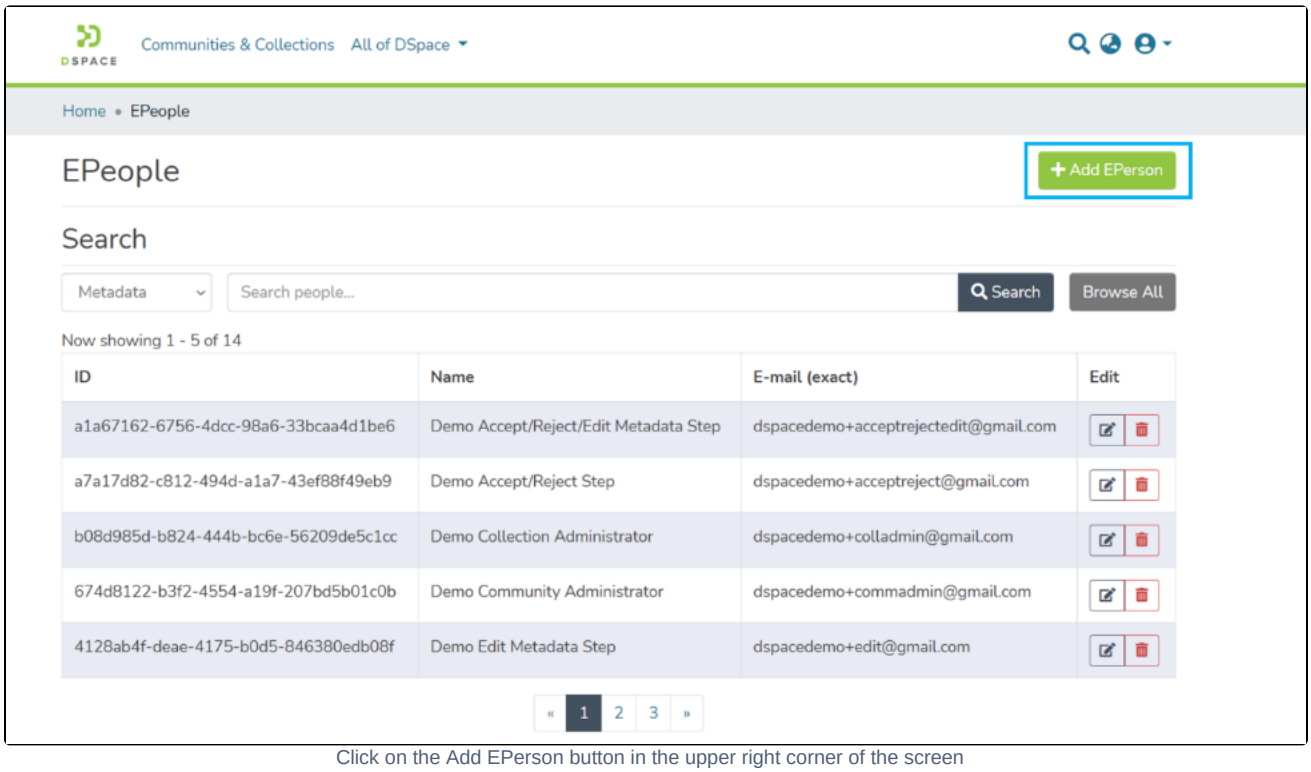
Admin menu displays vertically on the right side of the screen

Step 4: Rollover your cursor over the administration menu and click on Access control. Click on the People link to go to the EPeople module.



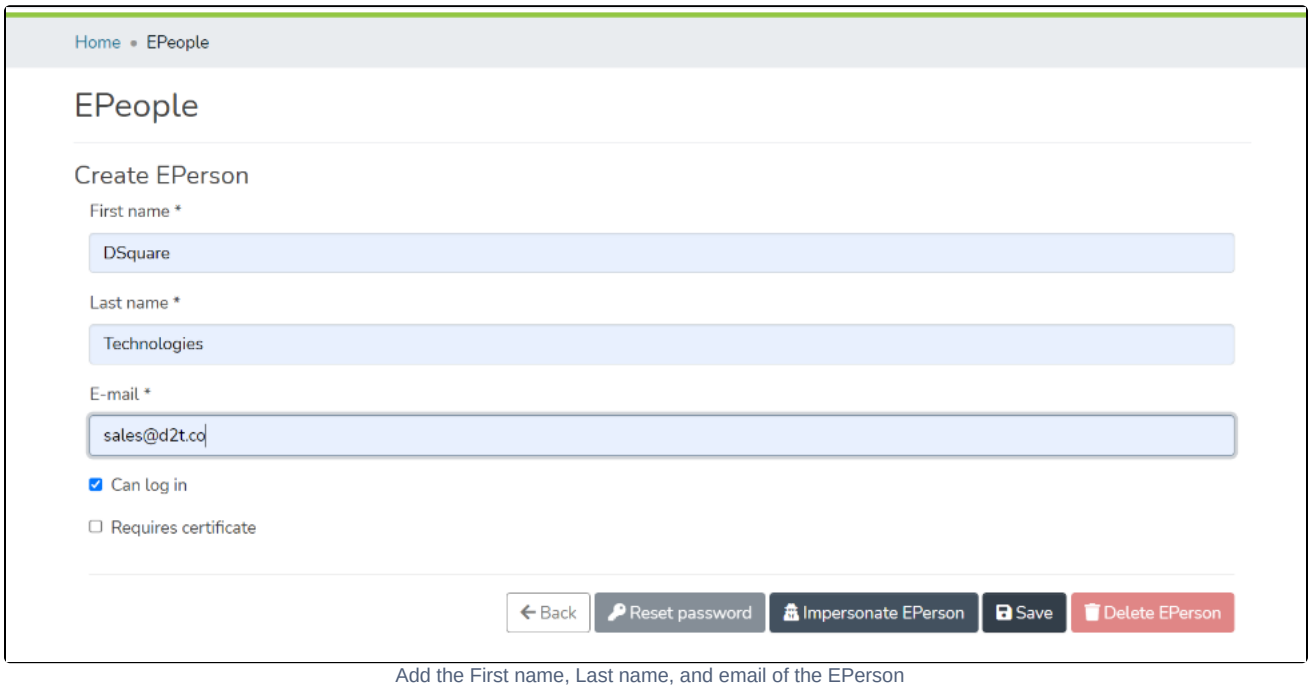
Click on Access Control and then People

Step 5: Click on the “Add EPerson” button for initiating the E-Person creation process.



Click on the Add EPerson button in the upper right corner of the screen

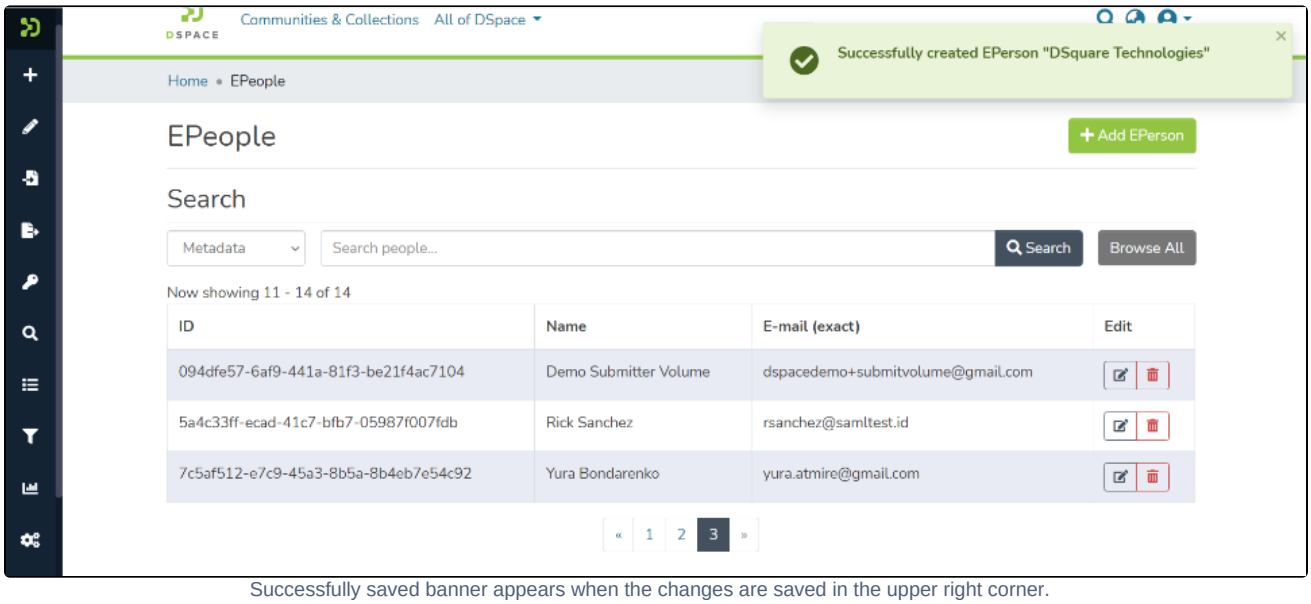
Step 6: Enter “First Name”, “Last Name,” and “E-Mail”. Select the “Can log in” check box for enabling id to login into the DSpace.



Add the First name, Last name, and email of the EPerson

Step 7: Upon populating values in mandatory fields “Save” button will get activated. Click on the Save button to complete the process.

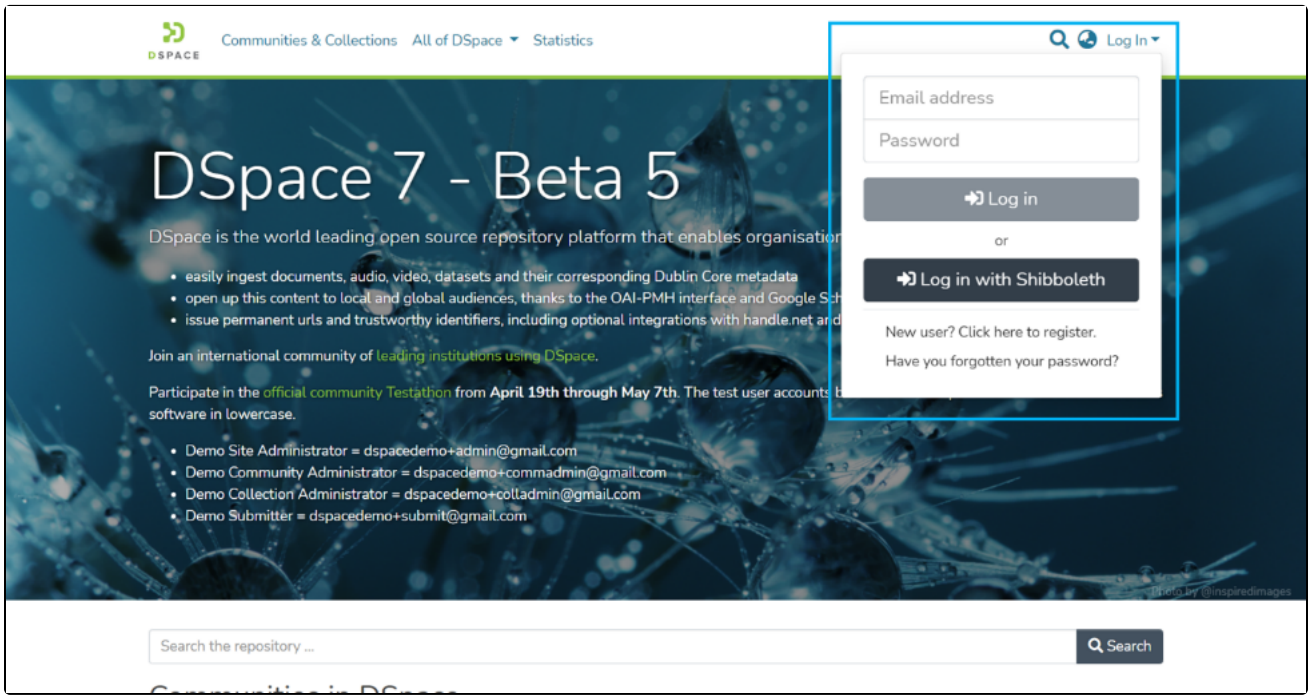
Step 8: Upon successfully adding EPerson in DSpace, you will see a success prompt on the screen.



Successfully saved banner appears when the changes are saved in the upper right corner.

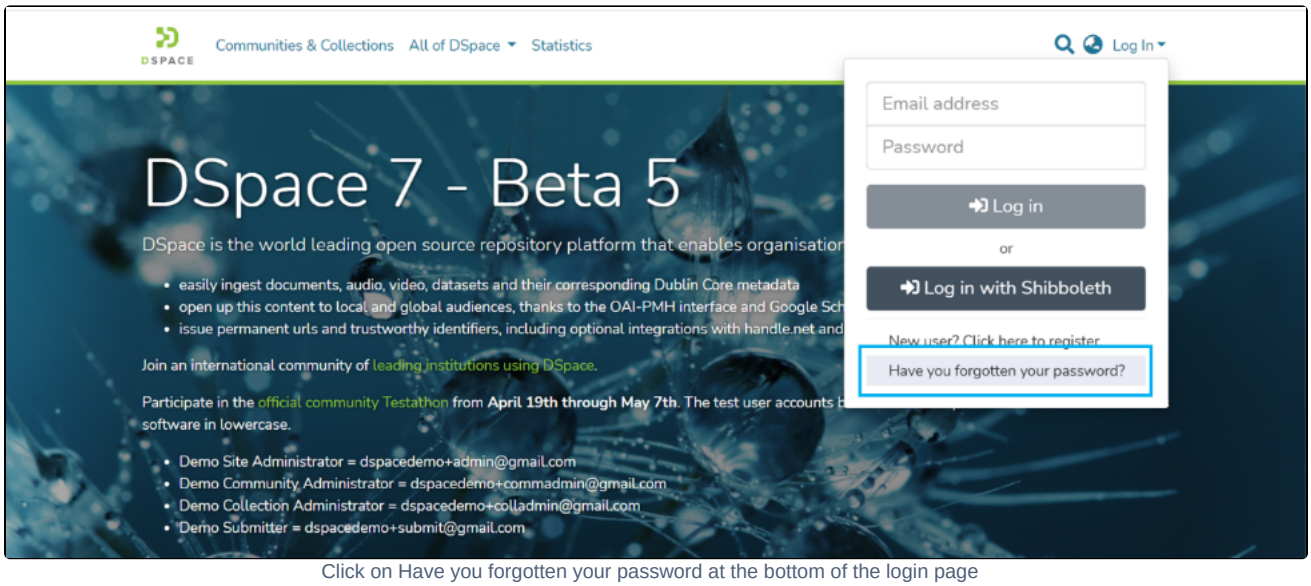
Step 9: The administrator can inform the user to generate its password using forgot password mechanism. The same is explained in the following steps.

Step 10: The user to go to the home page of DSpace and click on the “Log In” link appearing at the top right corner of the screen and pop up will open, as illustrated below screen.



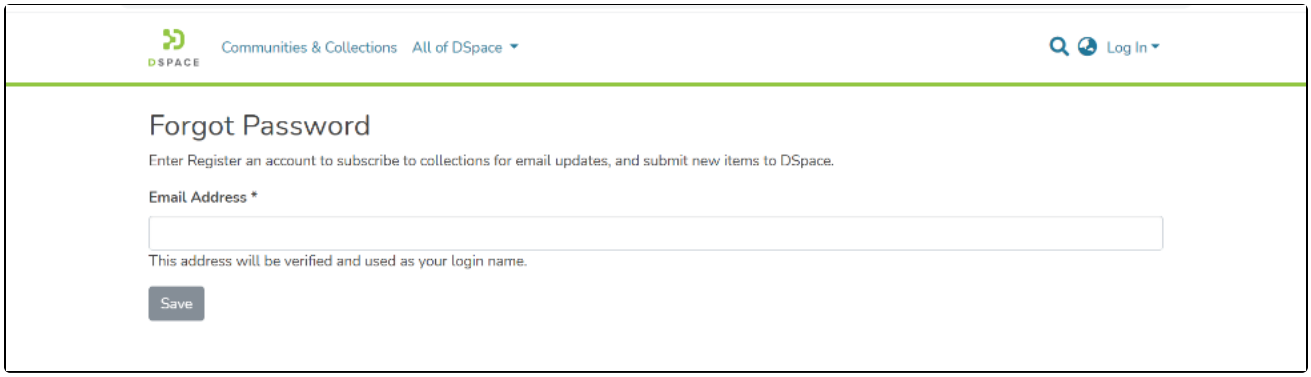
Login window

Step 11: Click on the “Have you forgotten your password?” password link reset page.



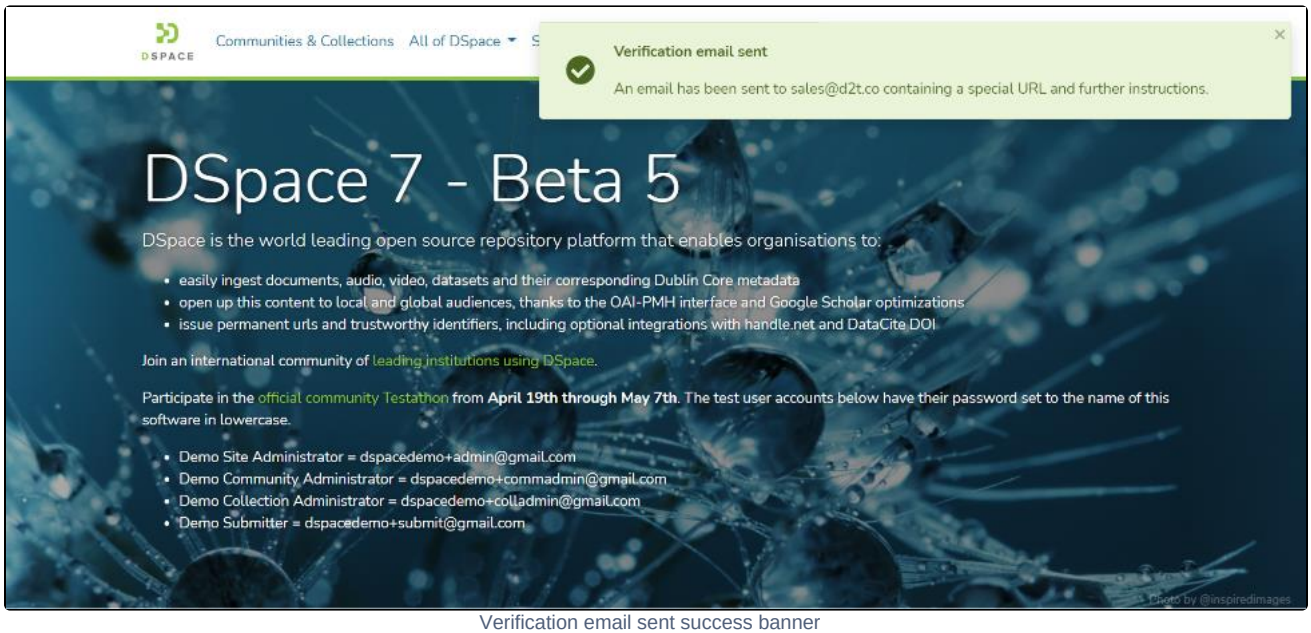
Click on Have you forgotten your password at the bottom of the login page

Step 12: Enter the Email ID used for Eperson creation in the DSpace and click on the “Save” button.



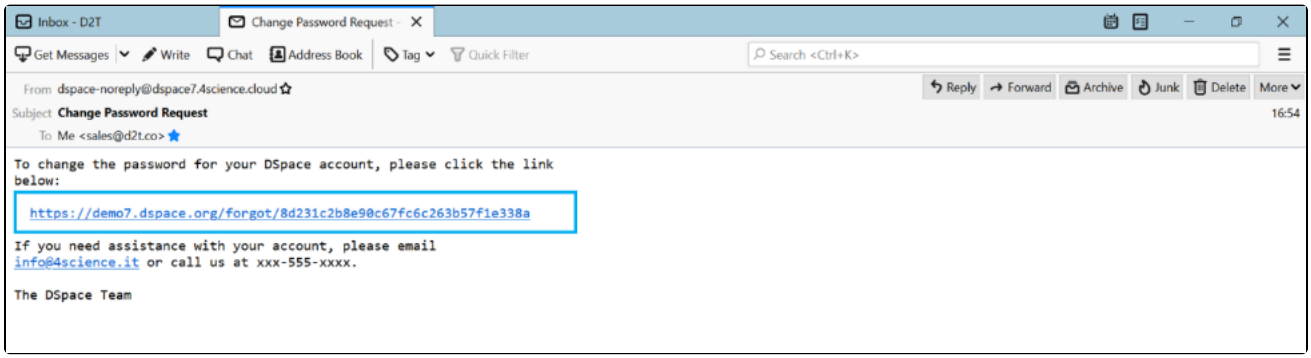
Enter the email address in the Forgot Password field

Step 13: A prompt confirming a password reset link dispatch to the registered email id will appear and the user will be redirected to the home page. This E Mail will go from the communication mail ID registered in the DSpace instance.



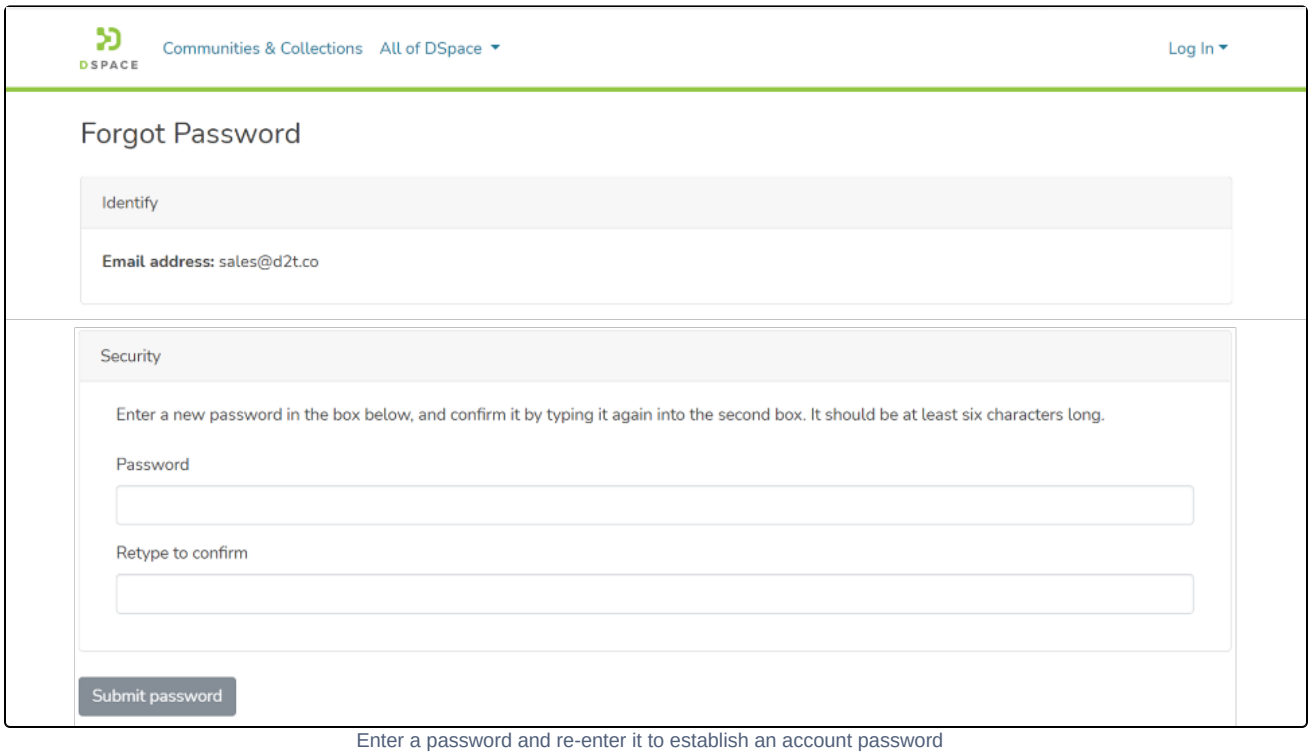
Verification email sent success banner

Step 14: Click on the unique registration link received in the email to continue the password reset process. Suppose the user's email client or server security settings do not show hyperlinks. In that case, the user can copy the link and paste it into its browser.



Click on the registration link sent in an email

Step 15: Enter a password of your choice and re-enter the same password. Followed by this, click on the "Submit Password" button to complete the password generation process.



Enter a password and re-enter it to establish an account password

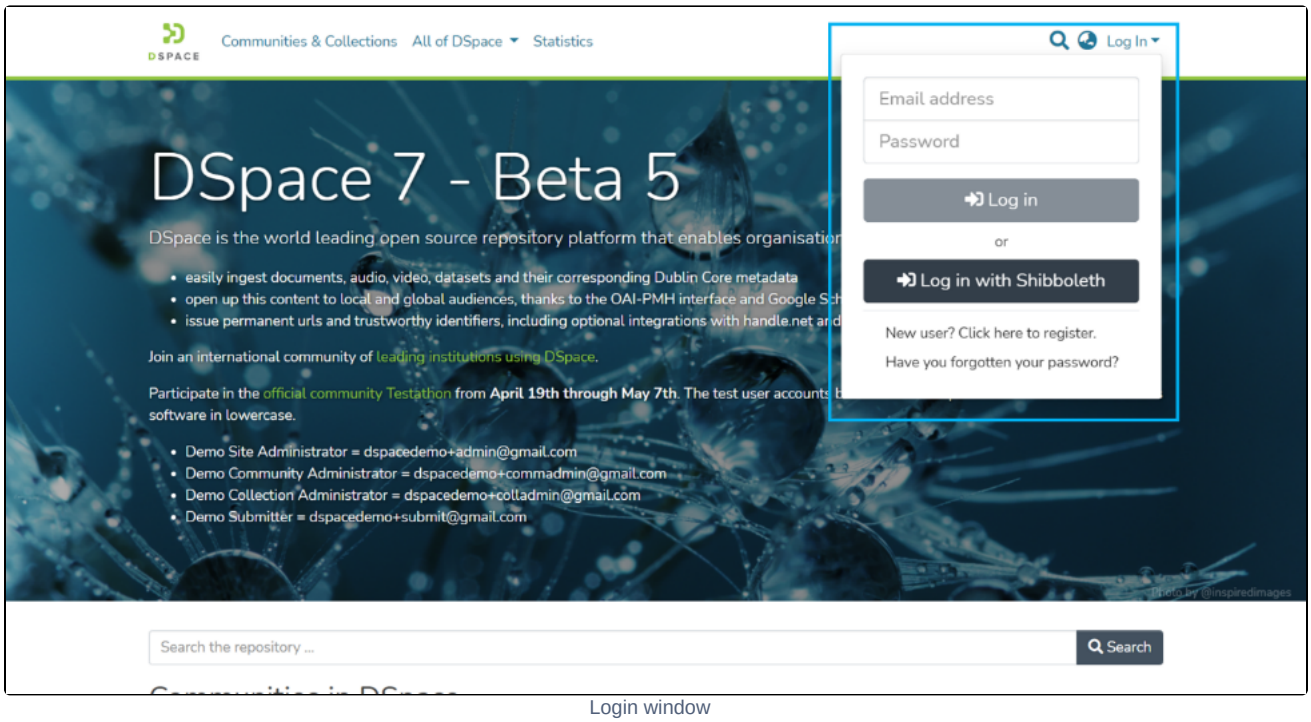
Update an Eperson

Eperson updates can be performed by users having System, Community, and Collection Administrator rights. These users can perform the following activities:

1. Update Eperson details
2. Manage Eperson Log in
3. Impersonate Eperson
4. Manage user groups membership
5. Delete Eperson from DSpace

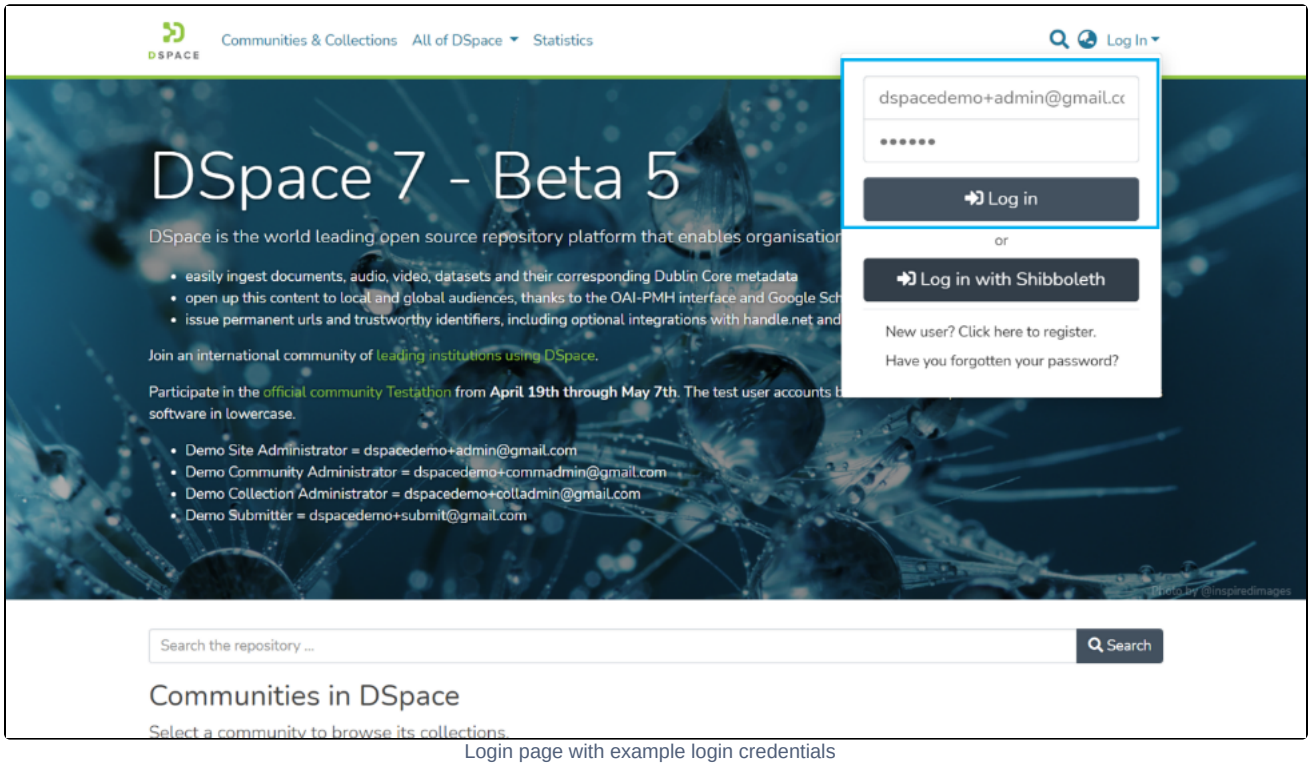
Update an Eperson – Update details & Manage Log in

Step 1: Go to DSpace's home page and click on the "Log In" link appearing at the top right corner of the screen, and the pop-up will open, as illustrated below screen.



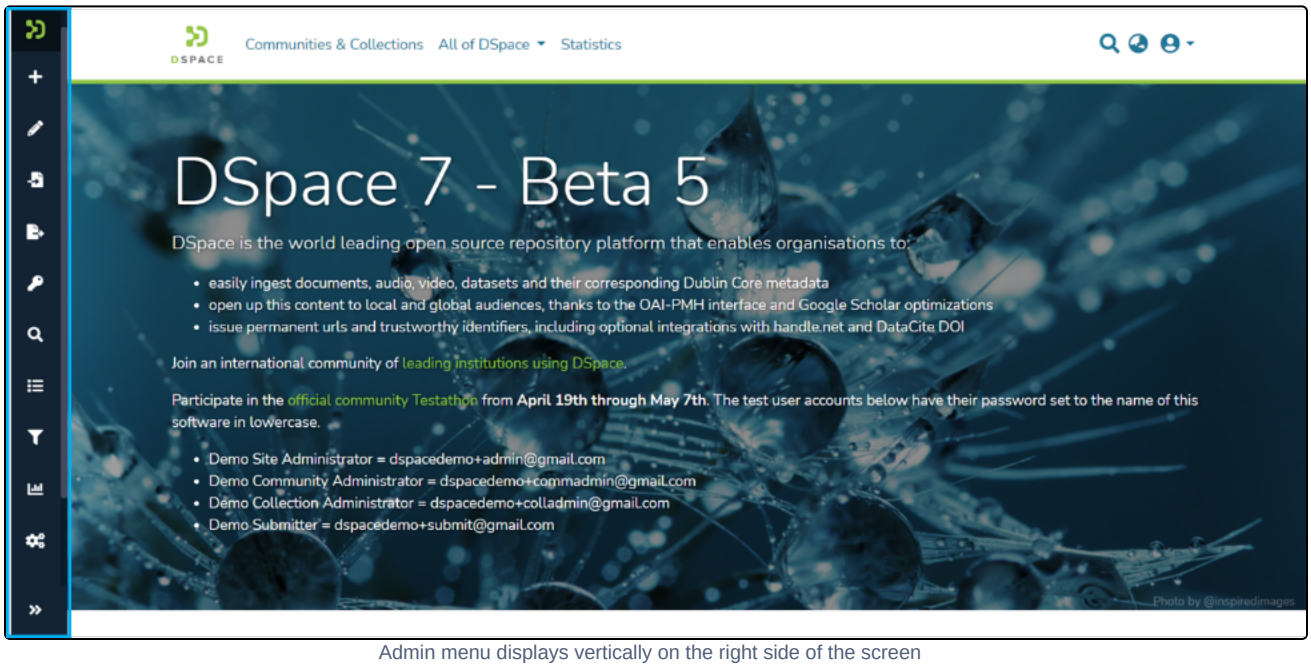
Login window

Step 2: Enter your user id and password and click on the login button for logging in to DSpace.



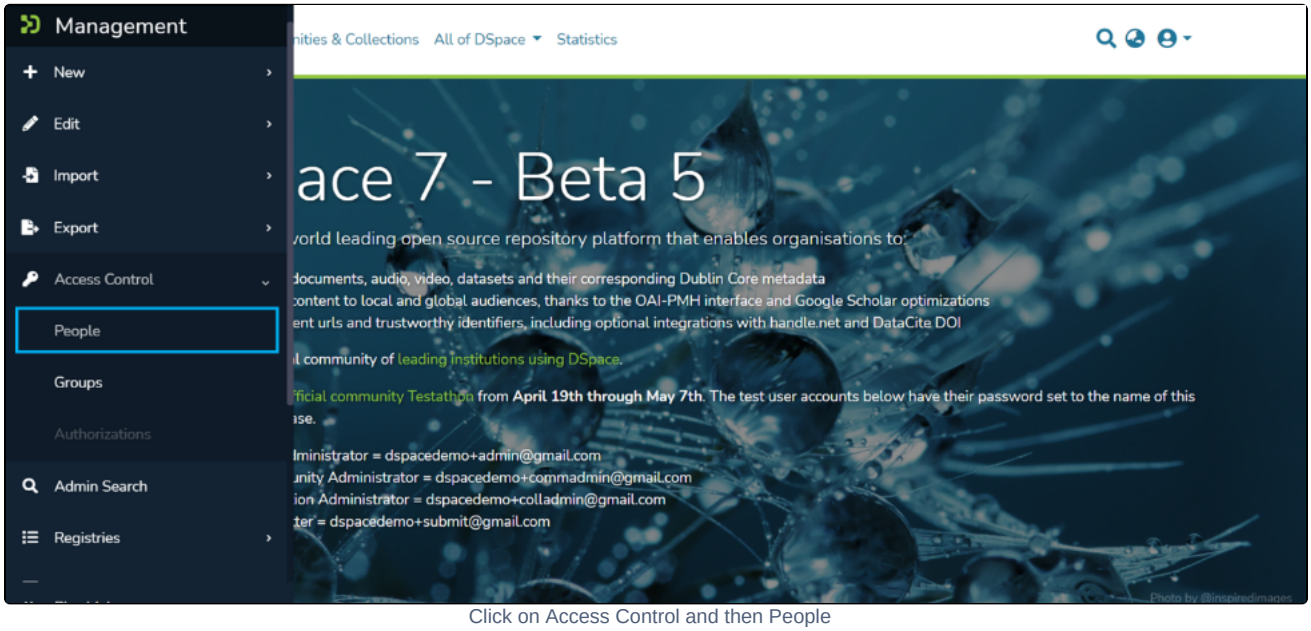
Login page with example login credentials

Step 3: Users with administrative rights will see the admin menu on the left-hand side of the screen, as shown below illustration.



Admin menu displays vertically on the right side of the screen

Step 4: Rollover your cursor over the administration menu and click on Access control. Click on the People link to go to the EPeople module.

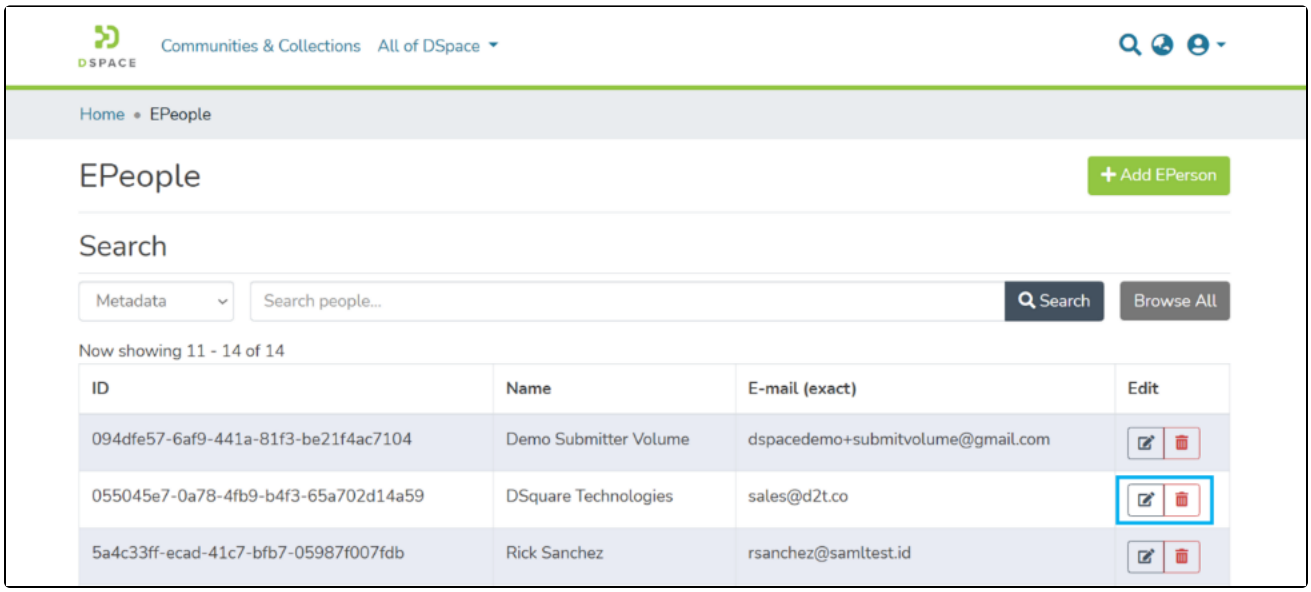


Click on Access Control and then People

Step 5: Scroll or search the eperson's name in the epeople list.

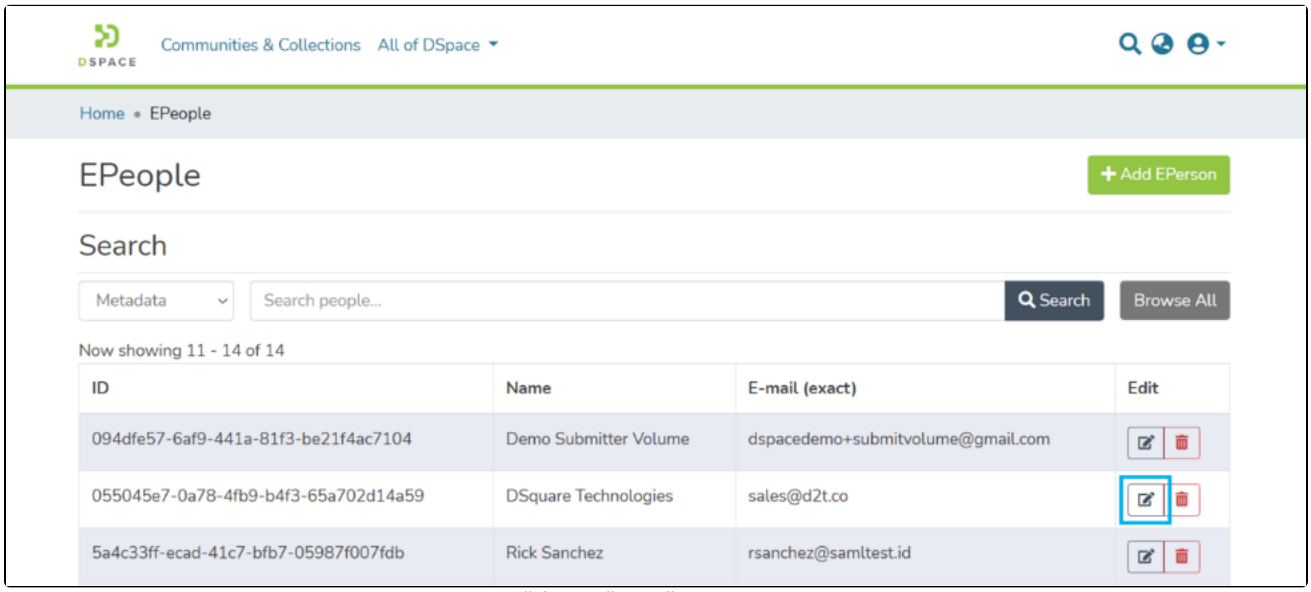
Step 6: Epeople table has the following elements:

- ID – Unique ID generated in DSpace for each Eperson
- Name – Full name of Eperson as entered during registration
- Email – Email id used for registering user
- Edit – Buttons showing various actions available for Eperson management. It has "Edit" and "Delete buttons. Click on the "Edit" button if you want to edit details of the Eperson, or click on the "Delete" button if you want to delete Eperson from DSpace permanently.



EPeople view

Step 7: Click on the “Edit” button to continue with Eperson editing.

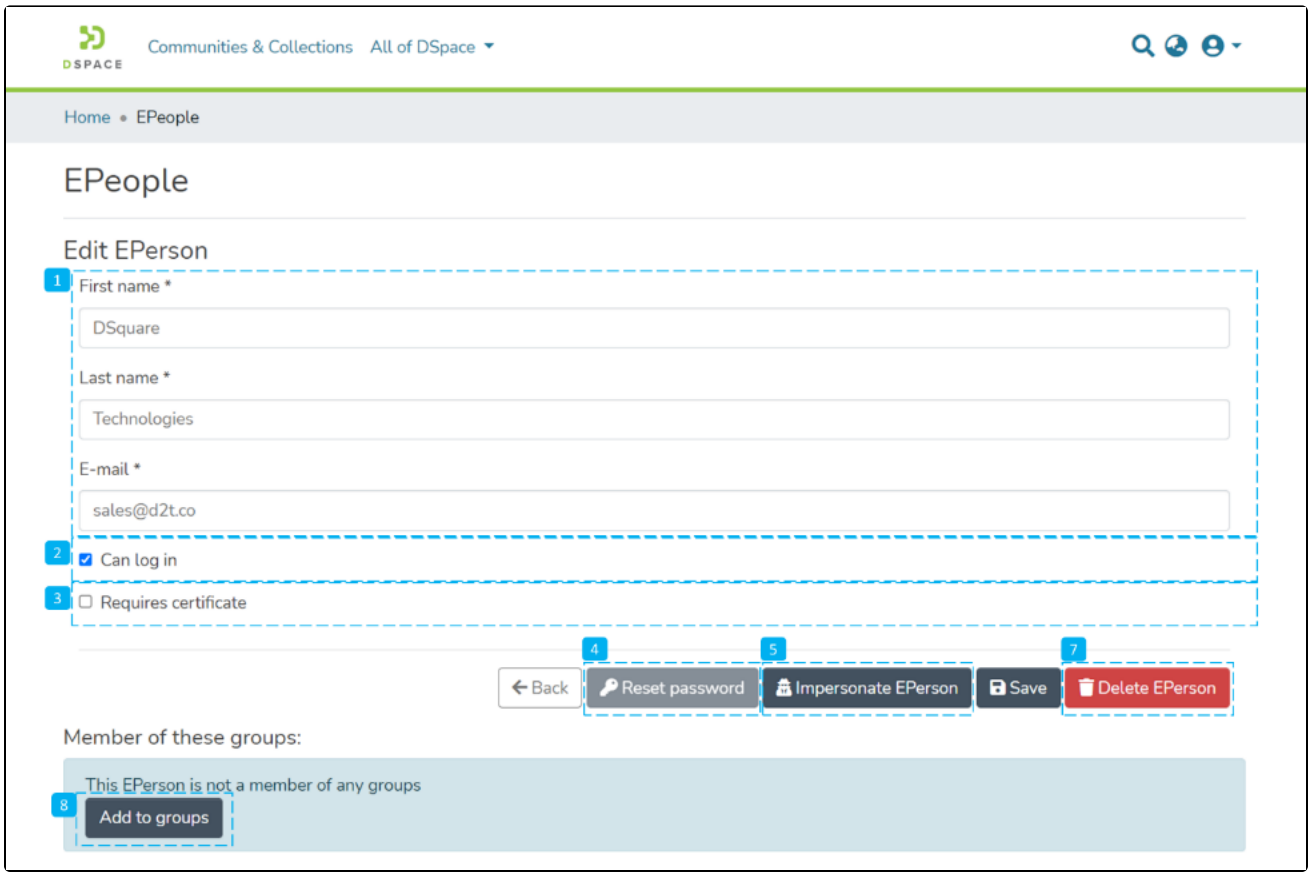


Click on Edit to edit an EPerson

Step 8: You can make the following updates on the Edit Eperson page

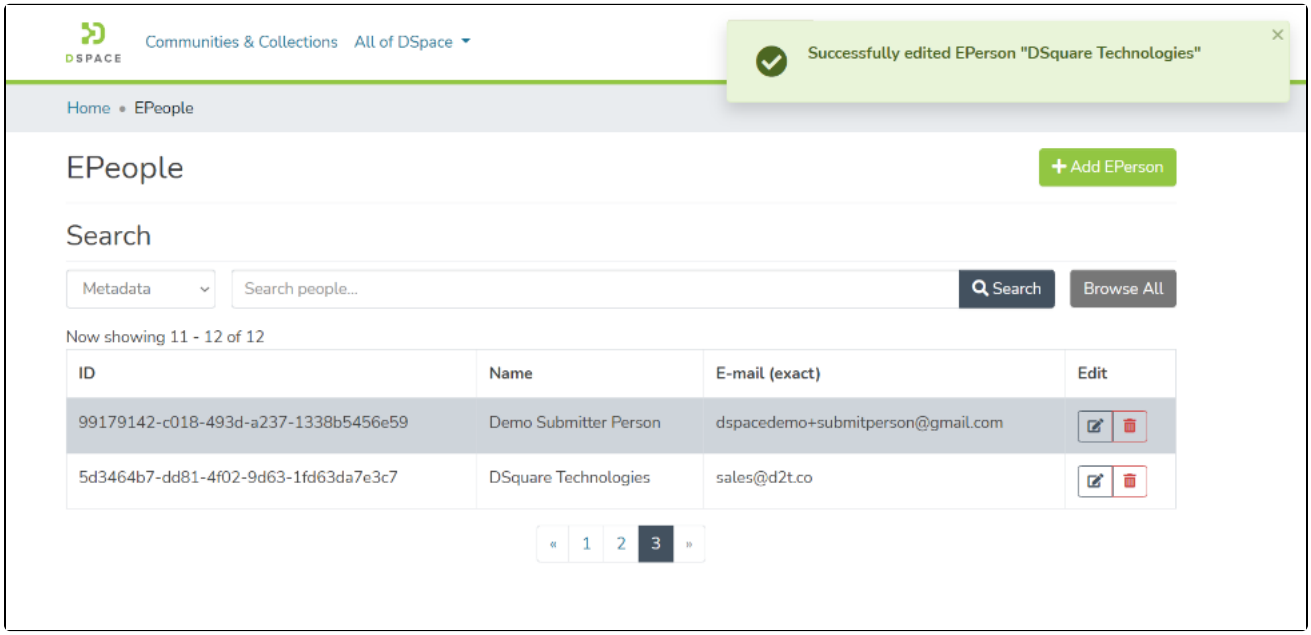
1. Update Eperson details – You can update First Name, Last Name, and Email ID as the user’s identification details.
2. Enable/Disable User Login – Uncheck this option to disable the user’s login into the DSpace without deleting it. This option is helpful for scenarios where users need to be disabled from logging in and performing specific actions temporarily.
3. Requires Certificate – Configure this option if a certificate is to be used for the login.
4. Reset Password – Password can be reset for Eperson using this option. A Password field should be present for utilizing this option.
5. Impersonate Eperson – The user with rights to Impersonate EPerson can impersonate the selected Eperson and perform all activities that the Eperson is entitled to.
6. Delete Eperson – Using this option, Eperson can be deleted permanently from DSpace
7. Update User group(s) – This button allows the administrator to add selected Eperson to multiple user groups.

You can make options # 1 to 3 and click on Save for updating the Eperson record.



Administrative options for managing EPeople

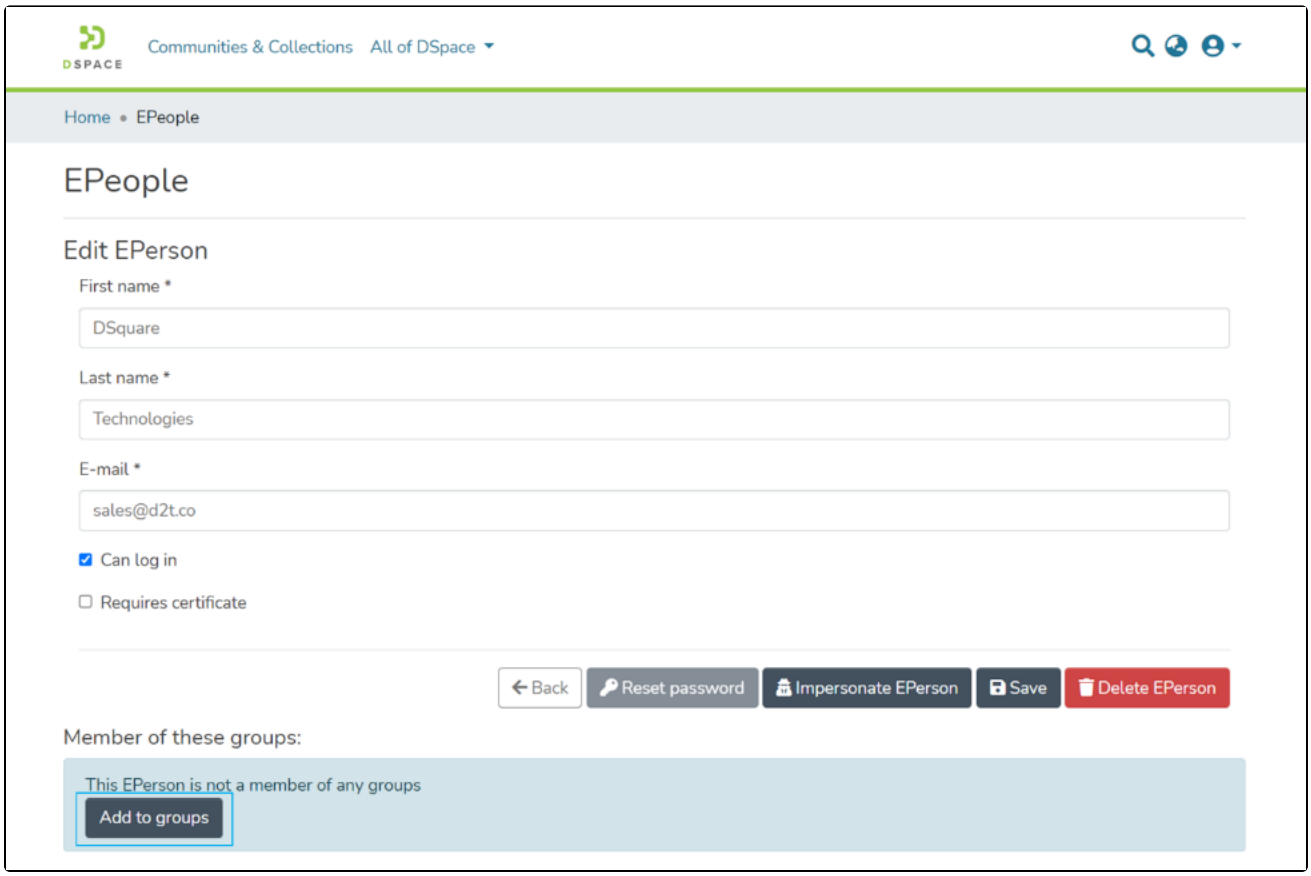
Step 9: Upon successful execution of the update, you will notice a success prompt on the screen.



Successfully edited banner

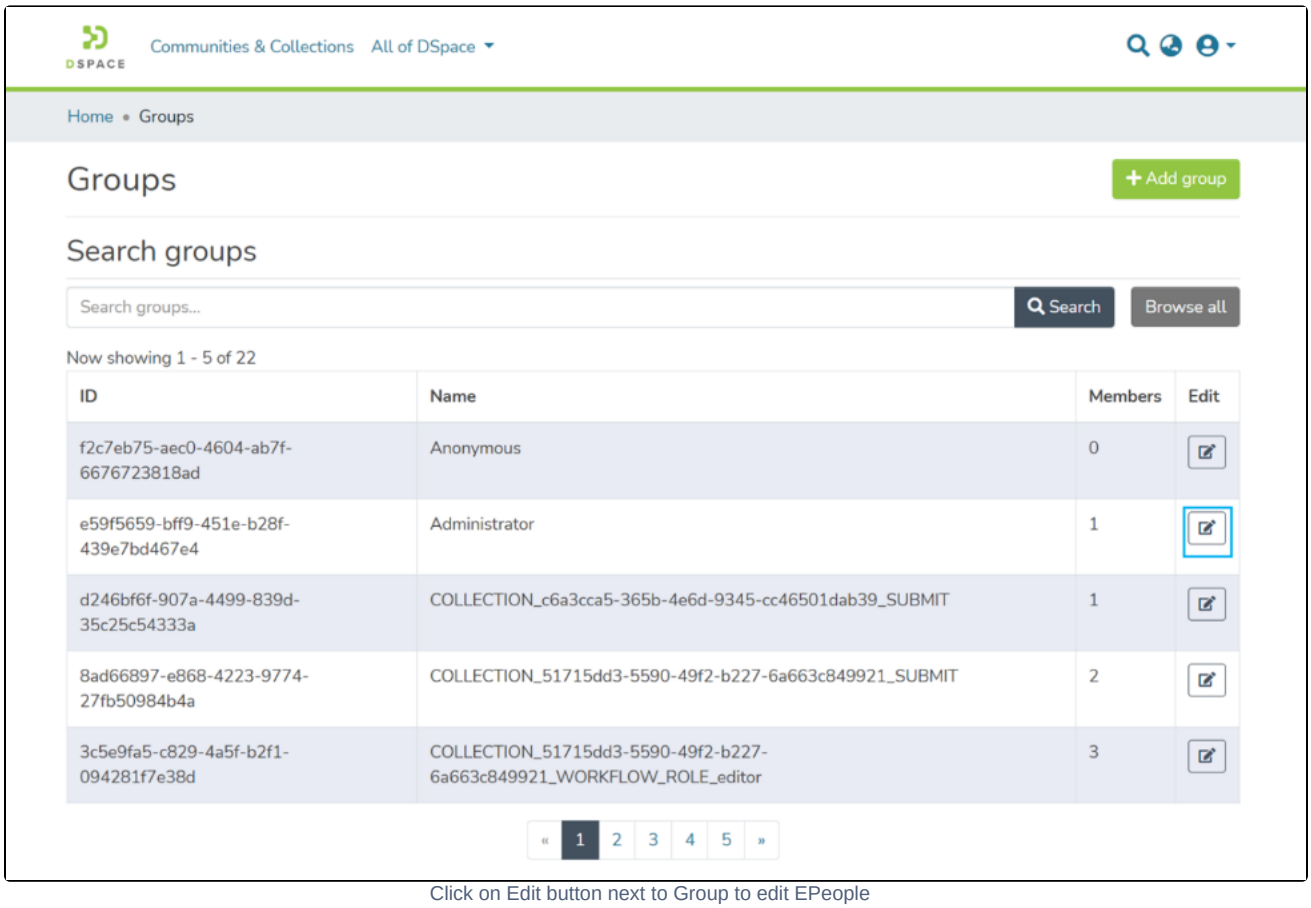
Update an Eperson – Manage user groups membership

Step 10: Click on the “Add to group” button to initiate the process of adding selected Eperson to a user group.



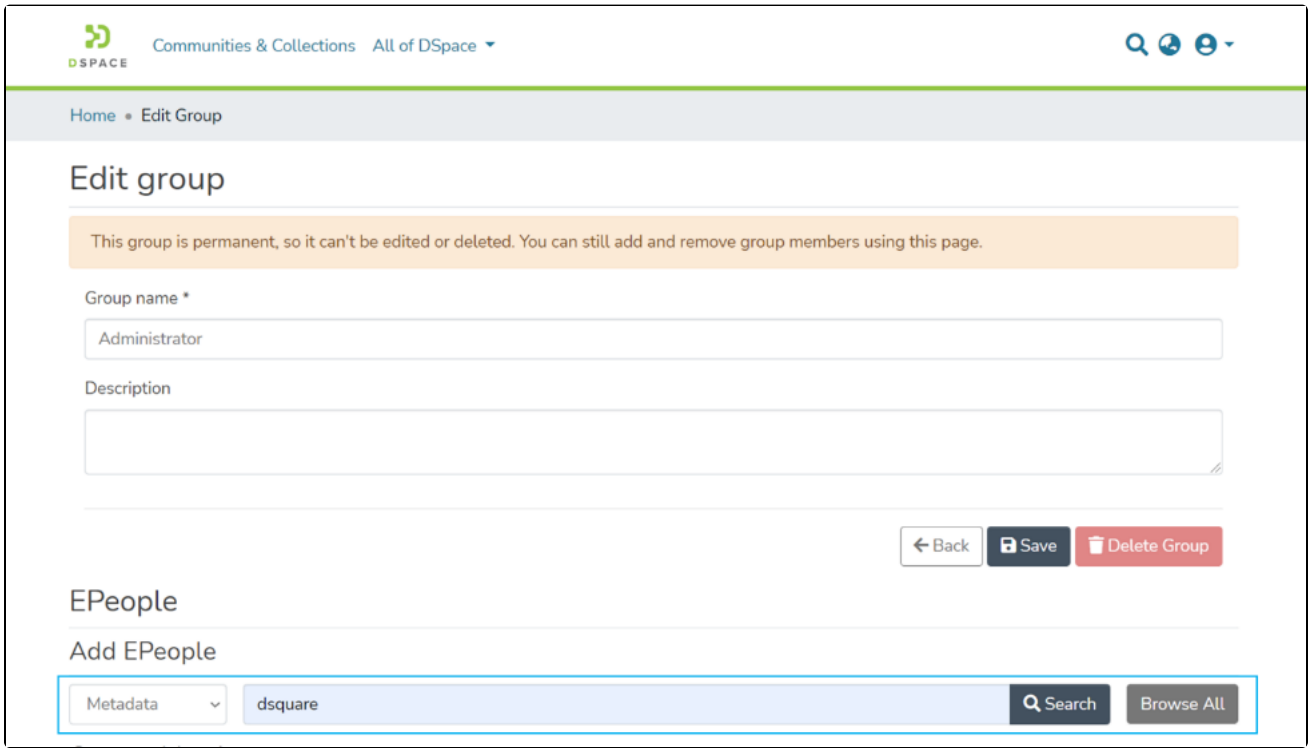
Click Add to Groups to add EPerson to Groups

Step 11: Please click on the “Edit” button next to the user group you want to select for this Eperson.



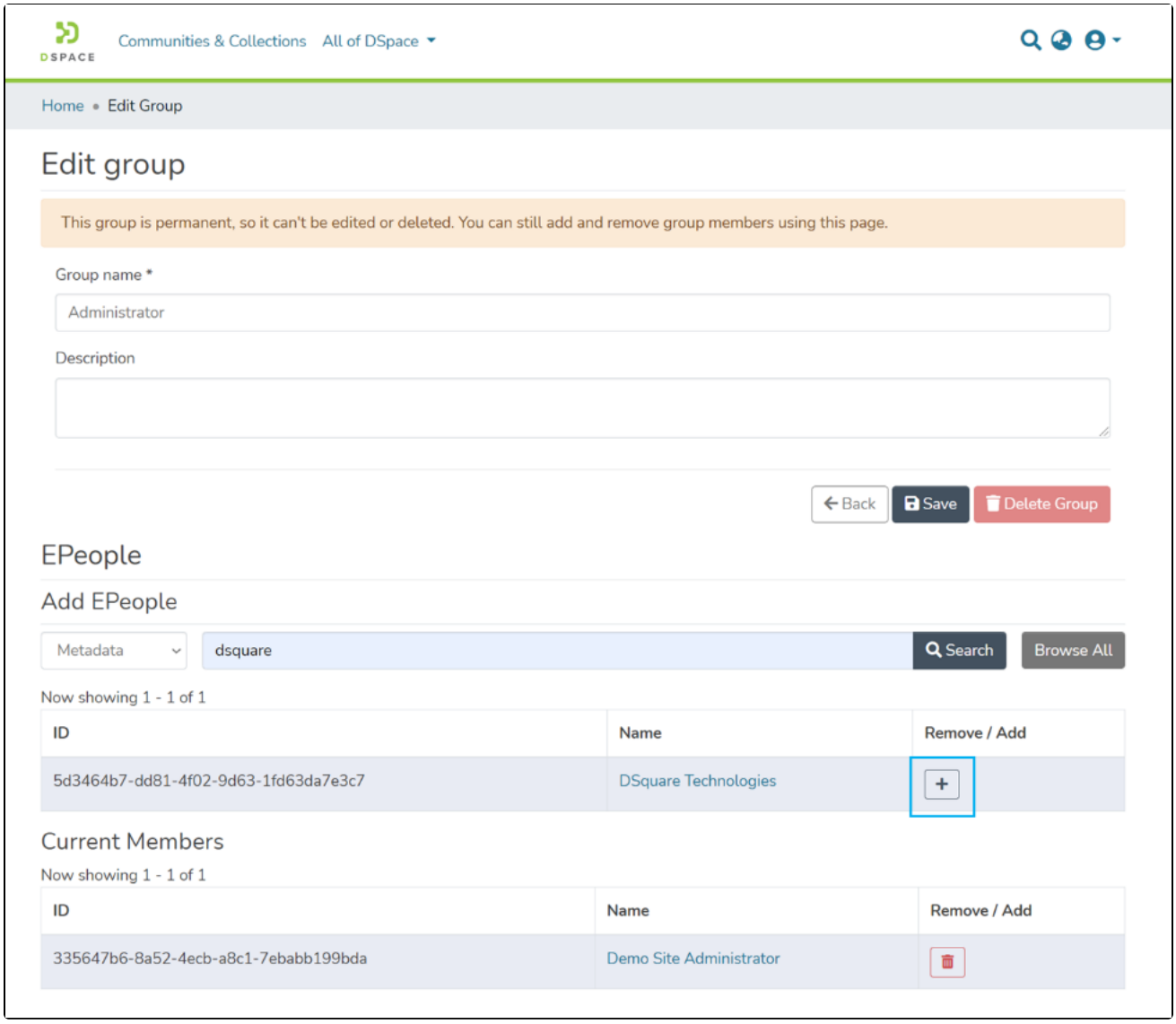
Click on Edit button next to Group to edit EPeople

Step 12: Enter the eperson's name or any other metadata value in the search field to find the target user.

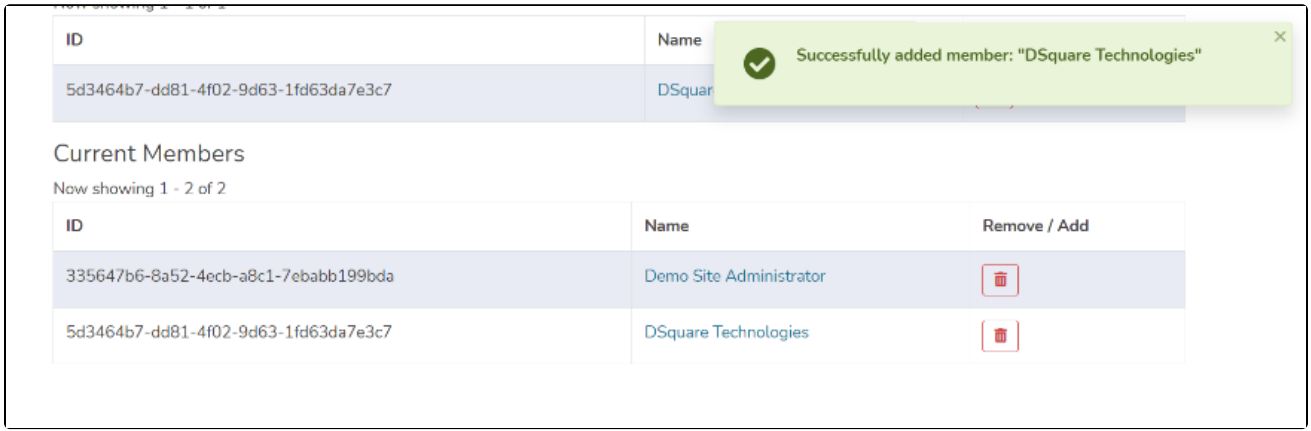


Type EPerson name or other data into metadata field to search for EPerson

Step 13: You will see Epeople appearing as a result of a search made. Click on the + button appearing next to Eperson you want to add to this group.

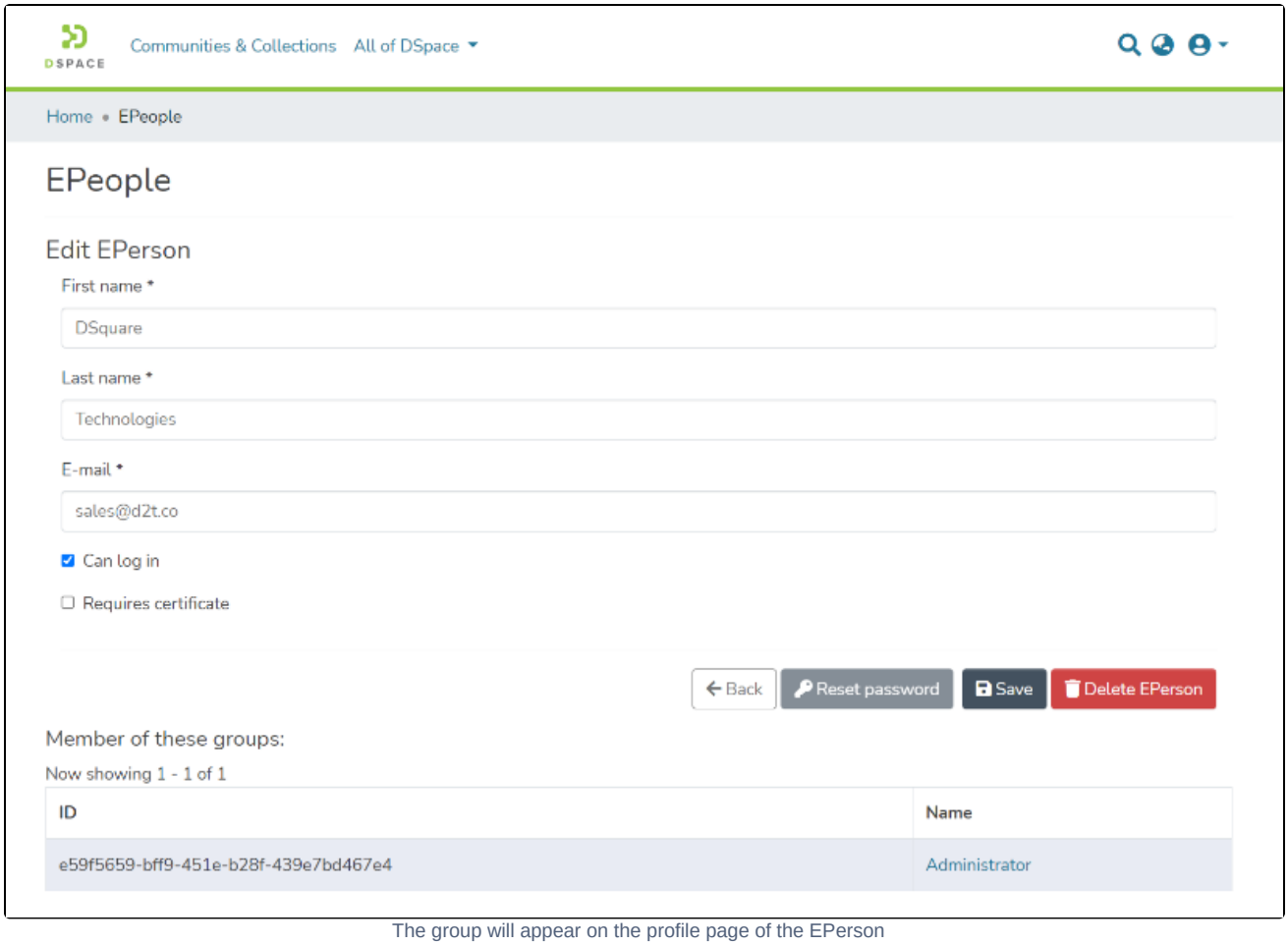


Step 14: Upon successfully adding Eperson to the group, a success prompt will appear on the screen, and Eperson will appear in the Current Members list of the group.



Successfully added Eperson banner

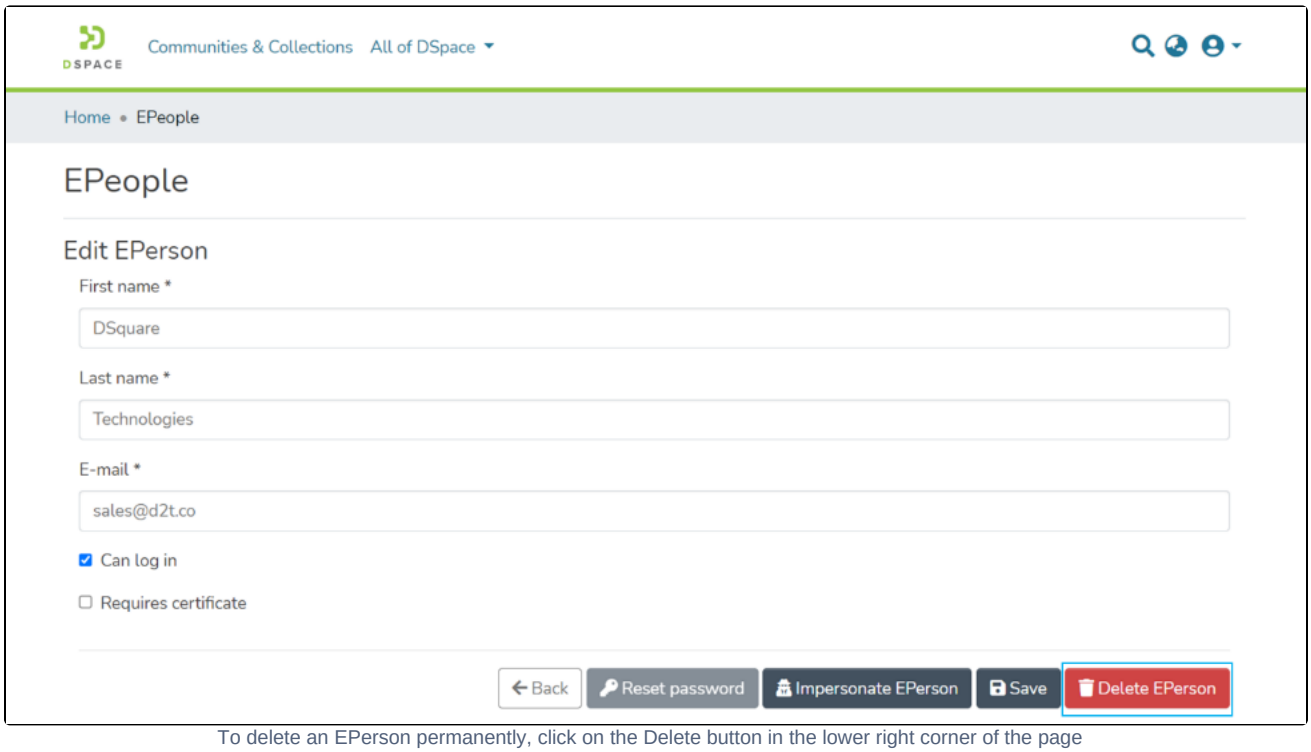
Step 15: You will see this group on the profile page of Eperson as well.



The group will appear on the profile page of the EPerson

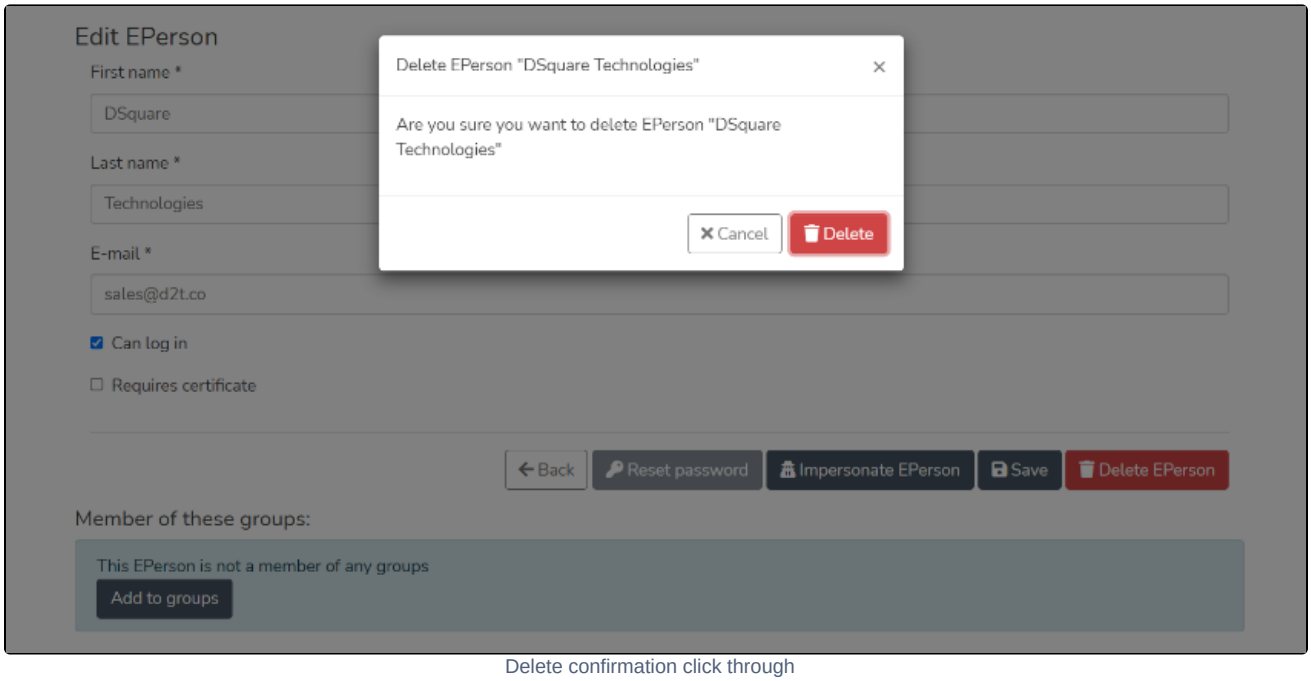
Update an Eperson – Delete Eperson

Step 16: Click the “Delete Eperson” button if you want to delete Eperson from DSpace permanently.



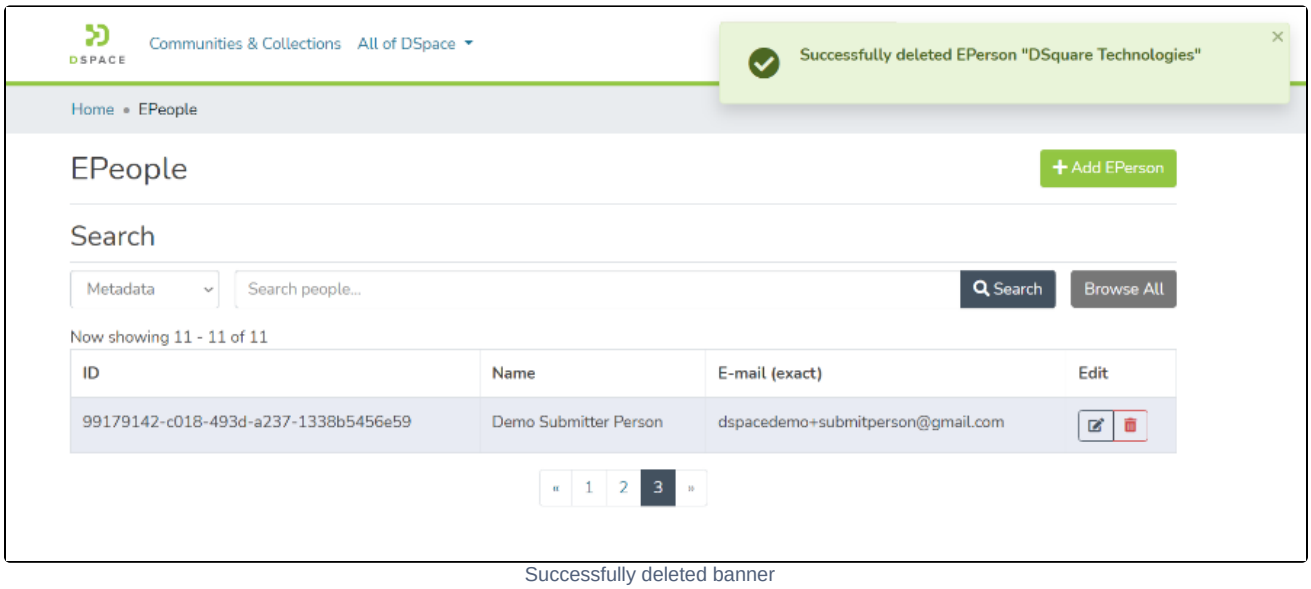
To delete an EPerson permanently, click on the Delete button in the lower right corner of the page

Step 17: DSpace will show you a confirmation prompt to re-confirm your decision of deleting the selected Eperson. Click on “Delete” if you want to continue with the deletion, or click on “Cancel” if you want to cancel the deletion.



Delete confirmation click through

Step 18: Successful deletion of the user will be confirmed by displaying a success promptly. You will be redirected to the Epeople page.



Successfully deleted banner

Create or manage a user group

User groups in DSpace are meant for creating a group of E-People. These groups can be assigned access rights to allow users to perform multiple activities or manage content access in the repository.

This section provides steps to add or update a User group in the application.

- Audience
- Add a user group
- Manage a user group
    - Update group details
    - Delete Group
    - Add/Manage EPeople
    - Add/Manage subgroups

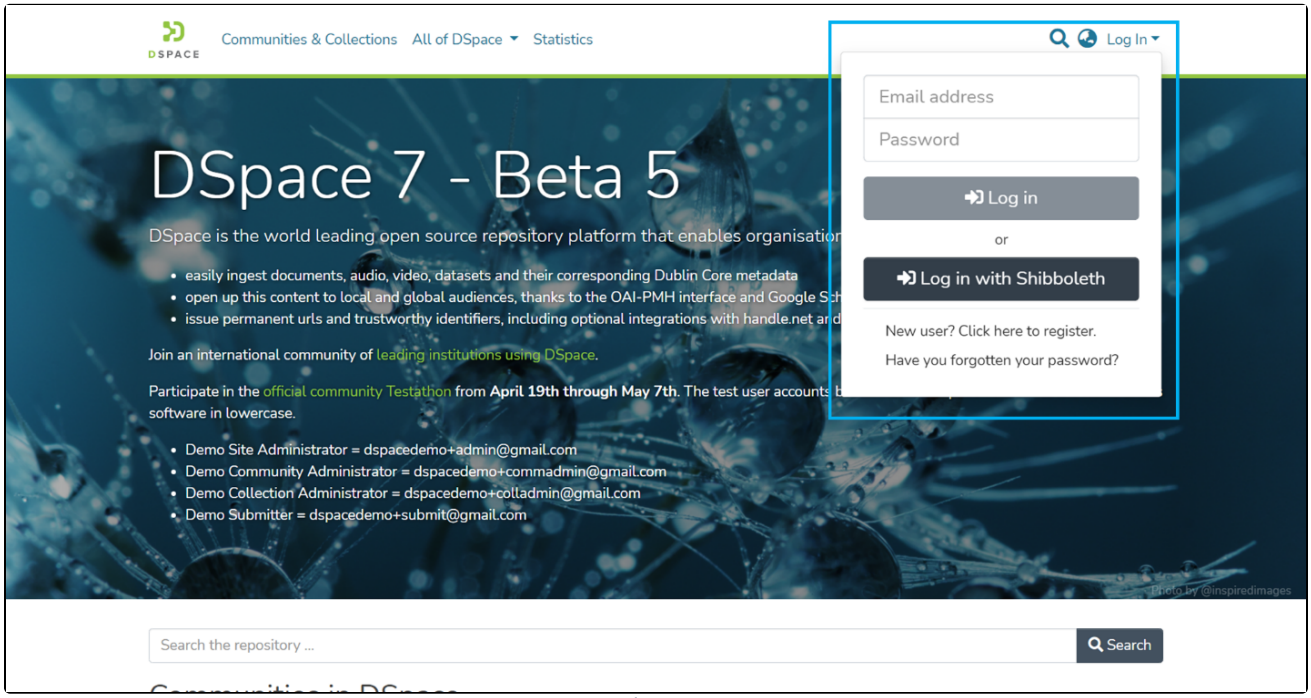
Audience

- Repository Administrator
- Community Administrator

Add a user group

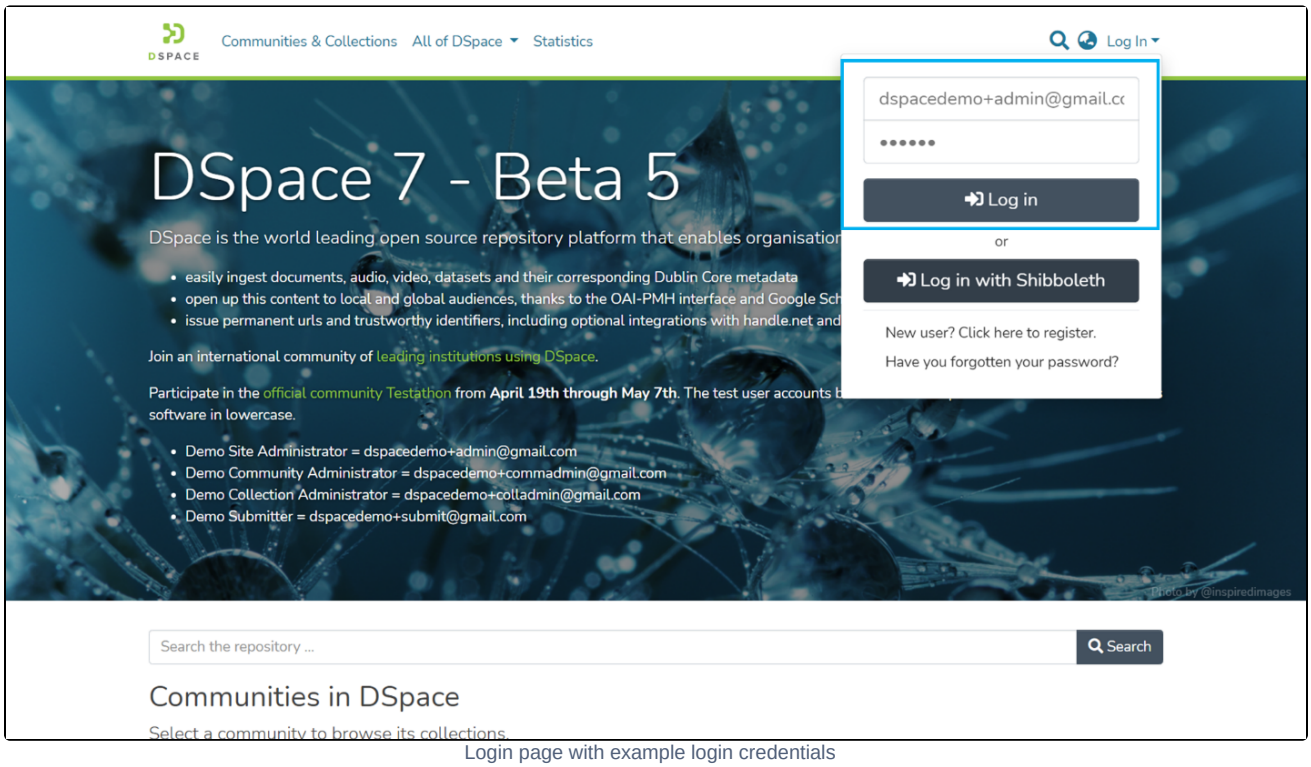
This process is for users having System and Community Administrator rights. As these user profiles can add and manage E People.

Step 1: Click on the "Log In" link at the top right corner of the DSpace's homepage, and the pop-up will open, as illustrated below the screen.



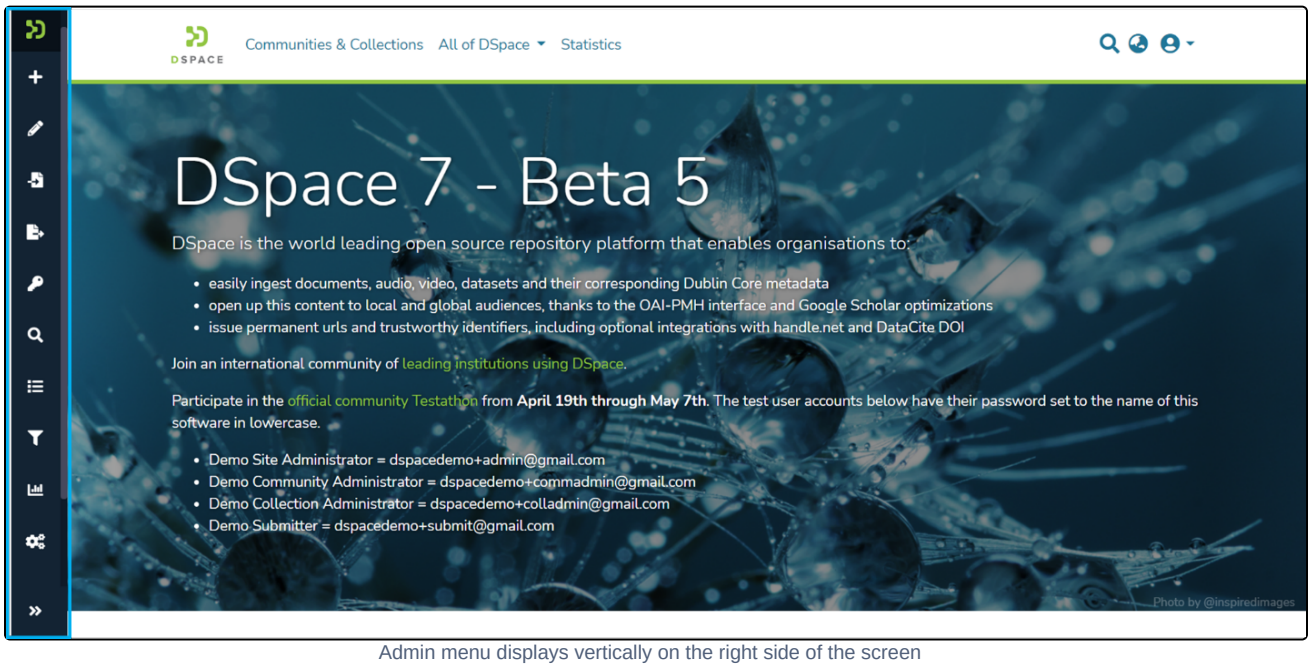
Login page

Step 2: Enter your user id and password and click on the "Log in" button to enter into DSpace.



Login page with example login credentials

Step 3: Users with administrative rights will see the admin menu on the left-hand side of the screen, as shown in the below illustration.



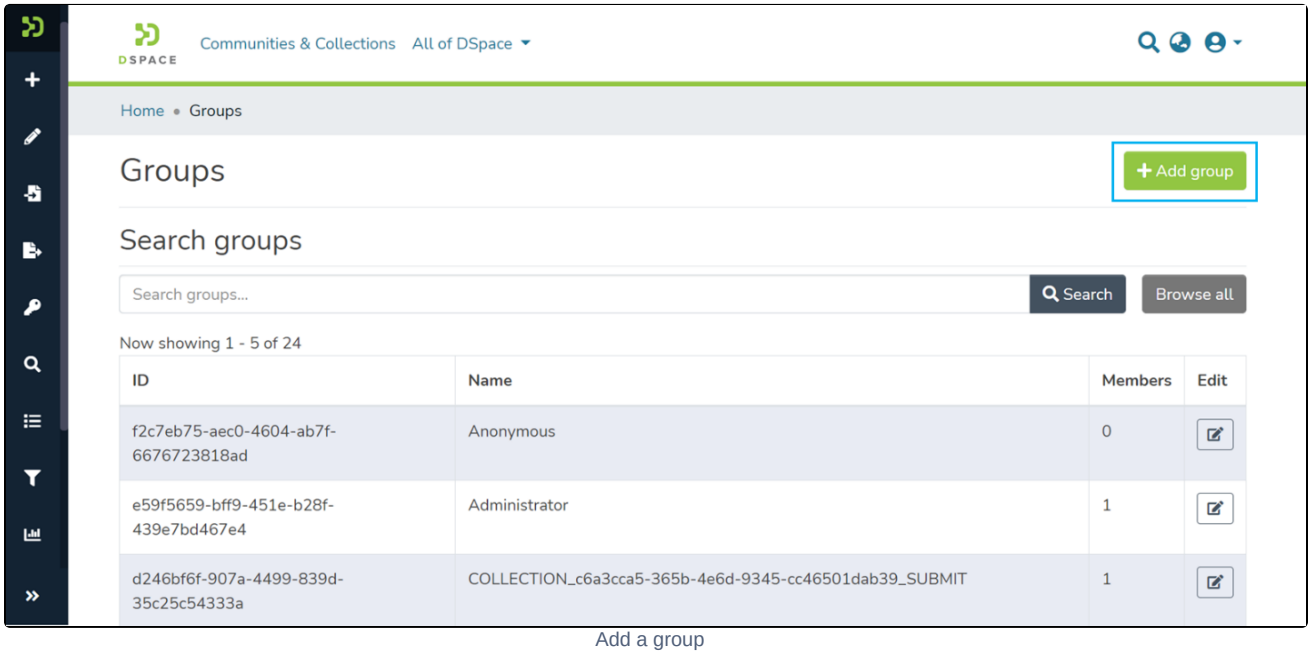
Admin menu displays vertically on the right side of the screen

Step 4: Rollover your cursor over the administration menu, click on "Access Control," and click on the "Groups" link.



In the admin menu, click on Access Control and then Groups

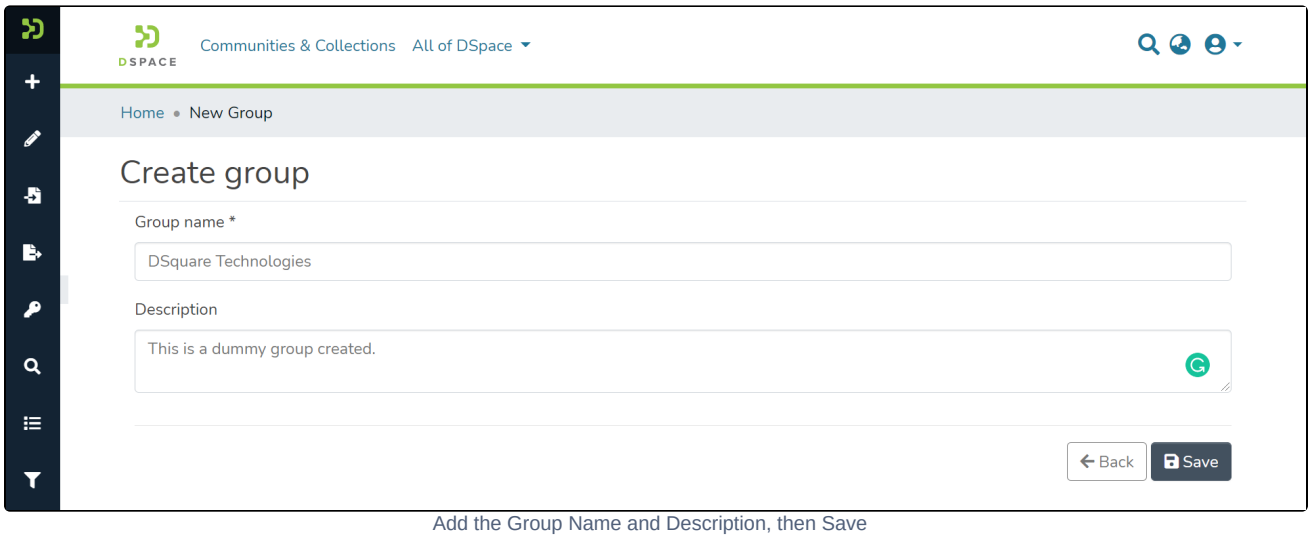
Step 5: Click on the "Add Group" button to create a Group.



Add a group

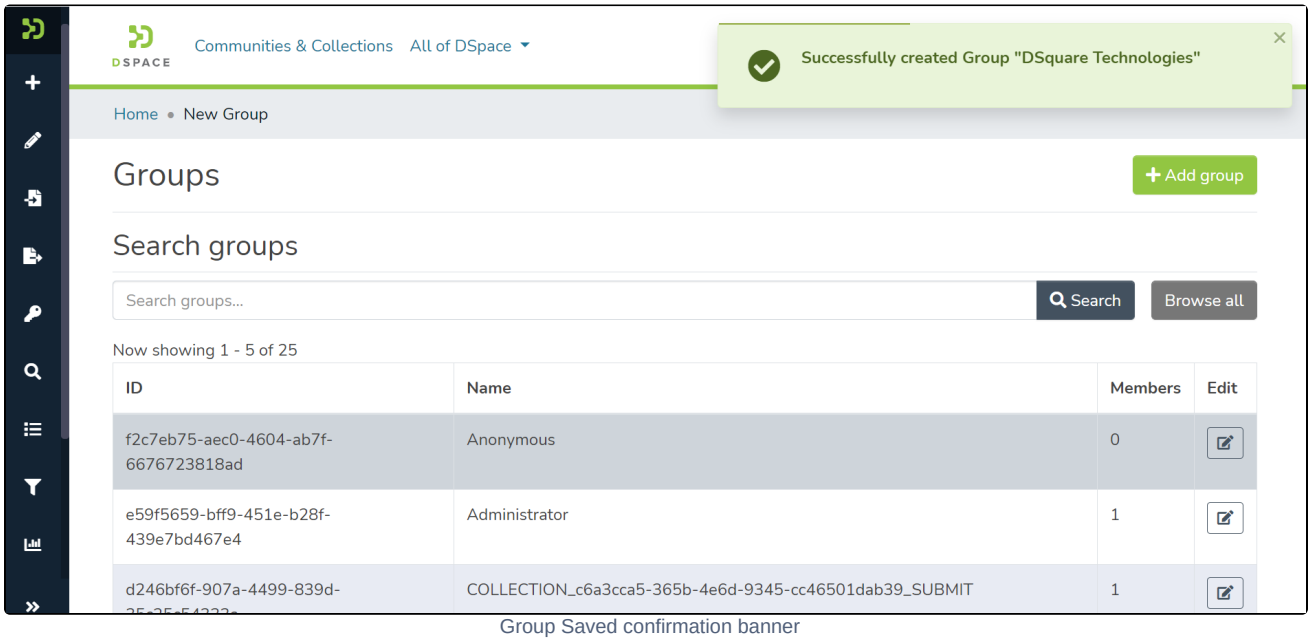
Step 6: The "Group Name" is compulsory. To benefit a broader user base, a description of the Group is a good practice.

The "Save" button will get activated upon entering the group name.



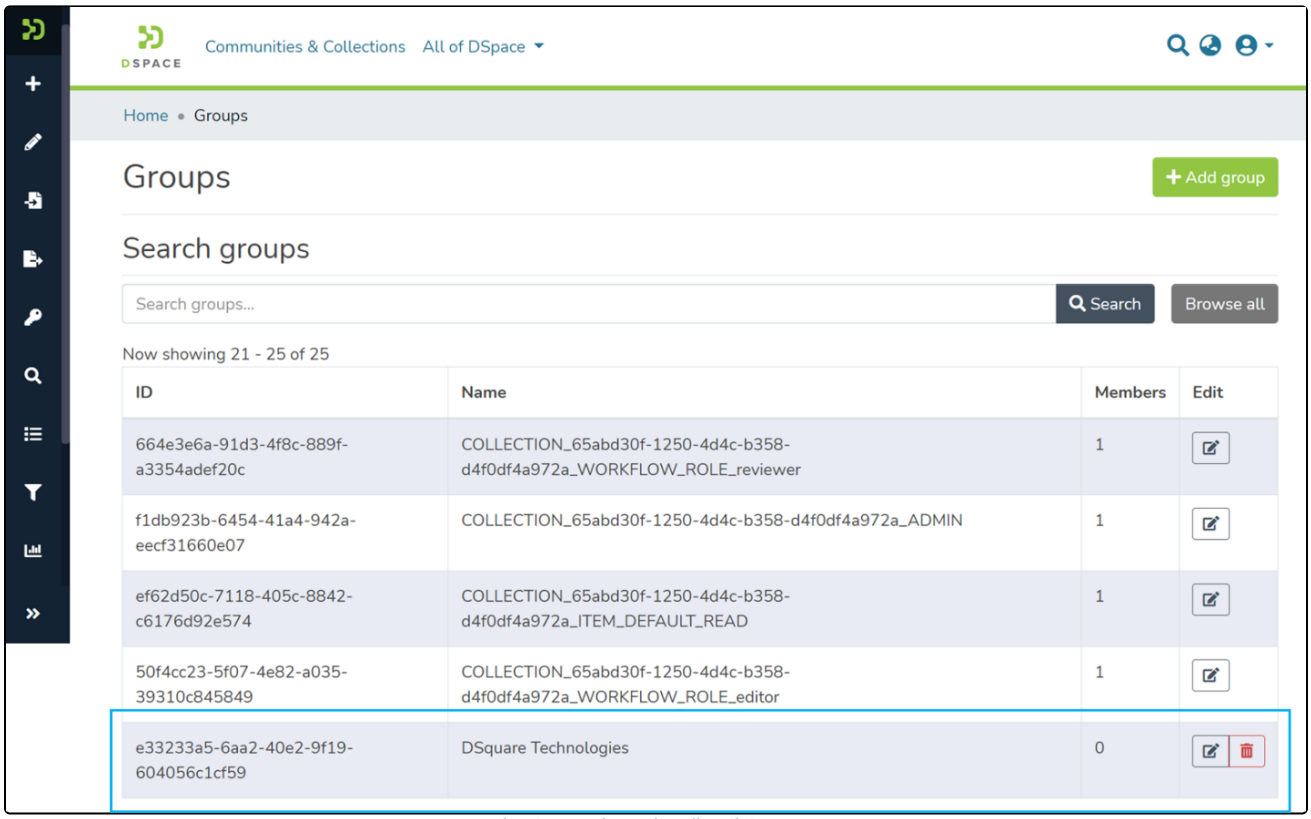
Add the Group Name and Description, then Save

Step 7: Upon successfully adding Group in DSpace, you will see a success prompt on the screen.



Group Saved confirmation banner

Step 8: The Group created will appear in the Group's list as highlighted below.



The Group shows in a list of Groups

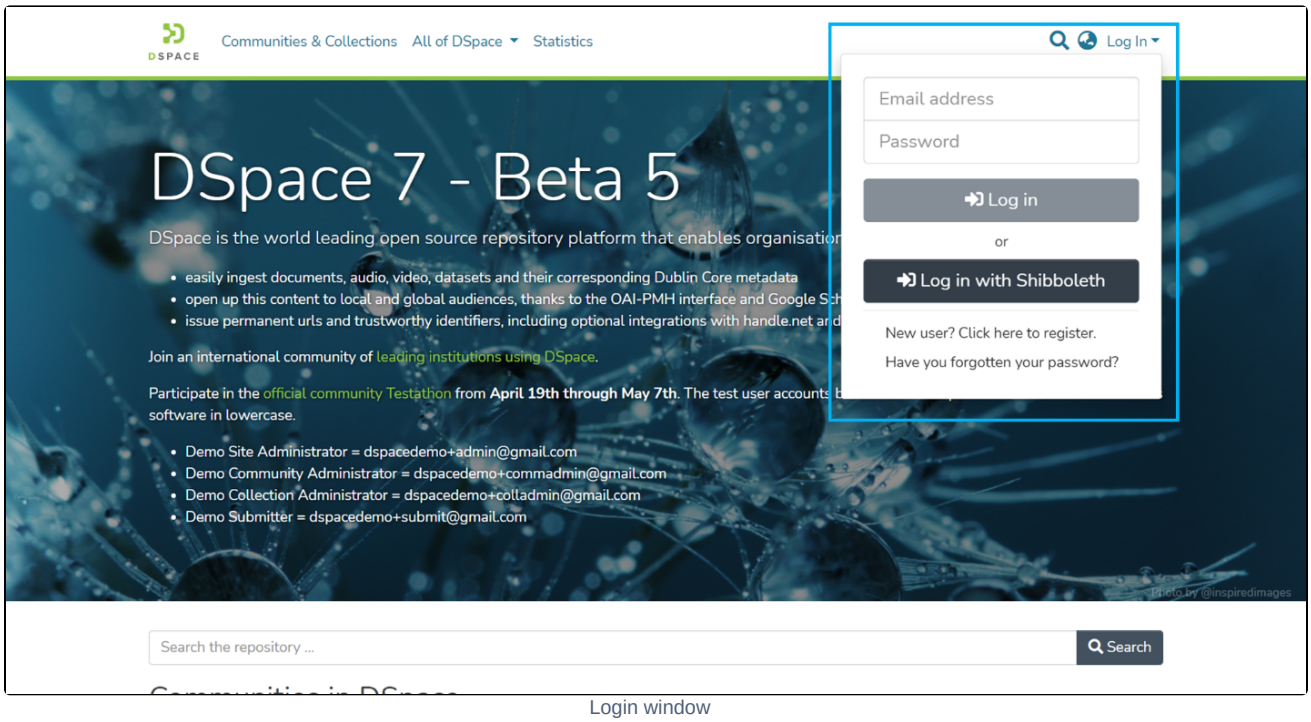
Manage a user group

The System, Community, and Collection administrator profiles can update user group(s). These users can perform the following activities:

- Update group details
- Delete group
- Add/remove E-people
- Add/Remove Subgroups

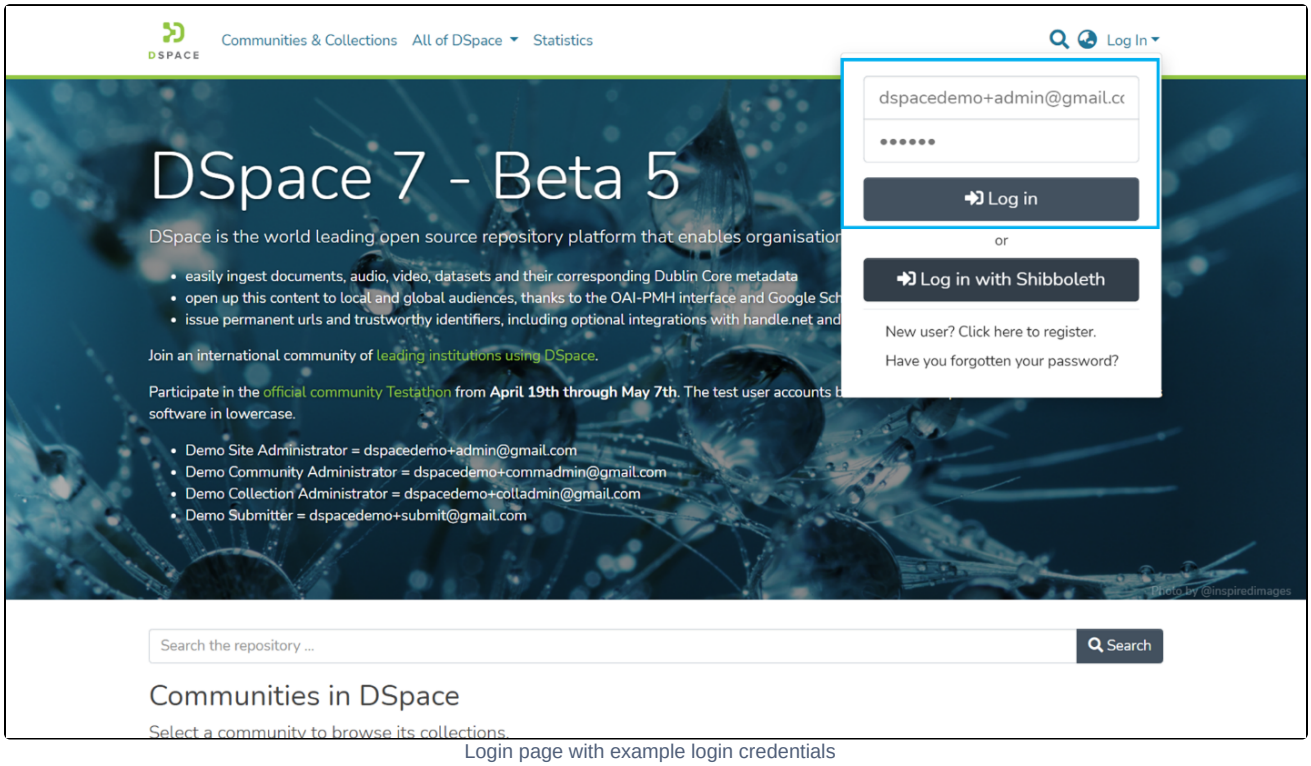
Update group details

Step 1: Click on the "Log In" link at the top right corner of the DSpace's homepage, and the pop-up will open, as illustrated below the screen.



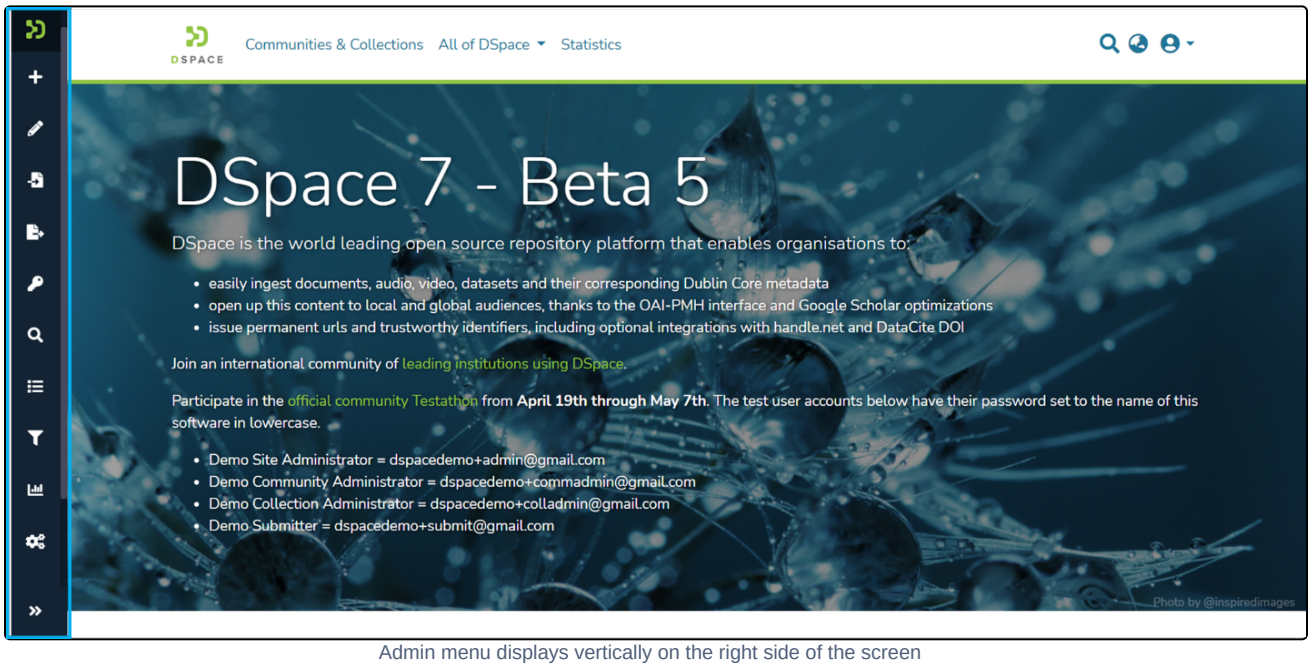
Login window

Step 2: Enter your user id and password and click on the "Log in" button to enter into DSpace.



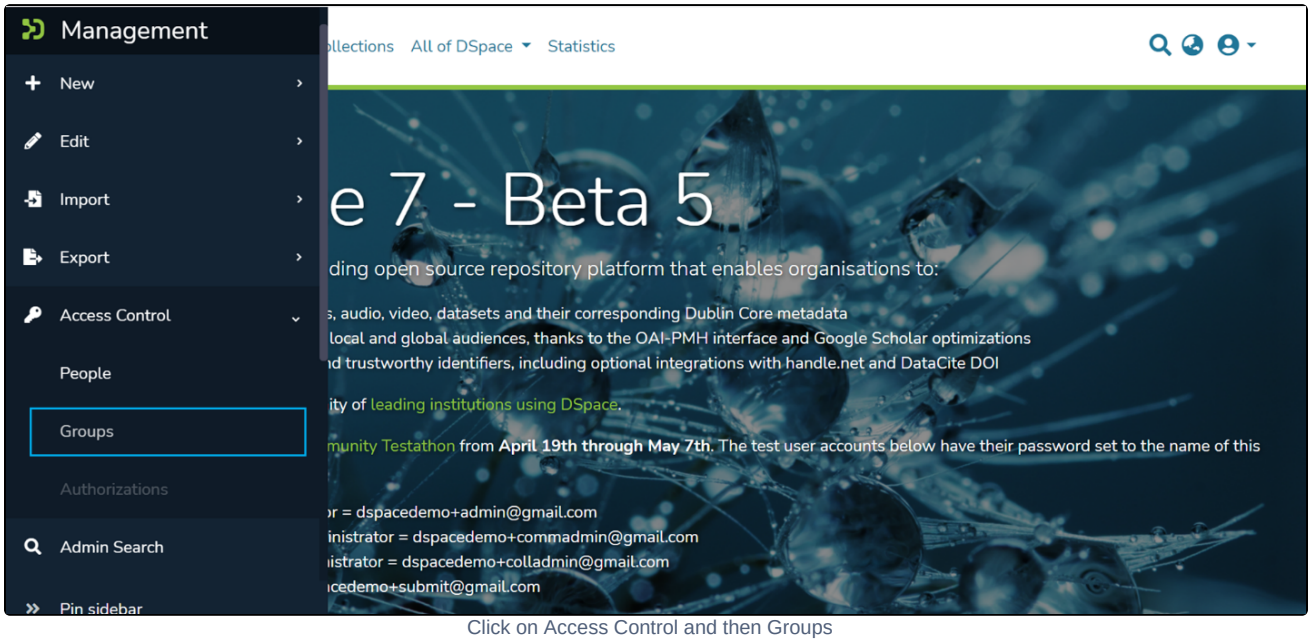
Login page with example login credentials

Step 3: Users with administrative rights will see the admin menu on the left-hand side of the screen, as shown in the below illustration.



Admin menu displays vertically on the right side of the screen

Step 4: Rollover your cursor over the administration menu, click on "Access Control," and click on the "Groups" link.

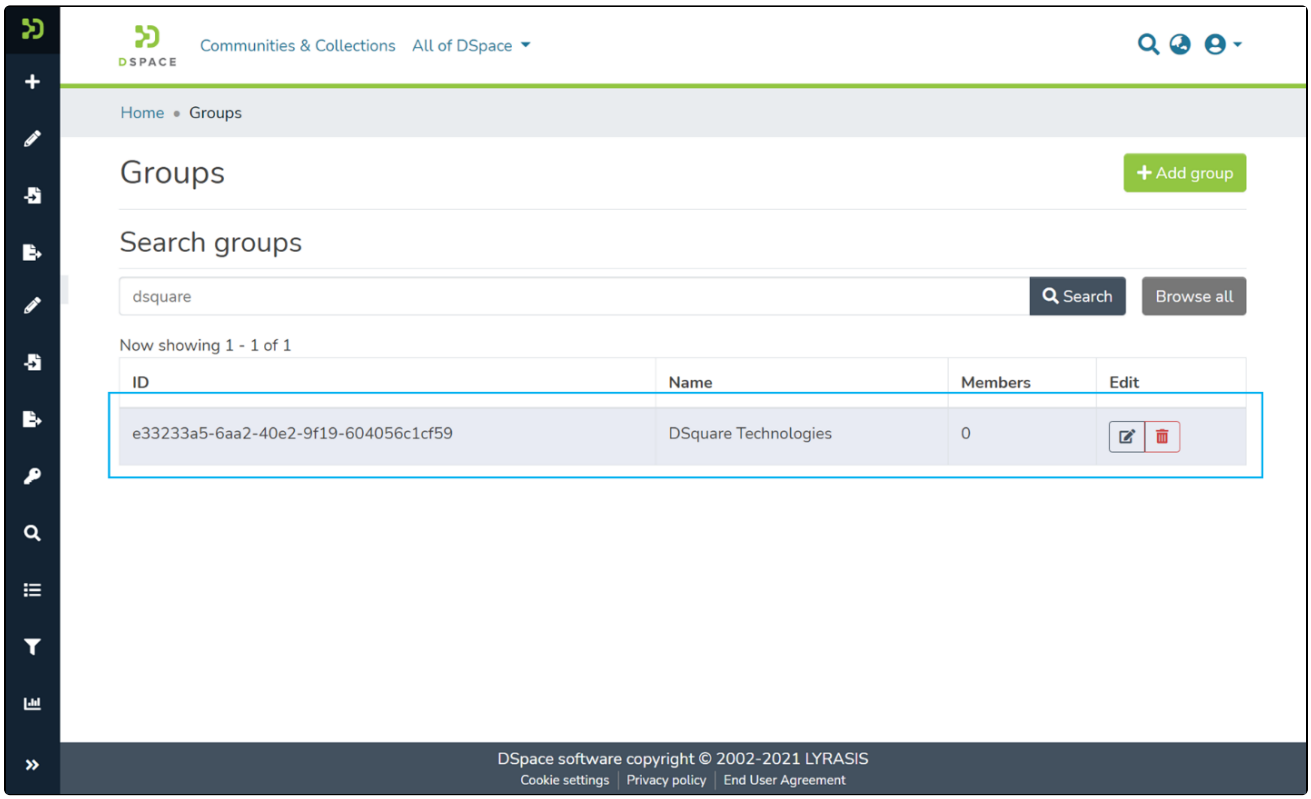


Click on Access Control and then Groups

Step 5: Users can find the Group required to be edited or deleted by scrolling down the Groups list or entering Group's metadata in the search field on the group page. Please see the illustration below showing an example.

The user group table shows the following details

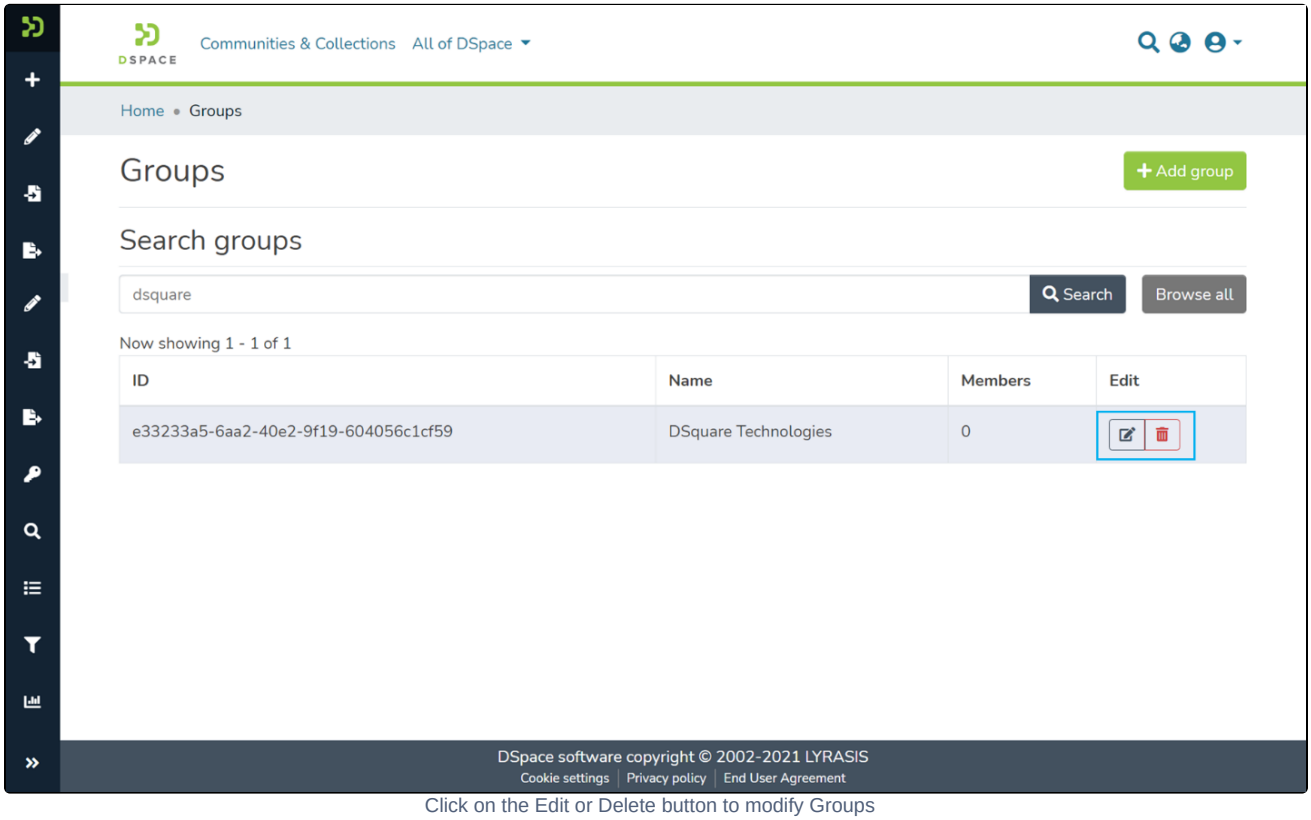
- **ID:** Unique ID generated in DSpace for each Group
- **Name:** Name of Group as entered during group creation
- **Members:** A count of Epeople were added to the Group as members.



Search Groups

Step 6: Click on the

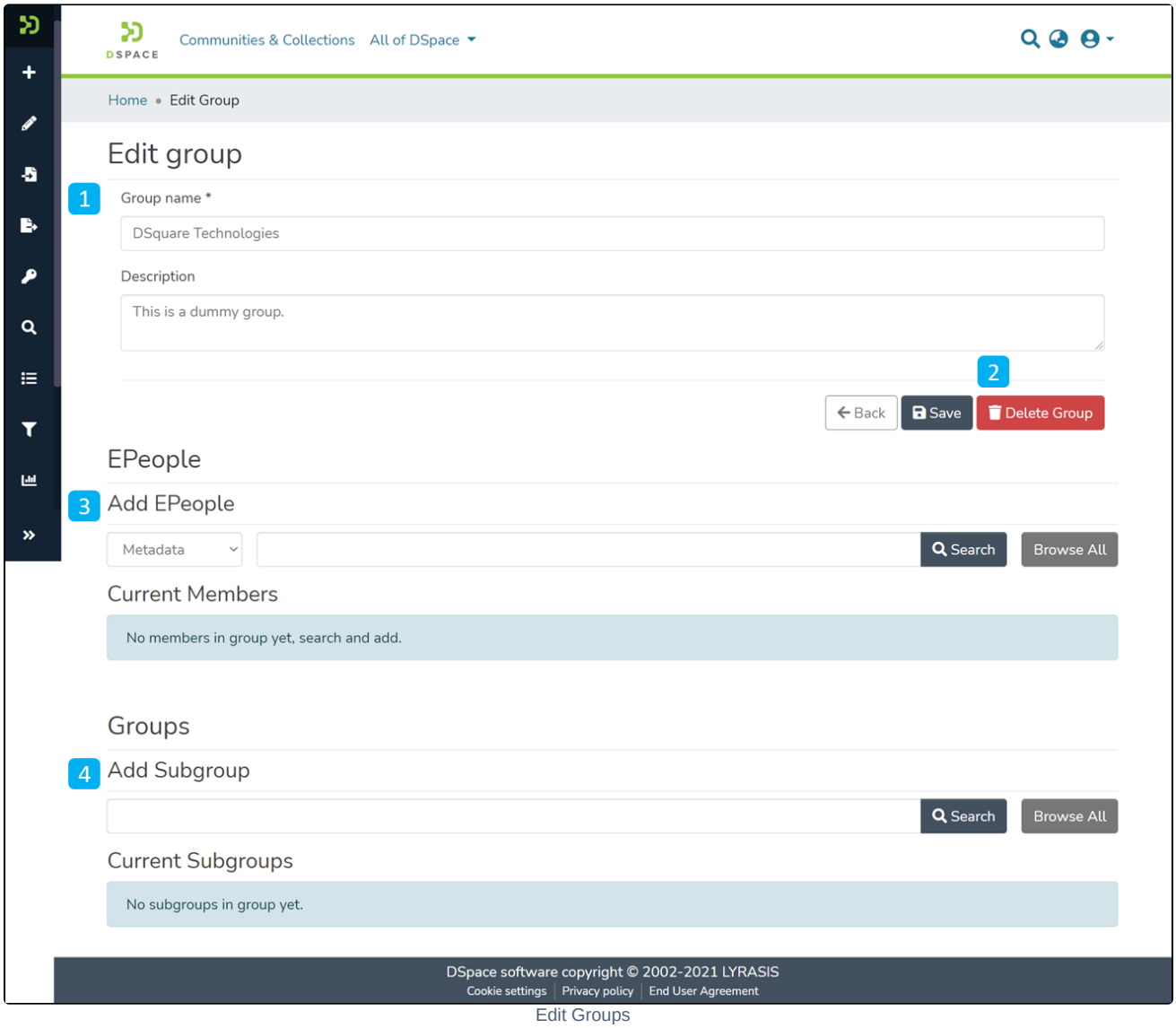
1. "Edit" button in the Edit column to edit the Group.
2. "Delete" button to delete the Group from DSpace permanently.



Click on the Edit or Delete button to modify Groups

Step 7: You can make the following updates on the Edit group page

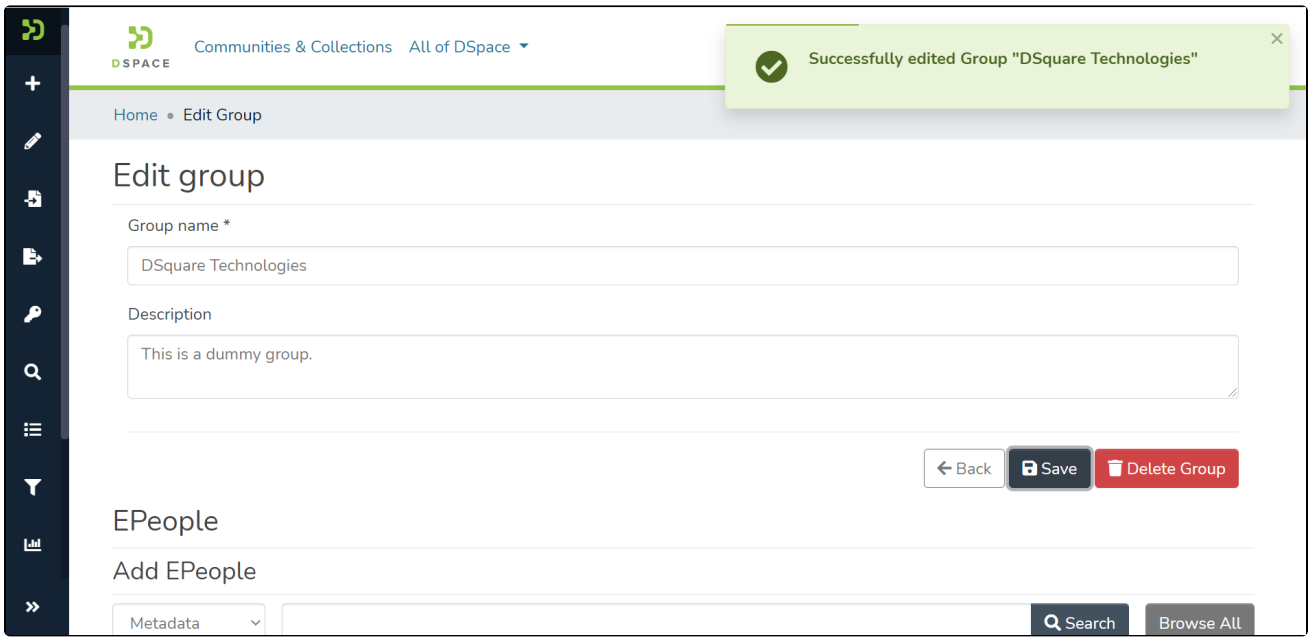
1. **Update group details:** Update the Group name and description.
2. **Delete Group:** Permanently delete the Group from DSpace.
3. **Add/Manage People:** Add Epeople to the Group using the search and browse function. Or delete existing group members using the delete option.
4. **Add/Manage subgroups:** Add existing groups as subgroups using the search and browse function. Or delete existing subgroups using the delete option.



Edit Groups

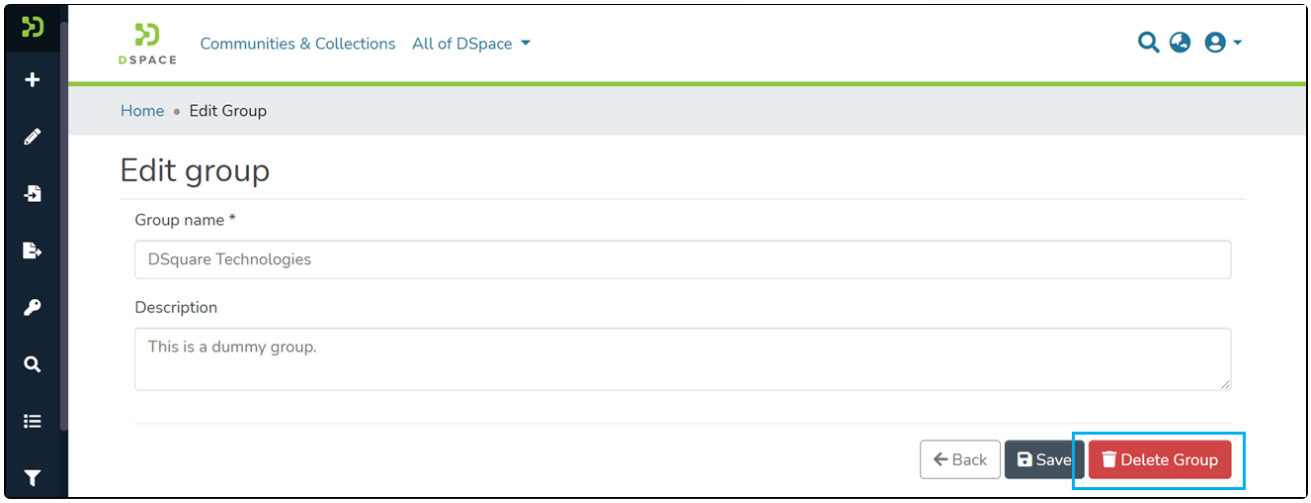
Step 8: Update the group name and description to update the group details and click on the "Save" button.

Step 9: A success prompt will appear, confirming a successful update as displayed below.



Delete Group

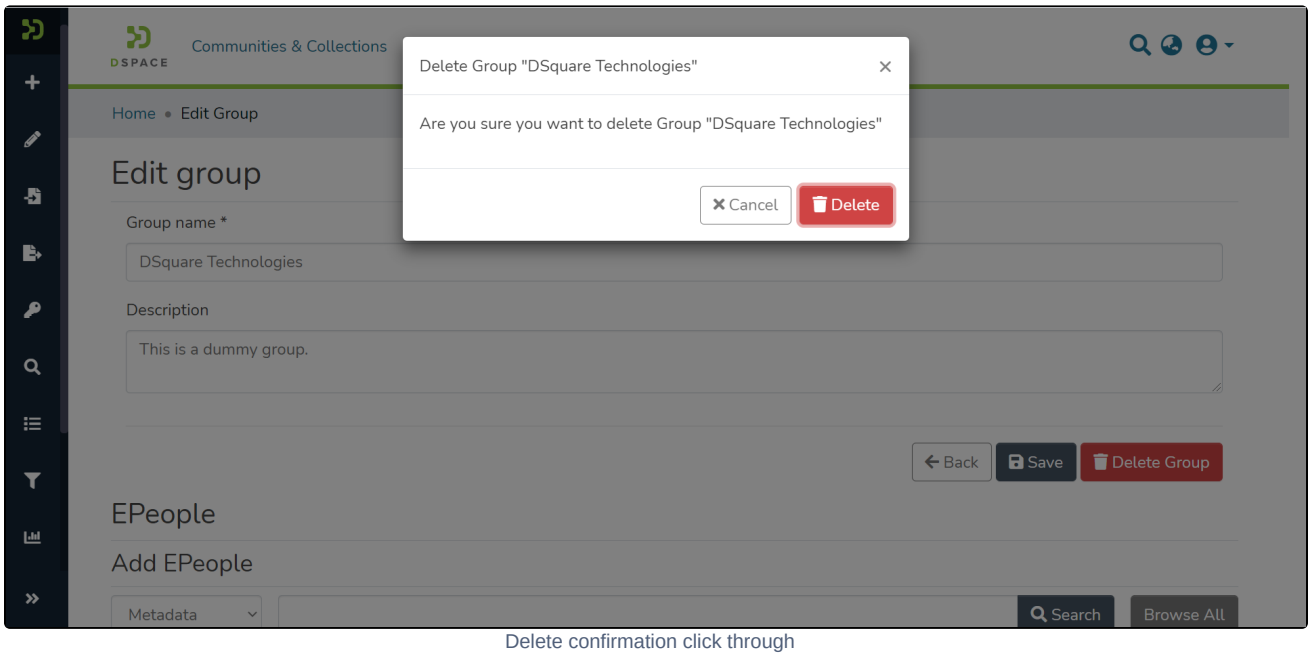
Step 10: Click on the "Delete Group" button to permanently delete the Group from DSpace.



Click on the red Delete button in. the bottom right corner to permanently delete a group

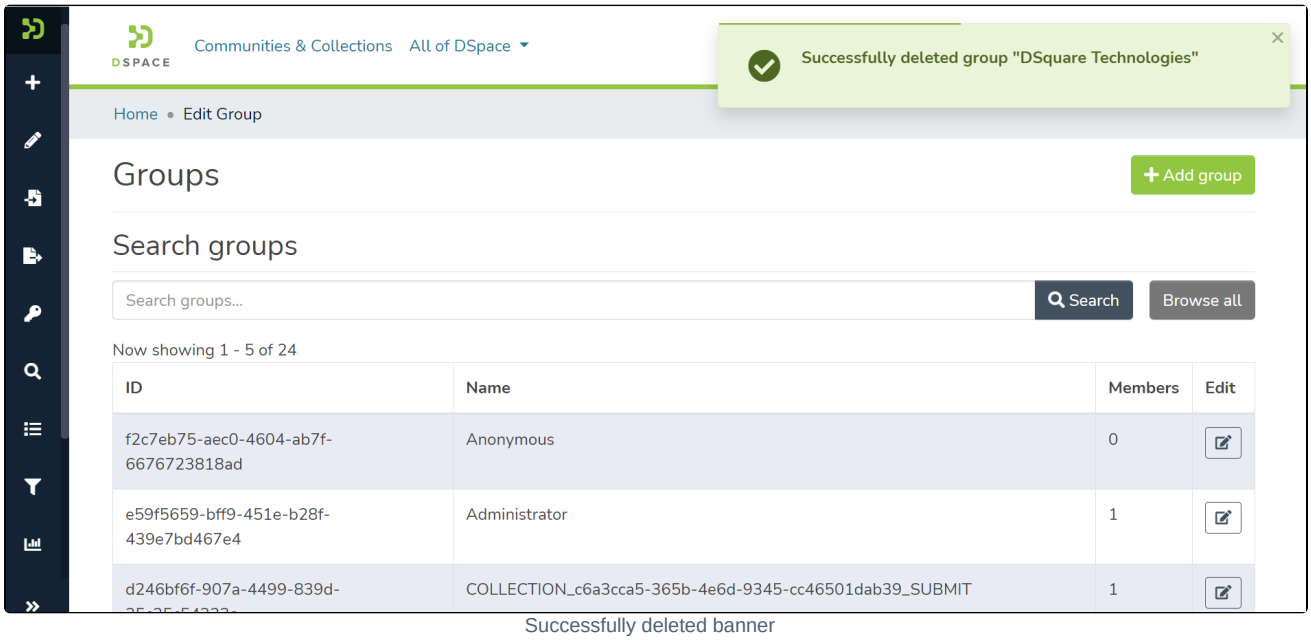
Step 11: DSpace will show you a confirmation prompt to re-assess your decision of deleting the Group. Click on "Delete" if you want to continue, or click "Cancel" to return.

Please note that the Group, once deleted, can not be recovered.



Delete confirmation click through

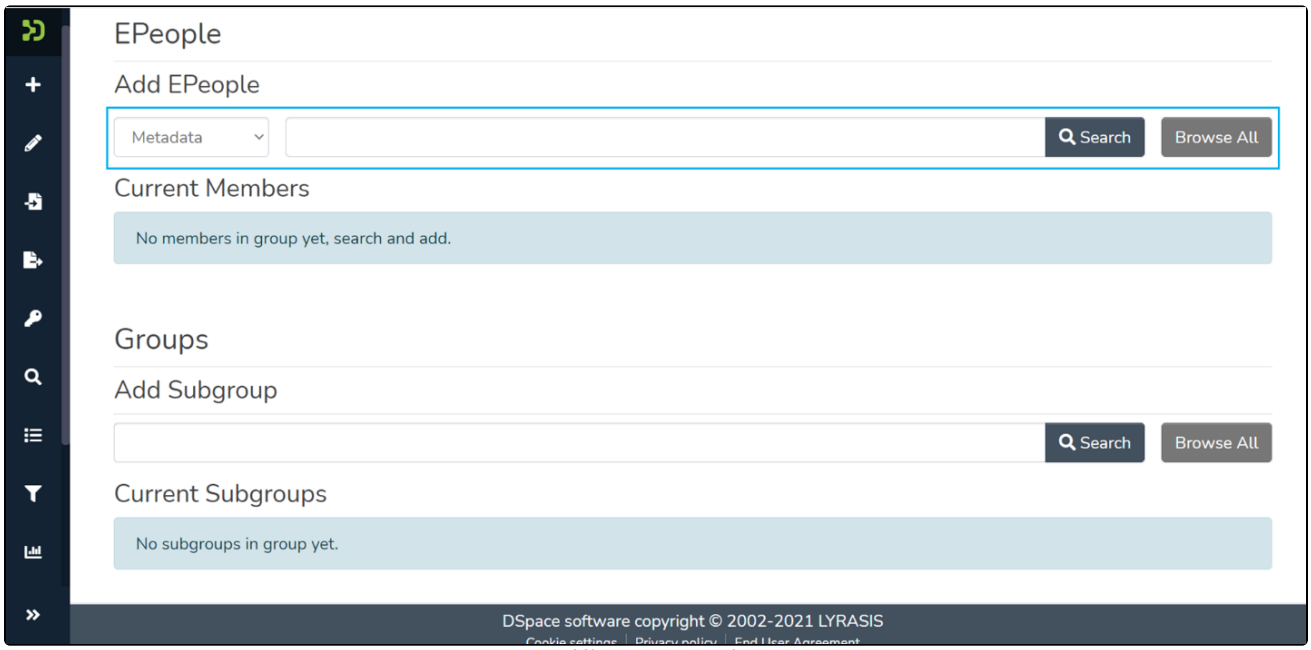
Step 12: Users will see a success prompt confirming the user group deletion, as demonstrated below.



Successfully deleted banner

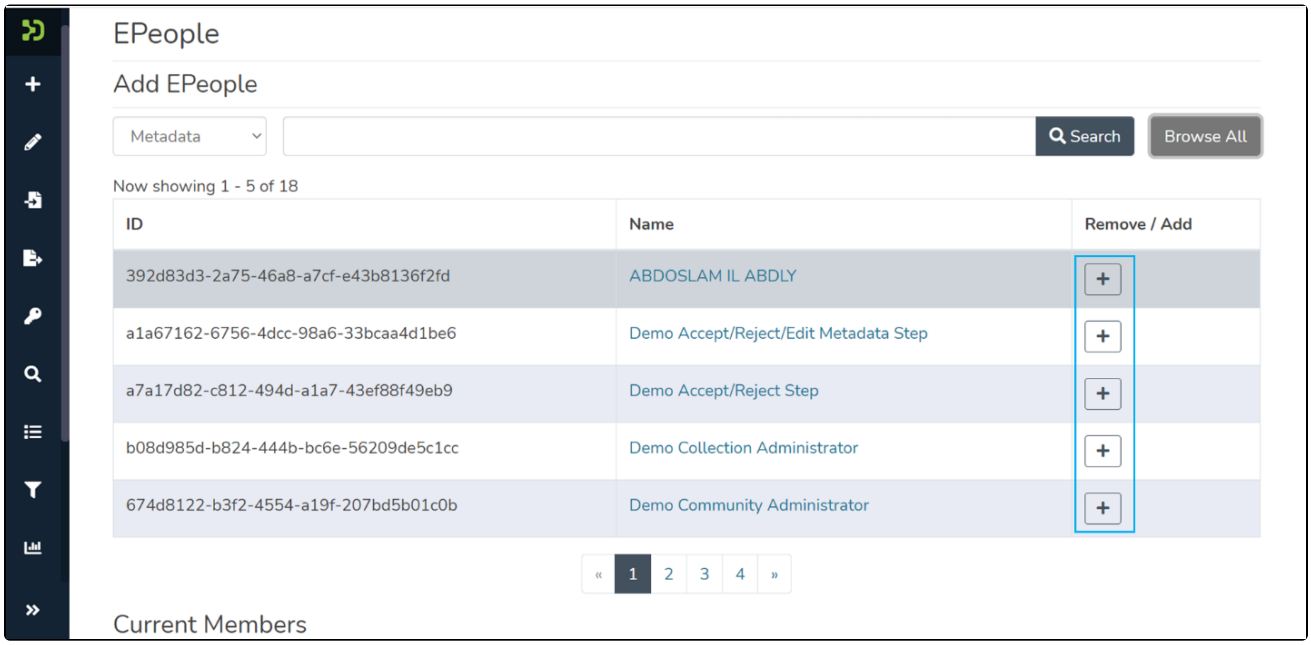
Add/Manage EPeople

Step 13: Scroll down to the Epeople section to find the user(s). Use metadata values to search Epeople and click the "Search" button, or click the "Browse All" button to list EPeople.

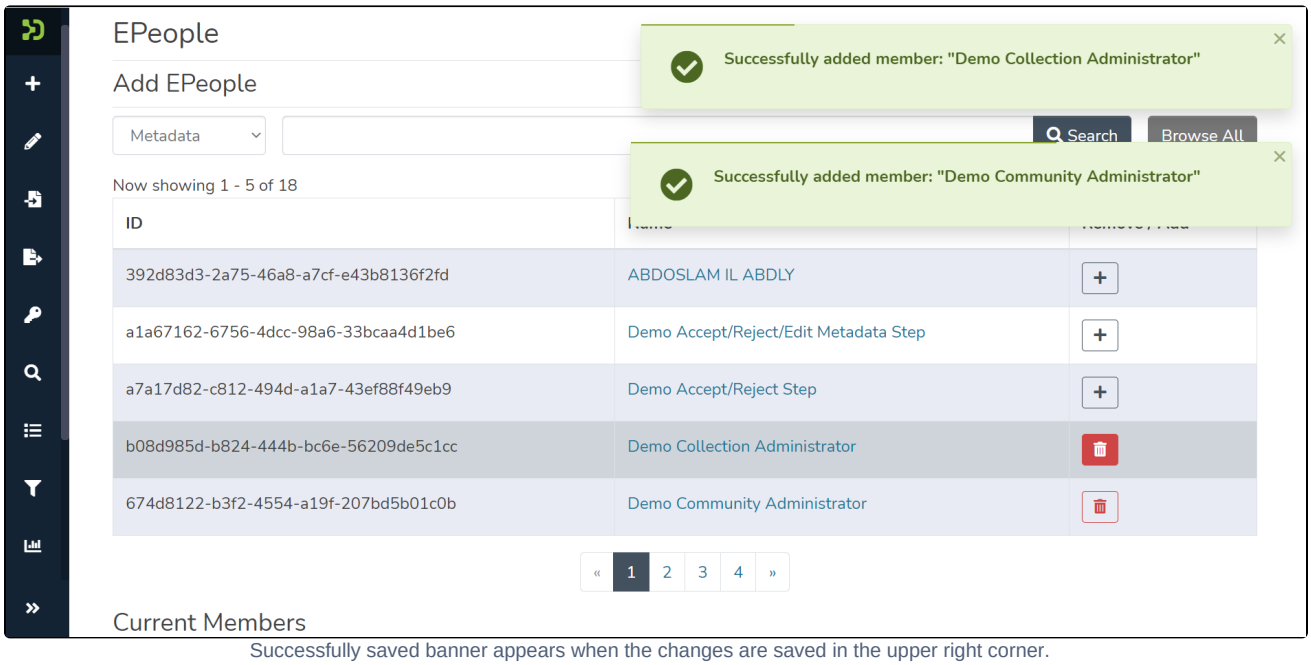


Add/Manage EPeople

Step 14: Click on the add button '+' next to each user name to add that user as a group member.

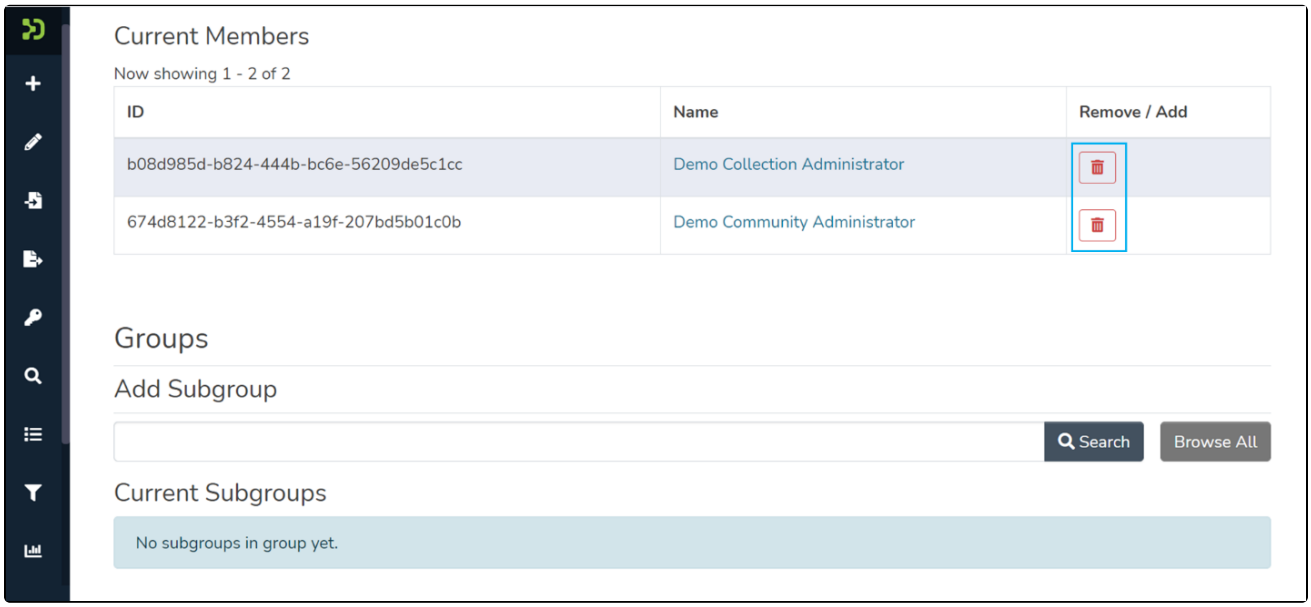


Step 15: A prompt will appear confirming the addition of users to the Group. As illustrated below, the "Delete" button will appear next to the user names.



Successfully saved banner appears when the changes are saved in the upper right corner.

Success prompt confirming user added as members

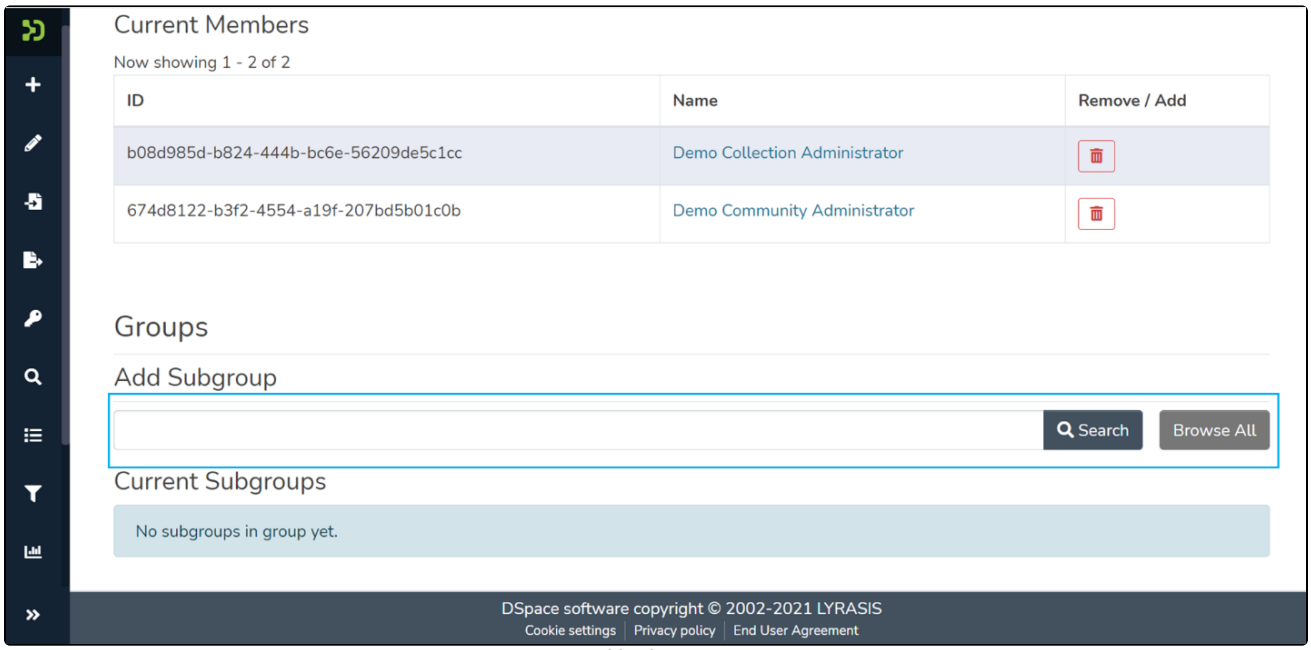


A list of current people

Users appearing as Members under the Current Members section

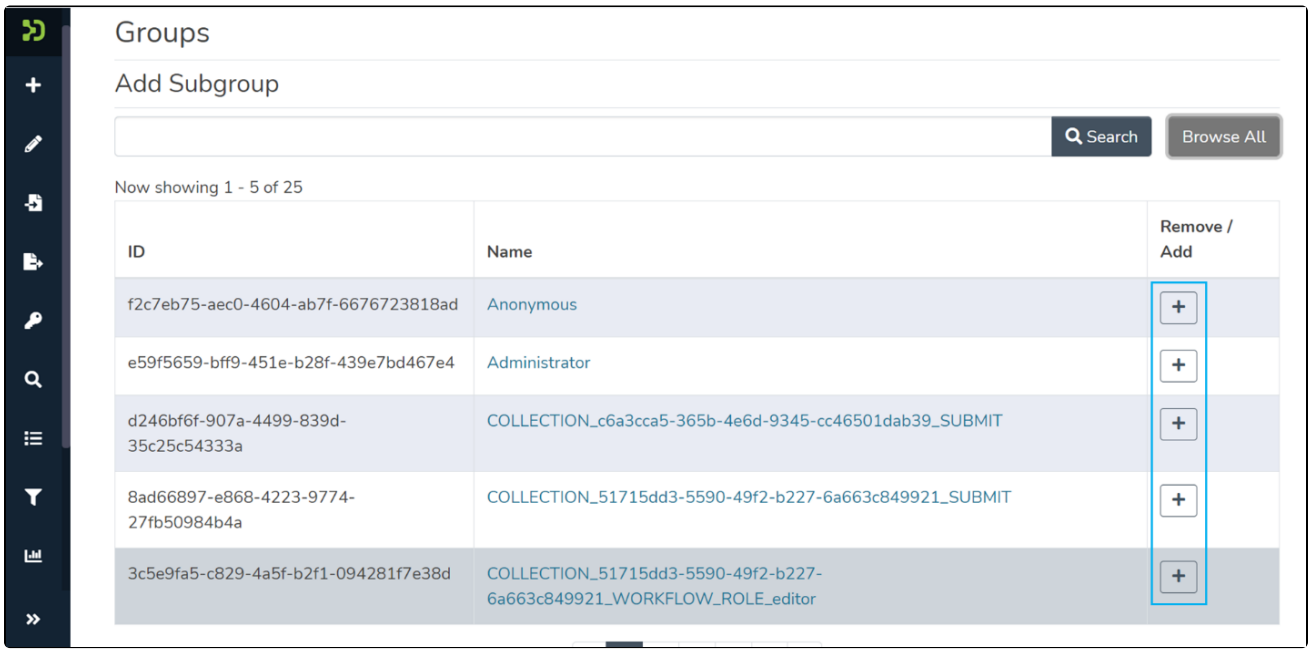
Add/Manage subgroups

Step 16: Scroll down to the Group section to find the Group (s). Enter a group name and click the "Search" button, or click the "Browse All" button to list groups.

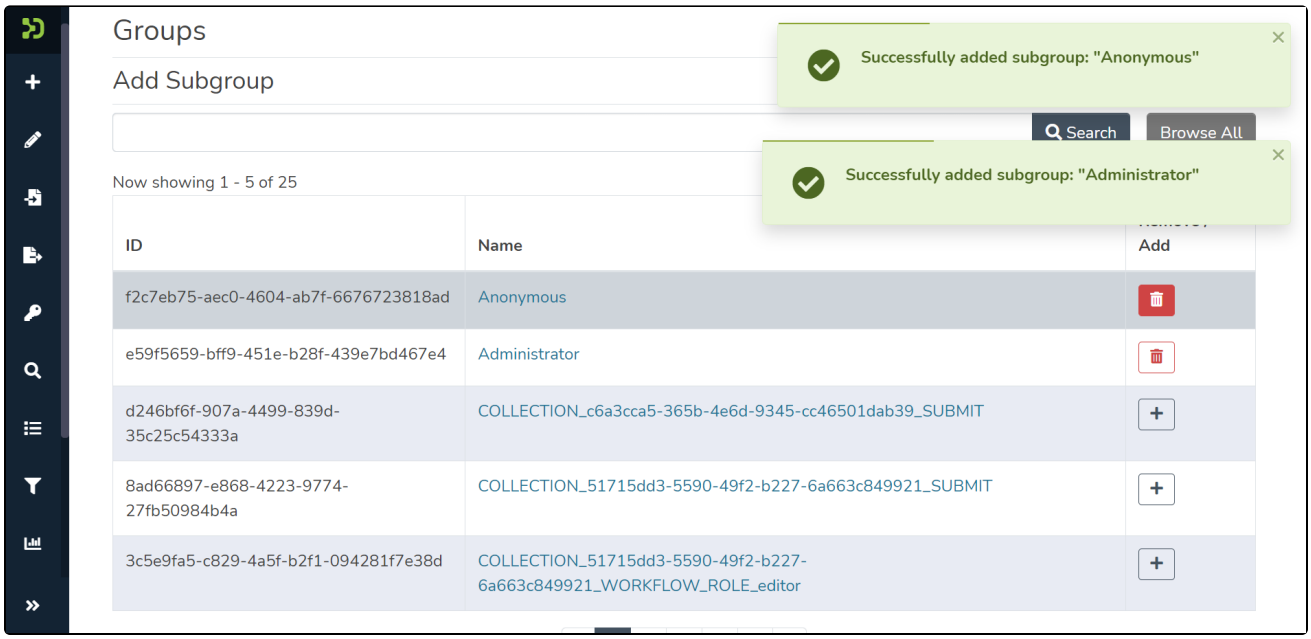


Add subgroup

Step 17: Click on the add button '+' next to each group name to add it as a sub-group.

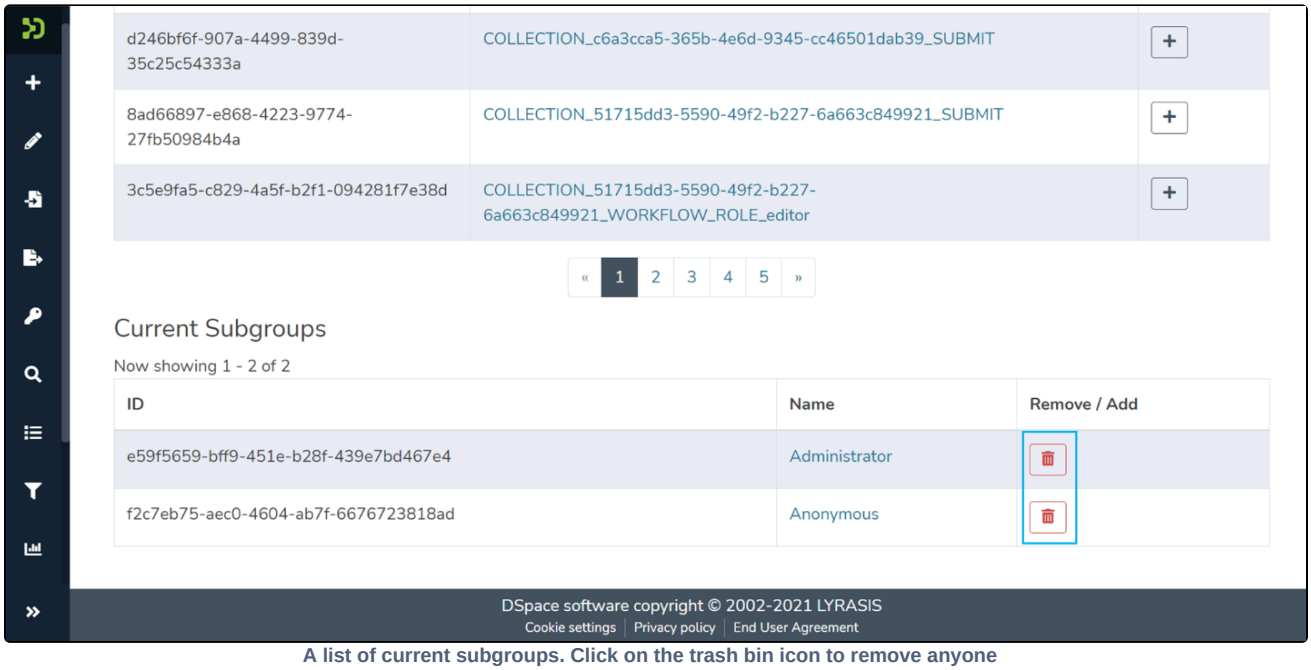


Step 18: A prompt will appear to confirm a subgroup in the Group. As illustrated below, the "Delete" button will appear next to the group names.



Successfully saved banner appears when the changes are saved in the upper right corner.

Success prompt confirming groups added as subgroups



A list of current subgroups. Click on the trash bin icon to remove anyone

Groups appearing as subgroups under the Current Subgroups section

System Administration

This top level node intends to hold all system administration aspects of DSpace including but not limited to:

- Installation
- Upgrading
- Troubleshooting system errors
- Managing Dependencies

In this context System administration is defined as all technical tasks required to get DSpace in a state in which it operates properly so its behaviour is predictable and can be used according to all the guidelines under "Using DSpace".

Introduction to DSpace System Administration

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

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

Command used:	*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

Many/most commands and scripts have a simple `[dspace]/bin/dspace <command>` command. See the Application Layer chapter for the details of the DSpace Command Launcher, and the Command Line Operations guide for common commands.

- AIP Backup and Restore
  - DSpace AIP Format
- Ant targets and options
- Command Line Operations
  - Executing streams of commands
  - Database Utilities
  - Miscellaneous Commands
- Handle.Net Registry Support
- Logical Item Filtering and DOI Filtered Provider for DSpace
- Mediafilters for Transforming DSpace Content
  - ImageMagick Media Filters
- Performance Tuning DSpace
- Ping or Healthcheck endpoints for confirming DSpace services are functional
- Proxy Configuration
- Scheduled Tasks via Cron
- Search Engine Optimization
  - Google Scholar Metadata Mappings
- Troubleshooting Information
- Validating CheckSums of Bitstreams

AIP Backup and Restore

1 Background & Overview
- 1.1 How does this differ from traditional DSpace Backups? Which Backup route is better?
- 1.2 How does this help backup your DSpace to remote storage or cloud services (like DuraCloud)?
- 1.3 AIPs are Archival Information Packages
- 1.4 AIP Structure / Format

2 Running the Code
- 2.1 Exporting AIPs
  - 2.1.1 Export Modes & Options
  - 2.1.2 Exporting just a single AIP
  - 2.1.3 Exporting AIP Hierarchy
    - 2.1.3.1 Exporting Entire Site
- 2.2 Ingesting / Restoring AIPs
  - 2.2.1 Ingestion Modes & Options
    - 2.2.1.1 The difference between "Submit" and "Restore/Replace" modes
  - 2.2.2 Submitting AIP(s) to create a new object
    - 2.2.2.1 Submitting a Single AIP
    - 2.2.2.2 Submitting an AIP Hierarchy
    - 2.2.2.3 Submitting AIP(s) while skipping any Collection Approval Workflows
  - 2.2.3 Restoring/Replacing using AIP(s)
    - 2.2.3.1 Default Restore Mode
    - 2.2.3.2 Restore, Keep Existing Mode
    - 2.2.3.3 Force Replace Mode
    - 2.2.3.4 Restoring Entire Site
- 2.3 Cleaning up from a failed import
- 2.4 Performance considerations
- 2.5 Disable User Interaction for Cron

3 Command Line Reference
- 3.1 Additional Packager Options
  - 3.1.1 How to use additional options

4 Configuration in 'dspace.cfg'
- 4.1 AIP Metadata Dissemination Configurations
- 4.2 AIP Ingestion Metadata Crosswalk Configurations
- 4.3 AIP Ingestion EPerson Configurations
- 4.4 AIP Configurations To Improve Ingestion Speed while Validating

5 Common Issues or Error Messages

Background & Overview

AIP Backup & Restore doesn't yet work for Configurable Entities

Configurable Entities are not fully supported by AIP Backup & Restore. Since Entities are Items, their metadata and files can be exported/imported via AIPs. However, their relationships to other Entities cannot yet be exported (or imported) via AIPs. Therefore, **restoring Entities via AIP Backup & Restore may result in accidental data loss** (namely loss of relationships). See https://github.com/DSpace/DSpace/issues/2882 for more information.

AIP Backup & Restore functionality only works with the Latest Version of Items

If you are using the Item Level Versioning functionality (disabled by default), you must be aware that this "Item Level Versioning" feature is **not yet compatible** with AIP Backup & Restore. **Using them together may result in accidental data loss.** Currently the AIPs that DSpace generates only store the *latest version* of an Item. Therefore, past versions of Items will always be lost when you perform a restore / replace using AIP tools.

Additional background information available in the Open Repositories 2010 Presentation entitled Improving DSpace Backups, Restores & Migrations

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

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

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

Benefits for the DSpace community:

- Allows one to more easily move entire Communities or Collections between DSpace instances.
- Allows for a potentially more consistent backup of this hierarchy (e.g. to DuraCloud, or just to your own local backup system), rather than relying on synchronizing a backup of your Database (stores metadata/relationships) and assetstore (stores files/bitstreams).
- Provides a way for people to more easily get their data out of DSpace (whatever the purpose may be).
- Provides a relatively standard format for people to migrate entire hierarchies (Communities/Collections) from one DSpace to another (or from another system into DSpace).

How does this differ from traditional DSpace Backups? Which Backup route is better?

Traditionally, it has always been recommended to backup and restore DSpace's database and files (also known as the "assetstore") separately. This is described in more detail in the Storage Layer section of the DSpace System Documentation. The traditional backup and restore route is still a recommended and supported option.

However, the new AIP Backup & Restore option seeks to try and resolve many of the complexities of a traditional backup and restore. The below table details some of the differences between these two valid Backup and Restore options.

	Traditional Backup & Restore (Database and Files)	**AIP Backup & Restore**
Supported Backup /Restore Types		
Can Backup & Restore all DSpace Content easily	Yes (Requires two backups/restores – one for Database and one for Files)	Yes (Though, will not backup/restore items which are not officially "in archive")
Can Backup & Restore a Single Community /Collection/Item easily	No (It is possible, but requires a strong understanding of DSpace database structure & folder organization in order to only backup & restore metadata/files belonging to that single object)	Yes
Backups can be used to move one or more Community/Collection /Items to another DSpace system easily.	No (Again, it is possible, but requires a strong understanding of DSpace database structure & folder organization in order to only move metadata/files belonging to that object)	Yes
Can Backup & Restore Item Versions	Yes (Requires two backups/restores – one for Database and one for Files)	**No** (Currently, AIP Backup & Restore is not fully compatible with Item Level Versioning. AIP Backup & Restore can only backup/restore the *latest version* of an Item)
Can Backup & Restore Configurable Entities	Yes (Requires two backups/restores – one for Database and one for Files)	**No** (Currently, AIP Backup & Restore is not fully compatible with Configurable Entities. AIP Backup & Restore can only backup/restore the metadata & files of the Entity, but cannot backup/restore relationships to other Entities)
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), if it tries to restore an Item Mapping to a Collection that it hasn't yet restored. But this error can be safely bypassed using the 'skipIfParentMissing' flag (see Additional Packager Options for more details).
Supports backup/restore of all in-process, uncompleted Submissions (or those currently in an approval workflow)	Yes	No (AIPs are only generated for objects which are completed and considered "in archive")
Supports backup/restore of Items using custom Metadata Schemas & Fields	Yes	Yes (Custom Metadata Fields will be automatically recreated. Custom Metadata Schemas must be manually created first, in order for DSpace to be able to recreate custom fields belonging to that schema. See Common Issues or Error Messages for more details.)

Supports backup/restore of all local DSpace Configurations and Customizations	Yes (if you backup your **entire** DSpace directory as part of backing up your files)	Not by default (unless you also backup parts of your DSpace directory – note, you wouldn't need to backup the '[dspace]/assetstore' folder again, as those files are already included in AIPs)

Based on your local institutions needs, you will want to choose the backup & restore process which is most appropriate to you. You may also find it beneficial to use both types of backups on different time schedules, in order to keep to a minimum the likelihood of losing your DSpace installation settings or its contents. For example, you may choose to perform a Traditional Backup once per week (to backup your local system configurations and customizations) and an AIP Backup on a daily basis. Alternatively, you may choose to perform daily Traditional Backups and only use the AIP Backup as a "permanent archives" option (perhaps performed on a weekly or monthly basis).

Don't Forget to Backup your Configurations and Customizations

If you choose to use the AIP Backup and Restore option, do not forget to also backup your local DSpace configurations and customizations. Depending on how you manage your own local DSpace, these configurations and customizations are likely in one or more of the following locations:

- `[dspace]` - The DSpace installation directory (Please note, if you also use the AIP Backup & Restore option, you do **not** need to backup your `[dspace]/assetstore` directory, as those files already exist in your AIPs).
- `[dspace-source]` - The DSpace source directory

How does this help backup your DSpace to remote storage or cloud services (like DuraCloud)?

While AIP Backup and Restore is primarily a way to export your DSpace content objects to a local filesystem (or mounted drive), it can also be used as the basis for ensuring your content is safely backed up in a remote location (e.g. DuraCloud or other cloud backup services).

Simply put, these AIPs can be generated and then replicated off to remote storage or a cloud backup service for safe keeping. You can then pull them down either as an entire set, or individually, in order to restore one or more objects into your DSpace instance. While you could simply backup your entire DSpace database and "assetstore" to a cloud service, you'd have to download the **entire** database backup again in order to restore any content. With AIPs, you can instead just download the individual AIP files you need (which can decrease your I/O costs, if any exist) for that restoration.

This upload/download of your AIPs to a backup location can be managed in a manual fashion (e.g. via your own custom code or shell scripts), or you can use a DSpace Replication Task Suite add-on to help ease this process

The Replication Task Suite add-on for DSpace allows you the ability to backup and restore DSpace contents to /from AIPs via the DSpace Administrative Web Interface. It also includes "connectors" to the DuraCloud API, so you can configure it to automatically backup/retrieve your AIPs to/from DuraCloud. Installing this add-on means you can now easily backup and restore DSpace to DuraCloud (or other systems) simply via the DSpace Administrative Web Interface. More information on installing and configuring this add-on can be found on the Replication Task Suite page.

Makeup and Definition of AIPs

AIPs are Archival Information Packages

- AIP is a package describing **one archival object** in DSpace.
  - The **archival object** may be a single **Item**, **Collection**, **Community**, or **Site** (Site AIPs contain site-wide information). Bitstreams are included in an Item's AIP.
  - Each AIP is logically self-contained, can be restored without rest of the archive. (So you could restore a single Item, Collection or Community)
  - Collection or Community AIPs do **not** include all child objects (e.g. Items in those Collections or Communities), as each AIP only describes **one** object. However, these container AIPs do contain references (links) to all child objects. These references can be used by DSpace to automatically restore all referenced AIPs when restoring a Collection or Community.
  - AIPs are only generated for objects which are currently in the "in archive" state in DSpace. This means that in-progress, uncompleted submissions are not described in AIPs and cannot be restored after a disaster. Permanently removed objects will also no longer be exported as AIPs after their removal. However, withdrawn objects will continue to be exported as AIPs, since they are still considered under the "in archive" status.
  - AIPs with identical contents will always have identical checksums. This provides a basic means of validating whether the contents within an AIP have changed. For example, if a Collection's AIP has the same checksum at two different points in time, it means that Collection has not changed during that time period.
  - AIP profile favors completeness and accuracy rather than presenting the semantics of an object in a standard format. It conforms to the quirks of DSpace's internal object model rather than attempting to produce a universally understandable representation of the object. When possible, an AIP tries to use common standards to express objects.
  - An AIP *can* serve as a DIP (Dissemination Information Package) or SIP (Submission Information Package), especially when transferring custody of objects to another DSpace implementation.
  - In contrast to SIP or DIP, the AIP should include all available DSpace structural and administrative metadata, and basic provenance information. AIPs also describe some basic system level information (e.g. Groups and People).

AIP Structure / Format

Generally speaking, an AIP is an Zip file containing a METS manifest and all related content bitstreams.

For more specific details of AIP format / structure, along with examples, please see DSpace AIP Format.

Running the Code

Exporting AIPs

Export Modes & Options

All AIP Exports are done by using the Dissemination Mode (`-d` option) of the `packager` command.

There are two types of AIP Dissemination you can perform:

- **Single AIP** (default, using `-d` option) - Exports just an AIP describing a single DSpace object. So, if you ran it in this default mode for a Collection, you'd just end up with a single Collection AIP (which would not include AIPs for all its child Items)
- **Hierarchy of AIPs** (using the `-d --all` or `-d -a`option) - Exports the requested AIP describing an object, plus the AIP for all child objects. Some examples follow:
  - For a Site - this would export **all** Communities, Collections & Items within the site into AIP files (in a provided directory)
  - For a Community - this would export that Community and all SubCommunities, Collections and Items into AIP files (in a provided directory)
  - For a Collection - this would export that Collection and all contained Items into AIP files (in a provided directory)
  - For an Item – this just exports the Item into an AIP as normal (as it already contains its Bitstreams/Bundles by default)

Exporting just a single AIP

To export in single AIP mode (default), use this "packager" command template:

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

for example:

```
[dspace]/bin/dspace packager -d -t AIP -e admin@myu.edu -i 4321/4567 aip4567.zip
```

The above code will export the object of the given handle (4321/4567) into an AIP file named "aip4567.zip". This will **not** include any child objects for Communities or Collections.

Exporting AIP Hierarchy

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

For example, use this 'packager' command template:

```
[dspace]/bin/dspace packager -d -a -t AIP -e <eperson> -i <handle> <file-path>
```

for example:

```
[dspace]/bin/dspace packager -d -a -t AIP -e admin@myu.edu -i 4321/4567 aip4567.zip
```

The above code will export the object of the given handle (4321/4567) into an AIP file named "aip4567.zip". In addition it would export all children objects to the same directory as the "aip4567.zip" file. The child AIP files are all named using the following format:

- File Name Format: `<Obj-Type>@<Handle-with-dashes>.zip`
  - e.g. COMMUNITY@123456789-1.zip, COLLECTION@123456789-2.zip, ITEM@123456789-200.zip
  - This general file naming convention ensures that you can easily locate an object to restore by its name (assuming you know its Object Type and Handle).
- Alternatively, if object doesn't have a Handle, it uses this File Name Format: `<Obj-Type>@internal-id-<DSpace-ID>.zip` (e.g. ITEM@internal-id-234.zip)

AIPs are only generated for objects which are currently in the "in archive" state in DSpace. This means that in-progress, uncompleted submissions are not described in AIPs and cannot be restored after a disaster.

Exporting Entire Site

To export an entire DSpace Site, pass the packager the Handle `<site-handle-prefix>/0`. For example, if your site prefix is "4321", you'd run a command similar to the following:

```
[dspace]/bin/dspace packager -d -a -t AIP -e admin@myu.edu -i 4321/0 sitewide-aip.zip
```

Again, this would export the DSpace Site AIP into the file "sitewide-aip.zip", and export AIPs for **all** Communities, Collections and Items into the same directory as the Site AIP.

Ingesting / Restoring AIPs

Ingestion Modes & Options

Ingestion of AIPs is a bit more complex than Dissemination, as there are several different "modes" available:

1. Submit/Ingest Mode (`-s` option, default) – submit AIP(s) to DSpace in order to create a new object(s) (i.e. AIP is treated like a SIP – Submission Information Package)
2. Restore Mode (`-r` option) – restore pre-existing object(s) in DSpace based on AIP(s). This also attempts to restore all handles and relationships (parent/child objects). This is a specialized type of "submit", where the object is created with a known Handle, known UUID and known relationships.
3. Replace Mode (`-r -f` option) – replace existing object(s) in DSpace based on AIP(s). This also attempts to restore all handles and relationships (parent/child objects). This is a specialized type of "restore" where the contents of existing object(s) is replaced by the contents in the AIP(s). By default, if a normal "restore" finds the object already exists, it will back out (i.e. rollback all changes) and report which object already exists.

Again, like export, there are two types of AIP Ingestion you can perform (using any of the above modes):

- **Single AIP** (default) - Ingests just an AIP describing a single DSpace object. So, if you ran it in this default mode for a Collection AIP, you'd just create a DSpace Collection from the AIP (but not ingest any of its child objects)
- **Hierarchy of AIPs** (by including the `--all` or `-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** (`-s` mode) - creates a new object (AIP is treated like a SIP)
  - By default, a new Handle is always assigned
    - However, you can force it to use the handle specified in the AIP by specifying `-o ignoreHandle=false` as one of your parameters
  - By default, a new Parent object **must** be specified (using the `-p` parameter). This is the location where the new object will be created.
    - However, you can force it to use the parent object specified in the AIP by specifying `-o ignoreParent=false` as one of your parameters
  - By default, will respect a Collection's Workflow process when you submit an Item to a Collection
    - However, you can specifically *skip* any workflow approval processes by specifying `-w` parameter.
  - **Always** adds a new Deposit License to Items
  - **Always** adds new DSpace System metadata to Items (includes new "dc.date.accessioned", "dc.date.available", "dc.date.issued" and "dc.description.provenance" entries)
  - **WARNING:** Submission mode may not be able to maintain Item Mappings between Collections. Because these mappings are recorded via the Collection Handles, mappings may be restored improperly if the Collection handle has changed when moving content from one DSpace instance to another.

- **Restore / Replace Mode** (`-r` mode) - restores a previously existing object (as if from a backup)
  - By default, the Handle specified in the AIP is restored
    - However, for restores, you can force a new handle to be generated by specifying `-o ignoreHandle=true` as one of your parameters. (NOTE: Doesn't work for *replace* mode as the new object always retains the handle of the replaced object)
    - Restore/Replace restores Handles as well as UUIDs. (*NOTE: UUID restoration only possible in 7.1 or above*)
  - By default, the object is restored under the Parent specified in the AIP
    - However, for restores, you can force it to restore under a different parent object by using the `-p` parameter. (NOTE: Doesn't work for *replace* mode, as the new object always retains the parent of the replaced object)
  - **Always** skips any Collection workflow approval processes when restoring/replacing an Item in a Collection
  - **Never** adds a new Deposit License to Items (rather it restores the previous deposit license, as long as it is stored in the AIP)
  - **Never** adds new DSpace System metadata to Items (rather it just restores the metadata as specified in the AIP)

Changing Submission/Restore Behavior

It is possible to change some of the default behaviors of both the Submission and Restore/Replace Modes. Please see the Additional Packager Options section below for a listing of command-line options that allow you to override some of the default settings described above.

Submitting AIP(s) to create a new object

The Submission mode (`-s`) always creates a new object with a newly assigned handle. In addition by default it respects all existing Collection approval workflows (so items may require approval unless the workflow is skipped by using the `-w` option). For information about how the "Submission Mode" differs from the "Replace / Restore mode", see The difference between "Submit" and "Restore/Replace" modes above.

Submitting a Single AIP

AIPs treated as SIPs

This option allows you to essentially use an AIP as a SIP (Submission Information Package). The default settings will create a new DSpace object (with a new handle and a new parent object, if specified) from your AIP.

To ingest a single AIP and create a new DSpace object under a parent of your choice, specify the `-p` (or `--parent`) package parameter to the command. Also, note that you are running the `packager` in `-s` (submit) mode.

NOTE: This only ingests the single AIP specified. It does **not** ingest all children objects.

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

If you leave out the `-p` parameter, the AIP package ingester will attempt to install the AIP under the same parent it had before. As you are also specifying the `-s` (submit) parameter, the `packager` will assume you want a new Handle to be assigned (as you are effectively specifying that you are submitting a **new** object). If you want the object to retain the Handle specified in the AIP, you can specify the `-o ignoreHandle=false` option to force the packager to *not* ignore the Handle specified in the AIP.

Submitting an AIP Hierarchy

AIPs treated as SIPs

This option allows you to essentially use a set of AIPs as SIPs (Submission Information Packages). The default settings will create a new DSpace object (with a new handle and a new parent object, if specified) from each AIP

To ingest an AIP hierarchy from a directory of AIPs, use the `-a` (or `--all`) package parameter.

For example, use this 'packager' command template:

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

for example:

```
[dspace]/bin/dspace packager -s -a -t AIP -e admin@myu.edu -p 4321/12 aip4567.zip
```

The above command will ingest the package named "aip4567.zip" as a child of the specified Parent Object (handle="4321/12"). The resulting object is assigned a new Handle (since `-s` is specified). In addition, any child AIPs referenced by "aip4567.zip" are also recursively ingested (a new Handle is also assigned for each child AIP).

Another example – **Ingesting a Top-Level Community** (by using the Site Handle, `<site-handle-prefix>/0`):

```
[dspace]/bin/dspace packager -s -a -t AIP -e admin@myu.edu -p 4321/0 community-aip.zip
```

The above command will ingest the package named "community-aip.zip" as a **top-level community** (i.e. the specified parent is "4321/0" which is a Site Handle). Again, the resulting object is assigned a new Handle. In addition, any child AIPs referenced by "community-aip.zip" are also recursively ingested (a new Handle is also assigned for each child AIP).

May want to skip Collection Approvals Workflows

Please note: If you are submitting a larger amount of content (e.g. multiple Communities/Collections) to your DSpace, you may want to tell the 'packager' command to skip over any existing Collection approval workflows by using the `-w` flag. By default, all Collection approval workflows will be respected. This means if the content you are submitting includes a Collection with an enabled workflow, you may see the following occur:

1. First, the Collection will be created & its workflow enabled
2. Second, each Item belonging to that Collection will be created & placed into the workflow approval process

   Therefore, if this content has already received some level of approval, you may want to submit it using the `-w` flag, which will skip any workflow approval processes. For more information, see Submitting AIP(s) while skipping any Collection Approval Workflows.

Item Mappings may not be maintained when submitting an AIP hierachy

When an Item is mapped to one or more Collections, this mapping is recorded in the AIP using the mapped Collection's handle. Unfortunately, since the submission mode (-s) assigns **new handles** to all objects in the hierarchy, this may mean that the mapped Collection's handle will have changed (or even that a different Collection will be available at the original mapped Collection's handle). DSpace does not have a way to uniquely identify Collections other than by handle, which means that item mappings are only able to be retained when the Collection handle is *also retained.*

If you encounter this issue, there are a few possible workarounds:

1. Use the restore/replace mode (-r) instead, as it will retain existing Collection Handles. Unfortunately though, this may not work if the content is being moved from a Test DSpace to a Production DSpace, as these existing handles may not be valid.
2. OR, use the submission mode with the "--o ignoreHandle=false". This will also retain existing Collection Handles. Unfortunately though, this may not work if the content is being moved from a Test DSpace to a Production DSpace, as these existing handles may not be valid.
3. OR, remove all existing Item Mappings and re-export AIPs (without Item Mappings). Then, import the hierarchy into the new DSpace instance (again without Item Mappings). Finally, recreate the necessary Item Mappings using a different tool, e.g. the Batch Metadata Editing tool supports bulk editing of Collection memberships/mappings.

Missing Groups or EPeople cannot be created when submitting an individual Community or Collection AIP

Please note, if you are using AIPs to move an entire Community or Collection from one DSpace to another, there is a known issue (see https://github.com/DSpace/DSpace/issues/4477) that the new DSpace instance will be unable to (re-)create any DSpace Groups or EPeople which are referenced by a Community or Collection AIP. The reason is that the Community or Collection AIP itself doesn't contain enough information to create those Groups or EPeople (rather that info is stored in the SITE AIP, for usage during Full Site Restores).

However, there are two possible ways to get around this known issue:

- **EITHER**, you can manually recreate all referenced Groups/EPeople in the new DSpace that you are submitting the Community or Collection AIP into.
- **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. For example, if you are migrating one or more Collections/Communities from one DSpace to another, you may choose to submit those AIPs with the `-w` option enabled. This will ensure that, if a Collection has a workflow approval process enabled, all its Items are available immediately rather than being all placed into the workflow approval process.

Restoring/Replacing using AIP(s)

Restoring is slightly different than just **submitting**. When restoring, we make every attempt to restore the object as it **used to be** (including its handle, parent object, etc.). For more information about how the "Replace/Restore Mode" differs from the "Submit mode", see The difference between "Submit" and "Restore/Replace" modes above.

There are currently three restore modes:

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

Default Restore Mode

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

Restore a Single AIP: Use this 'packager' command template to restore a single object from an AIP (not including any child objects):

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

Restore a Hierarchy of AIPs: Use this 'packager' command template to restore an object from an AIP along with all child objects (from their AIPs):

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

For example:

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

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

In the above example, the package "aip4567.zip" is restored to the DSpace installation with the Handle provided within the package itself (and added as a child of the parent object specified within the package itself). In addition, any child AIPs referenced by "aip4567.zip" are also recursively ingested (the `-a` option specifies to also restore all child AIPs). They are also restored with the Handles & Parent Objects provided with their package. If any object is found to already exist, all changes are rolled back (i.e. nothing is restored to DSpace)

Highly Recommended to Update Database Sequences after a Large Restore

In some cases, when you restore a large amount of content to your DSpace, the internal database counts (called "sequences") may get out of sync with the Handles of the content you just restored. As a best practice, it is **highly recommended to always** re-run the "update-sequences" script on your DSpace database after a larger scale restore. This database script should be run while DSpace is stopped (you may either stop Tomcat or just the DSpace webapps). PostgreSQL/Oracle must be running. Simply run:
`[dspace]/bin/dspace database update-sequences`

More Information on using Default Restore Mode with Community/Collection AIPs

- Using the Default Restore Mode without the `-a` option, will only restore the **metadata** for that specific Community or Collection. No child objects will be restored.
- Using the Default Restore Mode with the `-a` option, will only successfully restore a Community or Collection if that object along with any child objects (Sub-Communities, Collections or Items) do not already exist. In other words, if any objects belonging to that Community or Collection already exist in DSpace, the Default Restore Mode will report an error that those object(s) could not be recreated. If you encounter this situation, you will need to perform the restore using either the Restore, Keep Existing Mode *or* the Force Replace Mode (depending on whether you want to keep or replace those existing child objects).

Restore, Keep Existing Mode

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

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

Restore a Hierarchy of AIPs: Use this 'packager' command template to restore an object from an AIP along with all child objects (from their AIPs):

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

For example:

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

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

Force Replace Mode

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

May also be useful in some specific restoration scenarios

This mode may also be used to restore missing objects which refer to existing objects. For example, if you are restoring a missing Collection which had existing Items linked to it, you can use this mode to auto-restore the Collection and update those existing Items so that they again link back to the newly restored Collection.

Potential for Data Loss

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

Replace using a Single AIP: Use this 'packager' command template to replace a single object from an AIP (not including any child objects):

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

Replace using a Hierarchy of AIPs: Use this 'packager' command template to replace an object from an AIP along with all child objects (from their AIPs):

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

For example:

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

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

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

Restoring Entire Site

In order to restore an entire Site from a set of AIPs, you must do the following:

1. Install a completely "fresh" version of DSpace by following the Installation instructions in the DSpace Manual
    - At this point, you should have a completely empty, but fully-functional DSpace installation. You will need to create an initial Administrator user in order to perform this restore (as a full-restore can only be performed by a DSpace Administrator).
2. Once DSpace is installed, run the following command to restore all its contents from AIPs

    ```
    [dspace]/bin/dspace packager -r -a -f -t AIP -e <eperson> -i <site-handle-prefix>/0 -o
    skipIfParentMissing=true /full/path/to/your/site-aip.zip
    ```

    a. While the "`-o skipIfParentMissing=true`" flag is optional, it is often necessary whenever you are performing a large hierarchical site restoration. Please see the Additional Packager Options section below.

Please note the following about the above restore command:

- Notice that you are running this command in "Force Replace" mode (`-r -f`). This is necessary as your empty DSpace install will already include a few default groups (Administrators and Anonymous) and your initial administrative user. You need to replace these groups in order to restore your prior DSpace contents completely.
- `<eperson>` should be replaced with the Email Address of the initial Administrator (who you created when you reinstalled DSpace).
- `<site-handle-prefix>` should be replaced with your DSpace site's assigned Handle Prefix. This is equivalent to the `handle.prefix` setting in your `dspace.cfg`
- `/full/path/to/your/site-aip.zip` is the full path to the AIP file which represents your DSpace SITE. This file will be named whatever you named it when you actually exported your entire site. All other AIPs are assumed to be referenced from this SITE AIP (in most cases, they should be in the same directory as that SITE AIP).

Highly Recommended to Update Database Sequences after a Large Restore

In some cases, when you restore a large amount of content to your DSpace, the internal database counts (called "sequences") may get out of sync with the Handles of the content you just restored. As a best practice, it is **highly recommended to always** re-run the "update-sequences" script on your DSpace database after a larger scale restore. This database script should be run while DSpace is stopped (you may either stop Tomcat or just the DSpace webapps). PostgreSQL/Oracle must be running. Simply run:

```
[dspace]/bin/dspace database update-sequences
```

Cleaning up from a failed import

Sometimes your packager import of AIP packages can fail, due to lack of memory (see below for advice on better performance, please use JAVA_OPTS to set your memory higher than the default). If that happens, DSpace by design will leave the bitstreams it **did** import sucessfully, but they will be oprphaned, and will just occupy space in your assetstore. The standard DSpace cleanup cron job will clean up these orphaned bitstreams, however, you can also clean them up manually by running the following command:

Clean up after a failed import

```
[dspace]/bin/dspace cleanup -v
```

Performance considerations

When importing large structures like the whole site or a large collection/community, keep in mind that this can require a lot of memory, more than the default amount of heap allocated to the command-line launcher (256 Mb: `JAVA_OPTS="-Xmx256m -Dfile.encoding=UTF-8"`). This memory must be allocated in addition to the normal amount of memory allocated to Tomcat. For example, a site of 2500 fulltext items (2 Gb altogether) requires 5 Gb of maximum heap space and takes around 1 hour, including import and indexing.

You can raise the limit for a single run of the packager command by specifying memory options in the JAVA_OPTS environment variable, e.g.:

```
JAVA_OPTS="-Xmx4096m -Dfile.encoding=UTF-8" /dspace/bin/dspace packager -u -r -a -f -t AIP -e dspace@example.
com -i 123456789/0 sitewide-aip.zip
```

If the importer runs out of heap memory, it will crash either with "java.lang.OutOfMemoryError: GC overhead limit exceeded", which can be suppressed by adding "-XX:-UseGCOverheadLimit" to JAVA_OPTS, or with "java.lang.OutOfMemoryError: Java heap space". You can increase the allocated heap memory and try again, but keep in mind that although no changes were made in the database, the unsuccessfully imported files are still left in the assetstore (see https://github.com/DSpace/DSpace/issues/5593).

Disable User Interaction for Cron

If you wish to run any of the following commands from a cron job (or similar), then you may wish to **disable all user interaction** using the `-u` (`--no-user-interaction`) flag. For example, supposing you wanted to perform a full Site Backup (see Exporting Entire Site above) via a cronjob, you could simply run that command passing it the "-u" flage like this:

```
# Perform a full site backup to AIPs(with user interaction disabled) every Sunday at 1:00AM
# NOTE: Make sure to replace "123456789" with your actual Handle Prefix, and "admin@myu.edu" with your
Administrator account email.
0 1 * * * [dspace]/bin/dspace packager -u -d -a -t AIP -e admin@myu.edu -i 123456789/0 [full-path-to-backup-
folder]/sitewide-aip.zip
```

Command Line Reference

The following flags are valid to pass to the `[dspace]/bin/dspace packager` command:

Flag	Ingest or Export	Description / Usage
`-a (--all)`	both ingest and export	*For Ingest*: recursively ingest all child AIPs (referenced from this AIP). *For Export*: recursively export all child objects (referenced from this parent object)
`-d (--disseminate)`	export-only	This flag simply triggers the export of AIPs from the system. See Exporting AIPs
`-e (–eperson) [email-address]`	ingest-only	The email address of the EPerson who is ingesting the AIPs. Oftentimes this should be an Administrative account.
`-f (--force-replace)`	ingest-only	Ingest the AIPs in "Force Replace Mode" (must be specified in conjunction with `-r` flag), where existing objects will be replaced by the contents of the AIP.

`-h (--help)`	both ingest and export	Return help information. You should specify with `-t` for additional package specific help information
`-i (--identifier) [handle]`	both ingest and export	*For Ingest*: Only valid in "Force Replace Mode". In that mode this is the identifier of the object to replace. *For Export:* The identifier of the object to export to an AIP
`-k (--keep-existing)`	ingest-only	Specifies to use "Restore, Keep Existing Mode" during ingest (must be specified in conjunction with `-r` flag). In this mode, existing objects in DSpace will NOT be replaced by their AIPs, but missing objects will be restored from AIPs.
`-o (--option) [setting]=[value]`	both ingest and export	This flag is used to pass Additional Packager Options to the Packager command. Each type of packager may define its own custom Additional Options. For AIPs, the valid options are documented in the Additional Packager Options section below. This is repeatable (e.g. `-o [setting1]=[value] -o [setting2]=value`)
`-p (--parent) [handle]`	ingest only	Handle(s) of the parent Community or Collection to into which an AIP should be ingested. This may be repeatable.
`-r (--restore)`	ingest only	Specifies that this ingest is either "Restore Mode" (when standalone), "Restore, Keep Existing Mode" (when used with `-k` flag) or "Force Replace Mode" (when used with `-f` flag)
`-s (--submit)`	ingest only	Specifies that this ingest is in "Submit Mode" where an AIP is treated as a **new** object and assigned a new Handle /Identifier, etc.
`-t (--type) [package-type]`	both ingest and export	Specifies the type of package which is being ingested or exported. This controls which Ingester or Disseminator class is called. For AIPs, this is **always** set to "`-t AIP`"
`-u (--no-user-interaction)`	both ingest and export	Skips over all user interaction (e.g. question prompts). This flag can be used when running the packager from a script or cron job to bypass all user interaction. See also Disable User Interaction for Cron

Additional Packager Options

In additional to the various "modes" settings described under "Running the Code" above, the AIP Packager supports the following packager options. These options allow you to better tweak how your AIPs are processed (especially during ingests/restores/replaces).

Option	Ingest or Export	Default Value	Description
`filterBundles=[value]`	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=[value]`	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=[value]`	ingest-only	Restore/Replace Mode defaults to 'false', Submit Mode defaults to 'true'	If 'true', the AIP ingester will ignore any Parent object specified in the AIP itself, and instead ingest under a new Parent object (this is the default when running in Submit mode, using the `-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=[value]`	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=[value]`	both ingest and export	false	If 'true', the AIP Disseminator will only import/export a METS Manifest XML file (i.e. result will be an unzipped 'mets.xml' file), instead of a full AIP. This METS Manifest contains URI references to all content files, but does *not* contain any content files. **This option is experimental and is meant for debugging purposes only. It should never be set to 'true' if you want to be able to restore content files.** Again, please note that when you use this option, the final result will be an XML file, NOT the normal ZIP-based AIP format.
`passwords=[value]`	export-only	false	If 'true' (and the 'DSPACE-ROLES' crosswalk is enabled, see #AIP Metadata Dissemination Configurations), then the AIP Disseminator will export user password hashes (i.e. encrypted passwords) into Site AIP's METS Manifest. This would allow you to restore user's passwords from Site AIP. If 'false', then user password hashes are not stored in Site AIP, and passwords cannot be restored at a later time.
`skipIfParentMissing=[value]`	ingest-only	false	If 'true', ingestion will skip over any "Could not find a parent DSpaceObject" errors that are encountered during the ingestion process (Note: those errors will still be logged as "warning" messages in your DSpace log file). If you are performing a full site restore (or a restore of a larger Community/Collection hierarchy), you may encounter these errors if you have a larger number of Item mappings between Collections (i.e. Items which are mapped into several collections at once). When you are performing a recursive ingest, skipping these errors should not cause any problems. Once the missing parent object is ingested it will automatically restore the Item mapping that caused the error. For more information on this "Could not find a parent DSpaceObject" error see Common Issues or Error Messages.
`unauthorized=[value]`	export-only	*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=[value]`	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=[value]`	both ingest and export	Export defaults to 'true', Ingest defaults to 'false'	If 'true', every METS file in AIP will be validated before ingesting or exporting. By default, DSpace will validate everything on export, but will skip validation during import. Validation on export will ensure that all exported AIPs properly conform to the METS profile (and will throw errors if any do not). Validation on import will ensure every METS file in every AIP is first validated before importing into DSpace (this will cause the ingestion processing to take longer, but tips on speeding it up can be found in the "AIP Configurations To Improve Ingestion Speed while Validating" section below). *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 additional options

These options can be passed in two main ways:

From the Command Line

From the command-line, you can add the option to your command by using the `-o` or `--option` parameter.

```
 [dspace]/bin/dspace packager -r -a -t AIP -o [option1]=[value] -o [option2]=[value] -e admin@myu.edu aip4567.zip
```

For example:

```
 [dspace]/bin/dspace packager -r -a -t AIP -o ignoreParent=false -o createMetadataFields=false -e admin@myu.edu aip4567.zip
```

Via the Java API call

If you are programmatically calling the `org.dspace.content.packager.DSpaceAIPIngester` from your own custom script, you can specify these options via the `org.dspace.content.packager.PackageParameters` class.

As a basic example:

```
PackageParameters params = new PackageParameters;
params.addProperty("createMetadataFields", "false");
params.addProperty("ignoreParent", "true");
```

Configuration in 'dspace.cfg'

The following new configurations relate to AIPs:

AIP Metadata Dissemination Configurations

The following configurations allow you to specify what metadata is stored within each METS-based AIP. In 'dspace.cfg', the general format for each of these settings is:

- `aip.disseminate.<setting> = <mdType>:<DSpace-crosswalk-name> [, ...]`
  - <setting> is the setting name (see below for the full list of valid settings)
  - <mdType> is optional. It allows you to specify the value of the @MDTYPE or @OTHERMDTYPE attribute in the corresponding METS element.
  - <DSpace-crosswalk-name> is required. It specifies the name of the DSpace Crosswalk which should be used to generate this metadata.
  - Zero or more `<label-for-METS>:<DSpace-crosswalk-name>` may be specified for each setting

AIP Metadata Recommendations

It is recommended to **minimally** use the default settings when generating AIPs. DSpace can only restore information that is included within an AIP. Therefore, if you choose to no longer include some information in an AIP, DSpace will no longer be able to restore that information from an AIP backup

The default settings in 'dspace.cfg' are:

- `aip.disseminate.techMD` - Lists the DSpace Crosswalks (by name) which should be called to populate the `<techMD>` section of the METS file within the AIP (Default: `PREMIS, DSPACE-ROLES`)
  - The `PREMIS` crosswalk generates PREMIS metadata for the object specified by the AIP
  - The `DSPACE-ROLES` crosswalk exports DSpace Group / EPerson information into AIPs in a DSpace-specific XML format. Using this crosswalk means that AIPs can be used to recreated Groups & People within the system. (NOTE: The `DSPACE-ROLES` crosswalk should be used alongside the `METSRights` crosswalk if you also wish to restore the *permissions* that Groups/People have within the System. See below for more info on the `METSRights` crosswalk.)
- `aip.disseminate.sourceMD` - Lists the DSpace Crosswalks (by name) which should be called to populate the `<sourceMD>` section of the METS file within the AIP (Default: `AIP-TECHMD`)
  - The AIP-TECHMD Crosswalk generates technical metadata (in DIM format) for the object specified by the AIP
- `aip.disseminate.digiprovMD` - Lists the DSpace Crosswalks (by name) which should be called to populate the `<digiprovMD>` section of the METS file within the AIP (Default: *None*)
- `aip.disseminate.rightsMD` - Lists the DSpace Crosswalks (by name) which should be called to populate the `<rightsMD>` section of the METS file within the AIP (Default: `DSpaceDepositLicense:DSPACE_DEPLICENSE, CreativeCommonsRDF:DSPACE_CCRDF, CreativeCommonsText:DSPACE_CCTEXT, METSRights`)
  - The `DSPACE_DEPLICENSE` crosswalk ensures the DSpace Deposit License is referenced/stored in AIP
  - The `DSPACE_CCRDF` crosswalk ensures any Creative Commons RDF Licenses are reference/stored in AIP
  - The `DSPACE_CCTEXT` crosswalk ensures any Creative Commons Textual Licenses are referenced/stored in AIP
  - The `METSRights` crosswalk ensures that Permissions/Rights on DSpace Objects (Communities, Collections, Items or Bitstreams) are referenced/stored in AIP. Using this crosswalk means that AIPs can be used to restore permissions that a particular Group or Person had on a DSpace Object. (NOTE: The `METSRights` crosswalk should always be used in conjunction with the `DSPACE-ROLES` crosswalk (see above) or a similar crosswalk. The `METSRights` crosswalk can only restore permissions, and cannot re-create Groups or EPeople in the system. The `DSPACE-ROLES` can actually re-create the Groups or EPeople as needed.)
- `aip.disseminate.dmd` - Lists the DSpace Crosswalks (by name) which should be called to populate the `<dmdSec>`section of the METS file within the AIP (Default: MODS, DIM)
  - The MODS crosswalk translates the DSpace descriptive metadata (for this object) into MODS. As MODS is a relatively "standard" metadata schema, it may be useful to include a copy of MODS metadata in your AIPs if you should ever want to import them into another (non-DSpace) system.
  - The DIM crosswalk just translates the DSpace internal descriptive metadata into an XML format. This XML format is proprietary to DSpace, but stores the metadata in a format similar to Qualified Dublin Core.

AIP Ingestion Metadata Crosswalk Configurations

The following configurations allow you to specify what DSpace Crosswalks are used during the ingestion/restoration of AIPs. These configurations also allow you to ignore areas of the METS file (in the AIP) if you do not want that area to be restored.

In `dspace.cfg`, the general format for each of these settings is:

- `mets.dspaceAIP.ingest.crosswalk.<mdType> = <DSpace-crosswalk-name>`
  - <mdType> is the type of metadata as specified in the METS file. This corresponds to the value of the @MDTYPE attribute (of that metadata section in the METS). When the @MDTYPE attribute is "OTHER", then the <mdType> corresponds to the @OTHERMDTYPE attribute value.
  - <DSpace-crosswalk-name> specifies the name of the DSpace Crosswalk which should be used to ingest this metadata into DSpace. You can specify the "NULLSTREAM" crosswalk if you specifically want this metadata to be ignored (and skipped over during ingestion).

By default, the settings in `dspace.cfg` are:

```
mets.dspaceAIP.ingest.crosswalk.DSpaceDepositLicense = NULLSTREAM
mets.dspaceAIP.ingest.crosswalk.CreativeCommonsRDF = NULLSTREAM
mets.dspaceAIP.ingest.crosswalk.CreativeCommonsText = NULLSTREAM
```

The above settings tell the ingester to **ignore** any metadata sections which reference DSpace Deposit Licenses or Creative Commons Licenses. These metadata sections can be safely ignored as long as the "LICENSE" and "CC_LICENSE" bundles are included in AIPs (which is the default setting). As the Licenses are included in those Bundles, they will already be restored when restoring the bundle contents.

More Info on Default Crosswalks used

If unspecified in the above settings, the AIP ingester will automatically use the Crosswalk which is named the same as the @MDTYPE or @OTHERMDTYPE attribute for the metadata section. For example, a metadata section with an @MDTYPE="PREMIS" will be processed by the DSpace Crosswalk named "PREMIS".

AIP Ingestion EPerson Configurations

The following setting determines whether the AIP Ingester should create an EPerson (if necessary) when attempting to restore or ingest an Item whose Submitter cannot be located in the system. By default it is set to "false", as for AIPs the creation of EPeople (and Groups) is generally handled by the `DSPACE-ROLES` crosswalk (see #AIP Metadata Dissemination Configurations for more info on `DSPACE-ROLES` crosswalk.)

- `mets.dspaceAIP.ingest.createSubmitter = false`

AIP Configurations To Improve Ingestion Speed while Validating

It is recommended to validate all AIPs on ingestion (when possible). But validation can be extremely slow, as each validation request first must download all referenced Schema documents from various locations on the web (sometimes as many as 10 schemas may be necessary to download in order to validate a single METS file). To make matters worse, the same schema will be re-downloaded each time it is used (i.e. it is not cached locally). So, if you are validating just 20 METS files which each reference 10 schemas, that results in 200 download requests.

In order to perform validations in a speedy fashion, you can pull down a local copy of **all** schemas. Validation will then use this local cache, which can sometimes increase the speed up to 10 x.

To use a local cache of XML schemas when validating, use the following settings in 'dspace.cfg'. The general format is:

- `mets.xsd.<abbreviation> = <namespace> <local-file-name>`
  - `<abbreviation>` is a unique abbreviation (of your choice) for this schema
  - `<namespace>` is the Schema namespace
  - `<local-file-name>` the full name of the cached schema file (which should reside in your `[dspace]/config/schemas/` directory, by default this directory does not exist – you will need to create it)

The default settings are all commented out. But, they provide a full listing of all schemas currently used during validation of AIPs. In order to utilize them, uncomment the settings, download the appropriate schema file, and save it to your `[dspace]/config/schemas/` directory (by default this directory does not exist – you will need to create it) using the specified file name:

```
#mets.xsd.mets = http://www.loc.gov/METS/ mets.xsd
#mets.xsd.xlink = http://www.w3.org/1999/xlink xlink.xsd
#mets.xsd.mods = http://www.loc.gov/mods/v3 mods.xsd
#mets.xsd.xml = http://www.w3.org/XML/1998/namespace xml.xsd
#mets.xsd.dc = http://purl.org/dc/elements/1.1/ dc.xsd
#mets.xsd.dcterms = http://purl.org/dc/terms/ dcterms.xsd
#mets.xsd.premis = http://www.loc.gov/standards/premis PREMIS.xsd
#mets.xsd.premisObject = http://www.loc.gov/standards/premis PREMIS-Object.xsd
#mets.xsd.premisEvent = http://www.loc.gov/standards/premis PREMIS-Event.xsd
#mets.xsd.premisAgent = http://www.loc.gov/standards/premis PREMIS-Agent.xsd
#mets.xsd.premisRights = http://www.loc.gov/standards/premis PREMIS-Rights.xsd
```

Common Issues or Error Messages

The below table lists common fixes to issues you may encounter when backing up or restoring objects using AIP Backup and Restore.

Issue / Error Message	How to Fix this Problem

Ingest /Restore Error: "Group Administrator already exists"	If you receive this problem, you are likely attempting to Restore an Entire Site, but are not running the command in Force Replace Mode (`-r -f`). Please see the section on Restoring an Entire Site for more details on the flags you should be using.
Ingest /Restore Error: "Unknown Metadata Schema encountered (mycustomschema)"	If you receive this problem, one or more of your Items is using a custom metadata schema which DSpace is currently not aware of (in the example, the schema is named "mycustomschema"). Because DSpace AIPs do not contain enough details to recreate the missing Metadata Schema, you must create it manually via the DSpace Admin UI. **Please note that you only need to create the Schema. You do not need to manually create all the fields belonging to that schema, as DSpace will do that for you as it restores each AIP.** Once the schema is created in DSpace, re-run your restore command. DSpace will automatically re-create all fields belonging to that custom metadata schema as it restores each Item that uses that schema.
Ingest Error: "Could not find a parent DSpaceObject referenced as 'xxx/xxx'"	When you encounter this error message it means that an object could not be ingested/restored as it belongs to a **parent** object which doesn't currently exist in your DSpace instance. During a full restore process, this error can be skipped over and treated as a warning by specifying the '`-o skipIfParentMissing=true`' option (see Additional Packager Options). If you have a larger number of Items which are mapped to multiple Collections, the AIP Ingester will sometimes attempt to restore an item mapping before the **Collection itself** has been restored (thus throwing this error). Luckily, this is not anything to be concerned about. As soon as the Collection is restored, the Item Mapping which caused the error will also be automatically restored. So, if you encounter this error during a full restore, it is safe to bypass this error message using the '`-o 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]/bin/dspace database update-sequences`

DSpace AIP Format

1 Makeup and Definition of AIPs
- 1.1 AIPs are Archival Information Packages.
- 1.2 General AIP Structure / Examples
  - 1.2.1 Customizing What Is Stored in Your AIPs

2 AIP Details: METS Structure

3 Metadata in METS
- 3.1 DIM (DSpace Intermediate Metadata) Schema
  - 3.1.1 DIM Descriptive Elements for Item objects
  - 3.1.2 DIM Descriptive Elements for Collection objects
  - 3.1.3 DIM Descriptive Elements for Community objects
  - 3.1.4 DIM Descriptive Elements for Site objects
- 3.2 MODS Schema
- 3.3 AIP Technical Metadata Schema (AIP-TECHMD)
  - 3.3.1 AIP Technical Metadata for Item
  - 3.3.2 AIP Technical Metadata for Bitstream
  - 3.3.3 AIP Technical Metadata for Collection
  - 3.3.4 AIP Technical Metadata for Community
  - 3.3.5 AIP Technical Metadata for Site
- 3.4 PREMIS Schema
  - 3.4.1 PREMIS Metadata for Bitstream
- 3.5 DSPACE-ROLES Schema
  - 3.5.1 Example of DSPACE-ROLES Schema for a SITE AIP
  - 3.5.2 Example of DSPACE-ROLES Schema for a Community or Collection
- 3.6 METSRights Schema
  - 3.6.1 Example of METSRights Schema for an Item
  - 3.6.2 Example of METSRights Schema for a Collection
  - 3.6.3 Example of METSRights Schema for a Community

Makeup and Definition of AIPs

AIPs only store the Latest Version of Items

If you are using the Item Level Versioning functionality (disabled by default), you must be aware that this "Item Level Versioning" feature is **not yet compatible** with AIP Backup & Restore. **Using them together may result in accidental data loss.** Currently the AIPs that DSpace generates only store the *latest version* of an Item. Therefore, past versions of Items will always be lost when you perform a restore / replace using AIP tools.

AIPs are Archival Information Packages.

- AIP is a package describing **one archival object** in DSpace.
  - The **archival object** may be a single **Item**, **Collection**, **Community**, or **Site** (Site AIPs contain site-wide information). Bitstreams are included in an Item's AIP.
  - Each AIP is logically self-contained, can be restored without rest of the archive. (So you could restore a single Item, Collection or Community)
  - Collection or Community AIPs do **not** include all child objects (e.g. Items in those Collections or Communities), as each AIP only describes **one** object. However, these container AIPs do contain references (links) to all child objects. These references can be used by DSpace to automatically restore all referenced AIPs when restoring a Collection or Community.
  - AIPs are only generated for objects which are currently in the "in archive" state in DSpace. This means that in-progress, uncompleted submissions are not described in AIPs and cannot be restored after a disaster. Permanently removed objects will also no longer be exported as AIPs after their removal. However, withdrawn objects will continue to be exported as AIPs, since they are still considered under the "in archive" status.
  - AIPs with identical contents will always have identical checksums. This provides a basic means of validating whether the contents within an AIP have changed. For example, if a Collection's AIP has the same checksum at two different points in time, it means that Collection has not changed during that time period.
  - AIP profile favors completeness and accuracy rather than presenting the semantics of an object in a standard format. It conforms to the quirks of DSpace's internal object model rather than attempting to produce a universally understandable representation of the object. When possible, an AIP tries to use common standards to express objects.
  - An AIP *can* serve as a DIP (Dissemination Information Package) or SIP (Submission Information Package), especially when transferring custody of objects to another DSpace implementation.
  - In contrast to SIP or DIP, the AIP should include all available DSpace structural and administrative metadata, and basic provenance information. AIPs also describe some basic system level information (e.g. Groups and People).

General AIP Structure / Examples

Generally speaking, an AIP is an Zip file containing a METS manifest and all related content bitstreams, license files and any other associated files.

Some examples include:

- Site AIP (Sample: SITE-example.zip)
  - METS contains basic metadata about DSpace Site and persistent IDs referencing all Top Level Communities
  - METS also contains a list of all Groups and EPeople information defined in the DSpace system. (NOTE: By default, user passwords are not stored in AIPs, unless you specify the 'passwords' flag. See Additional Packager Options.)
- Community AIP (Sample: COMMUNITY@123456789-1.zip)

    - METS contains all metadata for Community and persistent IDs referencing all members (SubCommunities or Collections). Package may also include a Logo file, if one exists.
    - METS contains any Group information for Community-specific groups (e.g. `COMMUNITY_<ID>_ADMIN` group).
    - METS contains all Community permissions/policies (translated into METSRights schema)
- Collection AIP (Sample: COLLECTION@123456789-2.zip)
    - METS contains all metadata for Collection and persistent IDs referencing all members (Items). Package may also include a Logo file, if one exists.
    - METS contains any Group information for Collection-specific groups (e.g. `COLLECTION_<ID>_ADMIN`, `COLLECTION_<ID>_SUBMIT`, etc.).
    - METS contains all Collection permissions/policies (translated into METSRights schema)
    - If the Collection has an Item Template, the METS will also contain all the metadata for that Item Template.
- Item AIP (Sample: ITEM@123456789-8.zip)
    - METS contains all metadata for Item and references to all Bundles and Bitstreams. Package also includes all Bitstream files.
    - METS contains all Item/Bundle/Bitstream permissions/policies (translated into METSRights schema)

Notes:

- Bitstreams and Bundles are second-class archival objects; they are recorded in the context of an Item.
- BitstreamFormats are not even second-class; they are described implicitly within Item technical metadata, and reconstructed from that during restoration
- EPeople are only defined in Site AIP, but may be referenced from Community or Collection AIPs
- Groups may be defined in Site AIP, Community AIP or Collection AIP. Where they are defined depends on whether the Group relates specifically to a single Community or Collection, or is just a general site-wide group.

What is NOT in AIPs

- DSpace Site configurations ([dspace]/config/ directory) or customizations (themes, stylesheets, etc) are not described in AIPs
- DSpace Database model (or customizations therein) is not described in AIPs
- Any objects which are not currently in the "In Archive" state are not described in AIPs. This means that in-progress, unfinished submissions are never included in AIPs.

Customizing What Is Stored in Your AIPs

If you choose, you can customize exactly what information is stored in your AIPs. However, you should be aware that you can only restore information which is stored within your AIPs. If you choose to remove information from your AIPs, you will be unable to restore it later on (unless you are also backing up your entire DSpace database and assetstore folder).

AIP Recommendations

It is recommended to minimally use the default settings when generating AIPs. DSpace can only restore information that is included within an AIP. Therefore, if you choose to no longer include some information in an AIP, DSpace will no longer be able to restore that information from an AIP backup

There are two ways to go about customizing your AIP format:

1. You can customize your `dspace.cfg` settings pertaining to AIP generation. These configurations will allow you to specify exactly which DSpace Crosswalks will be called when generating the AIP METS manifest.
2. You can export your AIPs using one of the special options/flags.

AIP Details: METS Structure

This METS Structure is based on the structure decided for the original AipPrototype, developed as part of the MIT & UCSD PLEDGE project.

- `mets` element
    - `@PROFILE` fixed value="http://www.dspace.org/schema/aip/1.0/mets.xsd" (this is how we identify an AIP manifest)
    - `@OBJID` URN-format persistent identifier (i.e. Handle) if available, or else a unique identifier. (e.g. "hdl:123456789/1")
    - `@LABEL` title if available
    - `@TYPE` DSpace object type, one of "DSpace ITEM", "DSpace COLLECTION", "DSpace COMMUNITY" or "DSpace SITE".
    - `@ID` is a globally unique identifier, built using the Handle and the Object type (e.g. `dspace-COLLECTION-hdl:123456789/3`).
- `mets/metsHdr` element
    - `@LASTMODDATE` last-modified date for a DSpace Item, or nothing for other objects.
    - `agent` element:
        - `@ROLE` = "CUSTODIAN",
        - `@TYPE` = "OTHER",
        - `@OTHERTYPE` = "DSpace Archive",
        - `name` = *Site handle*. (Note: The Site Handle is of the format `[handle_prefix]/0`, e.g. "123456789/0")
    - `agent` element:
        - `@ROLE` = "CREATOR",
        - `@TYPE` = "OTHER",
        - `@OTHERTYPE` = "DSpace Software",
        - `name` = "DSpace [version]" (Where "[version]" is the specific version of DSpace software which created this AIP, e.g. "1.7.0")
- `mets/dmdSec` element(s)
    - By default, two `dmdSec` elements are included for all AIPs:
        1. object's descriptive metadata crosswalked to MODS (specified by `mets/dmdSec/mdWrap@MDTYPE="MODS"`). See #MODS Schema section below for more information.
        2. object's descriptive metadata in DSpace native DIM intermediate format, to serve as a complete and precise record for restoration or ingestion into another DSpace. Specified by `mets/dmdSec/mdWrap@MDTYPE="OTHER",@OTHERMDTYPE="DIM"`. See #DIM (DSpace Intermediate Metadata) Schema section below for more information.

    - For Collection AIPs, additional dmdSec elements may exist which describe the Item Template for that Collection. Since an Item template is not an actual Item (i.e. it only includes metadata), it is stored within the Collection AIP. The Item Template's dmdSec elements will be referenced by a `div @TYPE="DSpace ITEM Template"` in the METS `structMap`.
    - When the `mdWrap` `@TYPE` value is `OTHER`, the element *MUST* include a value for the `@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 section below for more information.
            - `DSPACE-ROLES` metadata may appear here to describe the Groups or EPeople related to this object (_currently only specified for Site, Community and Collection). Specified by `mdWrap@MDTYPE="OTHER",@OTHERMDTYPE="DSPACE-ROLES"`. See #DSPACE-ROLES Schema section below for more information.
        - `rightsMD` elements. By default, there are four possible types of `rightsMD` elements which may be included:
            - `METSRights` metadata may appear here to describe the permissions on this object. Specified by `mdWrap@MDTYPE="OTHER",@OTHERMDTYPE="METSRIGHTS"`. See #METSRights Schema section below for more information.
            - `DSpaceDepositLicense` if the object is an Item and it has a deposit license, it is contained here. Specified by `mdWrap@MDTYPE="OTHER",@OTHERMDTYPE="DSpaceDepositLicense"`.
            - `CreativeCommonsRDF` If the object is an Item with a Creative Commons license expressed in RDF, it is included here. Specified by `mdWrap@MDTYPE="OTHER",@OTHERMDTYPE="CreativeCommonsRDF"`.
            - `CreativeCommonsText` If the object is an Item with a Creative Commons license in plain text, it is included here. Specified by `mdWrap@MDTYPE="OTHER",@OTHERMDTYPE="CreativeCommonsText"`.
        - `sourceMD` element. By default, there is only one type of `sourceMD` element which may appear:
            - `AIP-TECHMD` metadata may appear here. This stores basic technical/source metadata about in object in a DSpace native format. Specified by `mdWrap@MDTYPE="OTHER",@OTHERMDTYPE="AIP-TECHMD"`. See #AIP Technical Metadata Schema (AIP-TECHMD) section below for more information.
        - `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:
                a. One linking to the Handle of that Item. Its `@LOCTYPE="HANDLE"`, and `@xlink:href` value is the raw Handle.
                b. (Optional) one linking to the location of the local AIP for that Item (if known). Its `@LOCTYPE="URL"`, and `@xlink:href` value is a relative link to the AIP file on the local filesystem.
            - If Collection has a Logo bitstream, there is an `fptr` reference to it in the very first `div`.
            - If the Collection includes an Item Template, there will be a `div` with `@TYPE="DSpace ITEM Template"` within the very first `div`. This `div @TYPE="DSpace ITEM Template"` must have a `@DMDID` specified, which links to the dmdSec element(s) that contain the metadata for the Item Template.
    - For COMMUNITY objects:
        1. Top-Level `div` with `@TYPE="DSpace Object Contents"`.
            - For every Sub-Community in the Community it contains a `div` with `@TYPE="DSpace COMMUNITY"`. Each Community `div` has up to two `mptr`elements:
                a. One linking to the Handle of that Community. Its `@LOCTYPE="HANDLE"`, and `@xlink:href` value is the raw Handle.
                b. (Optional) one linking to the location of the local AIP file for that Community (if known). Its `@LOCTYPE="URL"`, and `@xlink:href` value is a relative link to the AIP file on the local filesystem.
            - For every Collection in the Community there is a `div` with `@TYPE="DSpace COLLECTION"`. Each Collection `div` has up to two `mptr`elements:
                a. One linking to the Handle of that Collection. Its `@LOCTYPE="HANDLE"`, and `@xlink:href` value is the raw Handle.

            b. (Optional) one linking to the location of the local AIP file for that Collection (if known). Its `@LOCTYPE="URL"`, and `@xlink:href` value is a relative link to the AIP file on the local filesystem.
        - If Community has a Logo bitstream, there is an `fptr` reference to it in the very first `div`.
    - For SITE objects:
        1. Top-Level `div` with `@TYPE="DSpace Object Contents"`.
            - For every Top-level Community in Site, it contains a `div` with `@TYPE="DSpace COMMUNITY"`. Each Item `div` has up to two child `mptr`elements:
                a. One linking to the Handle of that Community. Its `@LOCTYPE="HANDLE"`, and `@xlink:href` value is the raw Handle.
                b. (Optional) one linking to the location of the local AIP for that Community (if known). Its `@LOCTYPE="URL"`, and `@xlink:href` value is a relative link to the AIP file on the local filesystem.
- `mets/structMap` - Structure Map to indicate object's Parent, `@LABEL="Parent", @TYPE="LOGICAL"`
    - Contains one `div` element which has the unique attribute value `TYPE="AIP Parent Link"` to identify it as the older of the *parent pointer*.
        - It contains a `mptr` element whose `xlink:href` attribute value is the raw Handle of the parent object, e.g. `1721.1/4321`.

Metadata in METS

The following tables describe how various metadata schemas are populated (via DSpace Crosswalks) in the METS file for an AIP.

DIM (DSpace Intermediate Metadata) Schema

DIM Schema is essentially a way of representing DSpace internal metadata structure in XML. DSpace's internal metadata is very similar to a Qualified Dublin Core in its structure, and is primarily meant for descriptive metadata. However, DSpace's metadata allows for custom elements, qualifiers or schemas to be created (so it is extendable to any number of schemas, elements, qualifiers). These custom fields/schemas may or may not be able to be translated into normal Qualified Dublin Core. So, the DIM Schema must be able to express metadata schemas, elements or qualifiers which may or may not exist within Qualified Dublin Core.

In the METS structure, DIM metadata always appears within a `dmdSec` inside an `<mdWrap MDTYPE="OTHER" OTHERMDTYPE="DIM">` element. For example:

```
<dmdSec ID="dmdSec_2190">
   <mdWrap MDTYPE="OTHER" OTHERMDTYPE="DIM">
    ...
   </mdWrap>
</dmdSec>
```

By default, DIM metadata is always included in AIPs. It is controlled by the following configuration in your `dspace.cfg`:

```
aip.disseminate.dmd = MODS, DIM
```

DIM Descriptive Elements for Item objects

As all DSpace Items already have user-assigned DIM (essentially Qualified Dublin Core) metadata fields, those fields are just exported into the DIM Schema within the METS file.

DIM Descriptive Elements for Collection objects

For Collections, the following fields are translated to the DIM schema:

DIM Metadata Field	Database field or value
dc.description	'introductory_text' field
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 within a `<premis:premis>` element. For example:

```
<amdSec ID="amd_2209">
    ...
    <techMD ID="techMD_2210">
       <mdWrap MDTYPE="PREMIS">
          <premis:premis>
            ...
          </premis:premis>
       </mdWrap>
    </techMD>
    ...
</amdSec>
```

Each Bitstream (file) has its own `amdSec` within a METS manifest. So, there will be a separate PREMIS `techMD` for each Bitstream within a single Item.

By default, PREMIS metadata is always included in AIPs. It is controlled by the following configuration in your `dspace.cfg`:

```
aip.disseminate.techMD = PREMIS, DSPACE-ROLES
```

PREMIS Metadata for Bitstream

The following Bitstream information is translated into PREMIS for each DSpace Bitstream (file):

Metadata Field	Value
<premis:objectIdentifier>	Contains Bitstream direct URL
<premis:objectCategory>	Always set to "File"
<premis:fixity>	Contains MD5 Checksum of Bitstream
<premis:format>	Contains File Format information of Bistream
<premis:originalName>	Contains original name of file

DSPACE-ROLES Schema

All DSpace Groups and EPeople objects are translated into a custom `DSPACE-ROLES` XML Schema. This XML Schema is a very simple representation of the underlying DSpace database model for Groups and EPeople. The `DSPACE-ROLES` Schemas is generated by DSpace's `RoleCrosswalk`.

Only the following DSpace Objects utilize the DSPACE-ROLES Schema in their AIPs:

- Site AIP – all Groups and EPeople are represented in DSPACE-ROLES Schema
- Community AIP – only Community-based groups (e.g. `COMMUNITY_1_ADMIN`) are represented in DSPACE-ROLES Schema
- Collection AIP – only Collection-based groups (e.g. `COLLECTION_2_ADMIN`, `COLLECTION_2_SUBMIT`, etc.) are represented in DSPACE-ROLES Schema

In the METS structure, DSPACE-ROLES metadata always appears within a `techMD` inside an `<mdWrap MDTYPE="OTHER" OTHERMDTYPE="DSPACE-ROLES">` element. For example:

```
<amdSec ID="amd_2068">
    ...
    <techMD ID="techMD_2070">
       <mdWrap MDTYPE="OTHER" OTHERMDTYPE="DSPACE-ROLES">
            ...
       </mdWrap>
    </techMD>
    ...
</amdSec>
```

By default, DSPACE-ROLES metadata is always included in AIPs. It is controlled by the following configuration in your `dspace.cfg`:

```
aip.disseminate.techMD = PREMIS, DSPACE-ROLES
```

Example of DSPACE-ROLES Schema for a SITE AIP

Below is a general example of the structure of a DSPACE-ROLES XML file, as it would appear in a SITE AIP.

```
<DSpaceRoles>
  <Groups>
    <Group ID="1" Name="Administrator">
      <Members>
        <Member ID="1" Name="bsmith@myu.edu" />
      </Members>
    </Group>
    <Group ID="0" Name="Anonymous" />
    <Group ID="70" Name="COLLECTION_hdl:123456789/57_ADMIN">
      <Members>
        <Member ID="1" Name="bsmith@myu.edu" />
      </Members>
    </Group>
    <Group ID="75" Name="COLLECTION_hdl:123456789/57_DEFAULT_READ">
      <MemberGroups>
        <MemberGroup ID="0" Name="Anonymous" />
      </MemberGroups>
    </Group>
    <Group ID="71" Name="COLLECTION_hdl:123456789/57_SUBMIT">
      <Members>
        <Member ID="1" Name="bsmith@myu.edu" />
      </Members>
    </Group>
    <Group ID="72" Name="COLLECTION_hdl:123456789/57_WORKFLOW_STEP_1">
      <MemberGroups>
        <MemberGroup ID="1" Name="Administrator" />
      </MemberGroups>
    </Group>
    <Group ID="73" Name="COLLECTION_hdl:123456789/57_WORKFLOW_STEP_2">
      <MemberGroups>
        <MemberGroup ID="1" Name="Administrator" />
      </MemberGroups>
    </Group>
    <Group ID="8" Name="COLLECTION_hdl:123456789/6703_DEFAULT_READ" />
    <Group ID="9" Name="COLLECTION_hdl:123456789/2_ADMIN">
      <Members>
        <Member ID="1" Name="bsmith@myu.edu" />
      </Members>
    </Group>
  </Groups>
  <People>
    <Person ID="1">
      <Email>bsmith@myu.edu</Email>
      <Netid>bsmith</Netid>
      <FirstName>Bob</FirstName>
      <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 https://github.com/DSpace/DSpace/issues/4492).*

The reasoning is that we were unable to translate an *Internal ID* into an *External ID* (i.e. Handle). If we are unable to do that translation, re-importing or restoring a group with an *old* internal ID could cause conflicts or instability in your DSpace system. In order to avoid such conflicts, these groups are renamed using a random, unique key.

Example of DSPACE-ROLES Schema for a Community or Collection

Below is a general example of the structure of a DSPACE-ROLES XML file, as it would appear in a Community or Collection AIP.

This specific example is for a Collection, which has associated Administrator, Submitter, and Workflow approver groups. In this very simple example, each group only has one Person as a member of it. Please notice that the Person's information (Name, NetID, etc) is NOT contained in this content (however they are available in the DSPACE-ROLES example for a SITE, as shown above)

```
<DSpaceRoles>
  <Groups>
    <Group ID="9" Name="COLLECTION_hdl:123456789/2_ADMIN" Type="ADMIN">
      <Members>
        <Member ID="1" Name="bsmith@myu.edu" />
      </Members>
    </Group>
    <Group ID="13" Name="COLLECTION_hdl:123456789/2_SUBMIT" Type="SUBMIT">
      <Members>
        <Member ID="2" Name="jjones@myu.edu" />
      </Members>
    </Group>
    <Group ID="10" Name="COLLECTION_hdl:123456789/2_WORKFLOW_STEP_1" Type="WORKFLOW_STEP_1">
      <Members>
        <Member ID="1" Name="bsmith@myu.edu" />
      </Members>
    </Group>
    <Group ID="11" Name="COLLECTION_hdl:123456789/2_WORKFLOW_STEP_2" Type="WORKFLOW_STEP_2">
      <Members>
        <Member ID="2" Name="jjones@myu.edu" />
      </Members>
    </Group>
    <Group ID="12" Name="COLLECTION_hdl:123456789/2_WORKFLOW_STEP_3" Type="WORKFLOW_STEP_3">
      <Members>
        <Member ID="1" Name="bsmith@myu.edu" />
      </Members>
    </Group>
  </Groups>
</DSpaceRoles>
```

METSRights Schema

All DSpace Policies (permissions on objects) are translated into the METSRights schema. This is different than the above DSPACE-ROLES schema, which only represents Groups and People objects. Instead, the METSRights schema is used to translate the permission statements (e.g. a group named "Library Admins" has Administrative permissions on a Community named "University Library"). But the METSRights schema doesn't represent who is a member of a particular group (that is defined in the DSPACE-ROLES schema, as described above).

METSRights should always be used with DSPACE-ROLES

The METSRights Schema must be used in conjunction with the DSPACE-ROLES Schema for Groups, People and Permissions to all be restored properly. As mentioned above, the METSRights metadata can only be used to restore permissions (i.e. DSpace policies). The DSPACE-ROLES metadata must also exist if you wish to restore the actual Group or EPeople objects to which those permissions apply.

All DSpace Object's AIPs (except for the SITE AIP) utilize the METSRights Schema in order to define what permissions people and groups have on that object. Although there are several sections to the METSRights Schema, DSpace AIPs *only use* the `<RightsDeclarationMD>` section, as this is what is used to describe rights on an object.

In the METS structure, METSRights metadata always appears within a `rightsMD` inside an `<mdWrap MDTYPE="OTHER" OTHERMDTYPE="METSRIGHTS">` element. For example:

```
  <amdSec ID="amd_2068">
      ...
      <rightsMD ID="rightsMD_2074">
         <mdWrap MDTYPE="OTHER" OTHERMDTYPE="METSRIGHTS">
              ...
         </mdWrap>
      </rightsMD>
      ...
  </amdSec>
```

By default, METSRights metadata is always included in AIPs. It is controlled by the following configuration in your `dspace.cfg`:

```
aip.disseminate.rightsMD = DSpaceDepositLicense:DSPACE_DEPLICENSE, \
    CreativeCommonsRDF:DSPACE_CCRDF, CreativeCommonsText:DSPACE_CCTEXT, METSRIGHTS
```

Example of METSRights Schema for an Item

An Item AIP will almost always contain several METSRights metadata sections within its METS Manifest. A separate METSRights metadata section is used to describe the permissions on:

- the Item itself
- each Bundle (group of files) in the Item
- each Bitstream (file) within an Item's bundle

Below is an example of a METSRights sections for a publicly visible Bitstream, Bundle or Item. Notice it specifies that the "GENERAL PUBLIC" has the permission to DISCOVER or DISPLAY this object.

```
<rights:RightsDeclarationMD xmlns:rights="http://cosimo.stanford.edu/sdr/metsrights/" RIGHTSCATEGORY="LICENSED">
  <rights:Context CONTEXTCLASS="GENERAL PUBLIC">
    <rights:Permissions DISCOVER="true" DISPLAY="true" MODIFY="false" DELETE="false" />
  </rights:Context>
</rights:RightsDeclarationMD>
```

As of DSpace 3, DSpace policies/permissions may also have a "start-date" or "end-date" (to support Embargo functionality). Such a policy on an Item may look like this. Notice it specifies that the "GENERAL PUBLIC" has the permission to DISCOVER or DISPLAY this object *starting on* 2015-01-01, while the Group "Staff" has permission to DISCOVER or DISPLAY this object *until* 2015-01-01.

```
<rights:RightsDeclarationMD xmlns:rights="http://cosimo.stanford.edu/sdr/metsrights/" RIGHTSCATEGORY="LICENSED">
  <rights:Context CONTEXTCLASS="GENERAL PUBLIC" start-date="2015-01-01" in-effect="false">
    <rights:Permissions DISCOVER="true" DISPLAY="true" MODIFY="false" DELETE="false" />
  </rights:Context>
  <rights:Context CONTEXTCLASS="MANAGED_GRP" end-date="2015-01-01" in-effect="true">
    <rights:UserName USERTYPE="GROUP">Staff</rights:UserName>
    <rights:Permissions DISCOVER="true" DISPLAY="true" MODIFY="false" DELETE="false" />
  </rights:Context>
</rights:RightsDeclarationMD>
```

Example of METSRights Schema for a Collection

A Collection AIP contains one METSRights section, which describes the permissions different Groups or People have within the Collection

Below is an example of a METSRights sections for a publicly visible Collection, which also has an Administrator group, a Submitter group, and a group for each of the three DSpace workflow approval steps. You'll notice that each of the groups is provided with very specific permissions within the Collection. Submitters & Workflow approvers can "ADD CONTENTS" to a collection (but cannot delete the collection). Administrators have full rights.

```
<rights:RightsDeclarationMD xmlns:rights="http://cosimo.stanford.edu/sdr/metsrights/" RIGHTSCATEGORY="LICENSED">
  <rights:Context CONTEXTCLASS="MANAGED_GRP">
    <rights:UserName USERTYPE="GROUP">COLLECTION_hdl:123456789/2_SUBMIT</rights:UserName>
    <rights:Permissions DISCOVER="true" DISPLAY="true" MODIFY="true" DELETE="false" OTHER="true"
```

```
OTHERPERMITTYPE="ADD CONTENTS" />
  </rights:Context>
  <rights:Context CONTEXTCLASS="MANAGED_GRP">
    <rights:UserName USERTYPE="GROUP">COLLECTION_hdl:123456789/2_WORKFLOW_STEP_3</rights:UserName>
    <rights:Permissions DISCOVER="true" DISPLAY="true" MODIFY="true" DELETE="false" OTHER="true"
OTHERPERMITTYPE="ADD CONTENTS" />
  </rights:Context>
  <rights:Context CONTEXTCLASS="MANAGED_GRP">
    <rights:UserName USERTYPE="GROUP">COLLECTION_hdl:123456789/2_WORKFLOW_STEP_2</rights:UserName>
    <rights:Permissions DISCOVER="true" DISPLAY="true" MODIFY="true" DELETE="false" OTHER="true"
OTHERPERMITTYPE="ADD CONTENTS" />
  </rights:Context>
  <rights:Context CONTEXTCLASS="MANAGED_GRP">
    <rights:UserName USERTYPE="GROUP">COLLECTION_hdl:123456789/2_WORKFLOW_STEP_1</rights:UserName>
    <rights:Permissions DISCOVER="true" DISPLAY="true" MODIFY="true" DELETE="false" OTHER="true"
OTHERPERMITTYPE="ADD CONTENTS" />
  </rights:Context>
  <rights:Context CONTEXTCLASS="MANAGED_GRP">
    <rights:UserName USERTYPE="GROUP">COLLECTION_hdl:123456789/2_ADMIN</rights:UserName>
    <rights:Permissions DISCOVER="true" DISPLAY="true" COPY="true" DUPLICATE="true" MODIFY="true" DELETE="true"
PRINT="true" OTHER="true" OTHERPERMITTYPE="ADMIN" />
  </rights:Context>
  <rights:Context CONTEXTCLASS="GENERAL PUBLIC">
    <rights:Permissions DISCOVER="true" DISPLAY="true" MODIFY="false" DELETE="false" />
  </rights:Context>
</rights:RightsDeclarationMD>
```

Example of METSRights Schema for a Community

A Community AIP contains one METSRights section, which describes the permissions different Groups or People have within that Community.

Below is an example of a METSRights sections for a publicly visible Community, which also has an Administrator group. As you'll notice, this content looks very similar to the Collection METSRights section (as described above)

```
<rights:RightsDeclarationMD xmlns:rights="http://cosimo.stanford.edu/sdr/metsrights/" RIGHTSCATEGORY="LICENSED">
  <rights:Context CONTEXTCLASS="MANAGED_GRP">
    <rights:UserName USERTYPE="GROUP">COMMUNITY_hdl:123456789/10_ADMIN</rights:UserName>
    <rights:Permissions DISCOVER="true" DISPLAY="true" COPY="true" DUPLICATE="true" MODIFY="true" DELETE="true"
PRINT="true" OTHER="true" OTHERPERMITTYPE="ADMIN" />
  </rights:Context>
  <rights:Context CONTEXTCLASS="GENERAL PUBLIC">
    <rights:Permissions DISCOVER="true" DISPLAY="true" MODIFY="false" DELETE="false" />
  </rights:Context>
</rights:RightsDeclarationMD>
```

Ant targets and options

1 Options
2 Targets

Ant targets should be run as the service user

A word of warning: in order to ensure proper permissions and file ownership are maintained, you are advised to run these ant targets as the service user (commonly 'dspace' or 'tomcat'). Running them as any other user may cause permission problems

Options

DSpace allows three property values to be set using the -D<property>=<value> option. They may be used in other contexts than noted below, but take care to understand how a particular property will affect a target's outcome.

overwrite

Whether to overwrite configuration files in [dspace]/config. If true, files from [dspace]/config and subdirectories are backed up with .old extension and new files are installed from [dspace-src]/dspace/config and subdirectories; if false, existing config files are untouched, and new files are written beside them with .new extension.	
Possible values:	true, false
Default:	true
Context:	update, init_configs

config

If a path is specified, ant uses values from the specified file and installs it in [dspace]/config in the appropriate contexts.	
Possible values:	path to configuration file to be used
Default:	[dspace-src]/config/dspace.cfg
Context:	update, update_configs, update_code, update_webapps, init_configs, fresh_install, test_database, setup_database, load_registries, clean_database

wars

If true, builds .war files; if false, no .war files are built.	
Possible values:	true, false
Default:	true
Context:	update, update_webapps, fresh_install

Targets

Target	Effect
update	Creates backup copies of the [dspace]/bin, /etc, /lib, and /webapps directories with the form /<directory>.bak-<date-time>. Creates new copies of [dspace]/config, /etc, and /lib directories. Does not affect data files or the database. (See *overwrite*, *config*, *war* options.)
update_configs	Updates the [dspace]/config directory with new configuration files. (See *config* option.)
update_code	Creates backup copies of the [dspace]/bin, /etc, and /lib directories with the form /<directory>.bak-<date-time>. Creates new copies of [dspace]/config, /etc, and /lib directories. (See *config* option.)
update_webapps	Updates [dspace]/webapps directory. (See *config*, *war* options.)
init_configs	Writes configuration files to [dspace]/config. (See *overwrite*, *config* options.)

install_code	Deletes existing [dspace]/bin, /lib, and /etc directories, and installs new copies; overwrites /solr application files, leaving data intact. (See *config* option.)
fresh_install	Performs a fresh installation of the software, including the database & config. (See *config*, *war* options.)
test_database	Tests database connection using parameters specified in dspace.cfg. (See *config* option.)
clean_backups	Removes [dspace]/bin, /etc, /lib, and /webapps directories with .bak* extensions.

Command Line Operations

1 Executing command line operations
2 Available operations
    2.1 General use
    2.2 Legacy statistics
    2.3 SOLR Statistics

The DSpace command launcher or CLI interface offers the execution of different maintenance operations. As most of these are already documented in related parts of the documentation, this page is mainly intended to provide an overview of all available CLI operations, with links to the appropriate documentation.

Executing command line operations

The CLI interface is found at `[dspace]/bin/dspace`. Execute it without arguments or with the `-h` option to see all available operations. Execute `dspace` *op* `-h` to see details about the *op* operation.

Examples:

bin/dspace -h
bin/dspace cleanup -h
bin/dspace cleanup
bin/dspace cleanup --verbose

Available operations

Some operations can also be run as "Processes" (or Scripts) from the administrative User Interface or REST API. Those Scripts have a detailed description in our REST API documentation at https://github.com/DSpace/RestContract/blob/main/scripts/

General use

- bitstore-migrate: Migrate all files (bitstreams) from one assetstore (bitstore) to another
- checker: Run the checksum checker
- checker-emailer: Send emails related to the checksum checker
- classpath: Calculate and display the DSpace classpath
- cleanup: Remove deleted bitstreams from the assetstore
- community-filiator: Tool to manage community and sub-community relationships
- create-administrator: Create a DSpace administrator account (see Installing DSpace)
- curate: Perform curation tasks on DSpace objects
- database: Perform various tasks / checks of the DSpace database
- doi-migrate: Convert DOI metadata from old fields and formats to a new one
- doi-organiser: Transmit information about DOIs to the registration agency.
- dsprop: View the value of a DSpace property from any configuration file (see Configuration Reference)
- dsrun: Run a (DSpace) Java class directly (used mainly for test purposes)
- embargo-lifter: Pre DSpace 3.0 embargo manager tool used to check, list and lift embargoes
- export: Export items or collections
- filter-media: Perform the media filtering to extract full text from documents and to create thumbnails
- generate-sitemaps: Generate search engine and html sitemaps (see Search Engine Optimization)
- harvest: Manage the OAI-PMH harvesting of external collections (see OAI harvesting docs)
- import: Import items into DSpace (see Importing and Exporting Items via Simple Archive Format)
- index-authority: import authorities and keep SOLR authority index up to date
- index-discovery: Update Discovery (Solr) search and browse Index
- itemupdate: Item update tool for altering metadata and bitstream content in items (see Updating Items via Simple Archive Format)
- make-handle-config: Run the handle server simple setup command
- metadata-export: Export metadata for batch editing
- metadata-import: Import metadata after batch editing
- migrate-embargo: Embargo manager tool used to migrate old version of Embargo to the new one included in dspace3
- oai: OAI script manager
- packager: Execute a packager
- process-cleaner: Delete old Processes from the system
- rdfizer: tool to convert contents to RDF
- read : execute a stream of commands from a file or pipe
- registry-loader: Load entries into a registry (see Metadata and Bitstream Format Registries)
- structure-builder: Build DSpace community and collection structure (see Exporting and Importing Community and Collection Hierarchy)
- sub-daily: Send daily subscription notices
- test-email: Test the DSpace email server settings are OK
- update-handle-prefix: Update handle records and metadata when moving from one Handle prefix to another
- user: Create, List, Update, Delete EPerson (user) records
- validate-date: Test date-time format rules
- version: Show DSpace version and other troubleshooting information

Legacy statistics

DSpace 7.x or later does not yet support

Legacy/log based statistics are not available in DSpace 7.x or later releases. They are under discussion as this feature is not widely used. Tentatively, they are scheduled for possible removal. See https://github.com/DSpace/DSpace/issues/2852

Legacy statistics parse the DSpace log files and compile information based on the "[dspace]/config/dstat.cfg" configuration file. They are no longer actively maintained, but still exist in the codebase because there is information they report on that is not yet accessible in (or replaced by) SOLR Statistics. Where possible, we recommend using SOLR Statistics and/or Google Analytics for more accurate data.

- stat-general: Compile the general statistics
- stat-initial: Compile the initial statistics
- stat-monthly: Compile the monthly statistics
- stat-report-general: Create the general statistics report
- stat-report-initial: Create the initial statistics report
- stat-report-monthly: Create the monthly statistics report

SOLR Statistics

Scripts for the statistics that are stored in SOLR:

- solr-export-statistics:Export Solr statistics data to CSV (for backup or moving to another server)
- solr-import-statistics: Import Solr statistics data from CSV (for restoration, or moving to another server)
- solr-reindex-statistics: Reindex Solr statistics data (for upgrades or updates to Solr schema)
- stats-log-converter: Convert dspace.log files ready for import into solr statistics
- stats-log-importer: Import previously converted log files into solr statistics
- stats-util: Statistics Client for Maintenance of Solr Statistics Indexes

Executing streams of commands

You can pass a sequence of commands into the `dspace` command-line tool using the `read` command.

Execute commands...	this way
...from a file	`[dspace]/bin/dspace read a-command-file`
...in a pipeline	`some-other-command | [dspace]/bin/dspace read -` `some-other-command | [dspace]/bin/dspace read`

Database Utilities

1 "database" command
2 Guide to Flyway Migration States

"database" command

This command can be used at any time to manage or upgrade the Database. It will also assist in troubleshooting PostgreSQL and Oracle connection issues with the database.

Command used:	`[dspace]/bin/dspace database`
Java class:	org.dspace.storage.rdbms.DatabaseUtils
Valid Arguments:	**Description**
`test`	Test the database connection settings (in `[dspace]/config/dspace.cfg` or `local.cfg`) are OK and working properly. This command also validates the database version is compatible with DSpace.
`info`	Provide detailed information about the DSpace database itself. This includes the database type, version, driver, schema, and any successful/failed/pending database migrations. This command, along with "test", is very useful in debugging issues with your database.
`migrate`	Migrate the database to the latest version (if not already on the latest version). This uses FlywayDB along with embedded migrations scripts to automatically update your database to the latest version. Additional 'migrate' options: • "`migrate ignored`" will run a migration which *also includes* any database migrations which are flagged as "Ignored" (or "Skipped") by the "info" command. If these "Ignored" migrations succeed, they will now be noted (in the "info" command) as having run "Out Of Order" (i.e. they were successful, but they were executed out of the normal, numerical order). • "`migrate force`" (*available in 7.1 and later*) will run a migration *even when no new migrations exist* (i.e. no migrations are currently flagged as "Pending" when using the "info" command). This can be used to force the post-migration ("callback") scripts to run. Normally, these post-migration scripts only run after a new migrations are applied. They will (re-)initialize your database with required objects, like the "Site" object, default groups (Administrator/Anonymous) and default metadata registry and bitstream format registry entries.
`repair`	Attempt to "repair" any migrations which are flagged as "Failed" by the "info" command and/or resolve failed checksum validation. This runs the FlywayDB repair command. Please note however, this will NOT automatically repair corrupt or broken data in your database. It merely tries to re-run previously "Failed" migrations and/or realign the checksums of the applied migrations to the ones of the available migrations.
`skip`	*(Available in 7.5 and later)* Allows you to "skip" individual database migrations. Skipping a migration will flag it as having run successfully (either "Success" or "Out of Order" status), but the migration *will not be executed.* *WARNING: You should ONLY skip migrations which are no longer required or have become obsolete.* Skipping a REQUIRED migration may result in DSpace failing to startup or function properly. The only fix to that scenario would be to run the migration manually (by executing the SQL directly on the database). Therefore, this "skip" command should ONLY be used when the migration is known to be obsolete or no longer valid. All other usages are unsupported.
`update-sequences`	Update database sequences after running a bulk ingest (e.g. AIP Backup and Restore) or data migration.
`validate`	Validate the checksums of all previously run database migrations. This runs the FlywayDB 'validate' command.
`clean`	Completely and permanently delete all tables and data in this database. WARNING: There is no turning back! If you run this command, you will lose your entire database and all its contents. This command is only useful for testing or for reverting your database to a "fresh install" state (e.g. running "`dspace database clean`" followed by "`dspace database migrate`" will return your database to a fresh install state) By default the 'clean' command is **disabled** (to avoid accidental data loss). In order to enable it, you must first set `db.cleanDisabled=false` in either your local.cfg or dspace.cfg.

Guide to Flyway Migration States

Whenever you run the "info" or "status" option (e.g. ./dspace database info), you'll see a table listing all Flyway migrations which were run on your database.

In the "State" column of that table you'll see the various states that Flyway returns for each migration:

- "Baseline" - This state simply notes when your database was originally created. In other words, it's the **first** migration in the table
- "Success" - The state means that the migration was successful. It completed without any errors.
- "Out of Order" - This state means the migration was successful *but was run in a different order than the default.* By default, Flyway will run all migrations in numerical order (based on the "Version" column in that table). **These "Out of Order" entries are normal and will occur in almost every production DSpace.** These entries are a sign of *backported bug fixes* which required a database migration. Those backported fixes may be required to run in *non-numerical order.*
    - For example: If we found a database issue that impacts both 7.x and 8.x, we may create a migration numbered "7.9.[date]" which is *safe to run on both 7.x and 8.x.* If your site is already on 8.x, this migration would be flagged as "Out of Order" because it starts with "7.9.*" and was run **after** other "8.0.*" migrations were already run.
- "Ignored" - This state means the migration was **skipped** (because it's not in numerical order) and *has not yet been run.* To run it, you'd run ". /dspace database migrate ignored". Ignored migrations will move into the "Out of Order" state if they are successfully run.
- "Pending" - This state means the migration *has not yet been run*. It's waiting for the next time you call "./dspace database migrate".

Miscellaneous Commands

- test-email
- registry-loader
  - Usage
  - Parameters
  - Examples

test-email

Sends an email message to the administrator address (configuration property `mail.admin`). This is useful for testing basic email sending as well as the templates for various specific email bodies.

With no arguments, `dspace test-email` sends a simple test message.

With arguments, the command composes a message using the template named in the first argument, with message parameters given by subsequent arguments:

```
bin/dspace config/emails/test-email export_error "An error" https://dspace.myu.edu/feedback/
```

registry-loader

This command can be used to load bitstream format or metadata registries into the database.

See also Metadata and Bitstream Format Registries

Usage

```
[dspace]/bin/dspace registry-loader [-b <arg>] [-h] [-m <arg>]
```

Parameters

-h,--help	print help message
-b,--bitstream <arg>	load bitstream format registry from specified file
-m,--metadata <arg>	load metadata registry from specified file

Examples

```
# Load Bitstream metadata
[dspace]/bin/dspace registry-loader -b [dspace]/config/registries/bitstream-formats.xml

# Load metadata registry for 'local' namespace
[dspace]/bin/dspace registry-loader -m [dspace]/config/registries/local-types.xml
```

Handle.Net Registry Support

- 1 To install your Handle resolver on the host where DSpace runs
- 2 To install a Handle resolver on a separate machine
- 3 To install a Handle resolver on a separate machine using template handles
- 4 Updating Existing Handle Prefixes

DSpace comes with support for CNRI's Handle.Net Registry (HNR). This feature is *completely optional*, as DSpace functions the same with or without using a Handle Server/Registry.

A few things to keep in mind:

- You'll notice that while you've been playing around with a test server, DSpace has apparently been creating (fake) 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. One is included with DSpace.
- If you want to use the Handle system, you'll need to obtain a Handle prefix from the central CNRI Handle site. This requires a small yearly fee to CNRI
- Again, all of this is **completely optional.** But, the key benefit is that it provides you with persistent, permanent URLs (of the form `https://hdl.handle.net/[prefix]/[suffix]`) for every object within your DSpace site. Those persistent URLs may be useful for citations or even during upgrades/migrations, as DSpace + Handle.Net ensures that these URLs always go to the right object, even if your site's main URL changes.

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

You can either use a Handle server running on the same machine as DSpace, or you can install it on a separate machine. Installing it on the same machine is a little bit easier. If you install it on a separate machine, you can use one Handle server for more than one DSpace installation.

- To install your Handle resolver on the host where DSpace runs
- To install a Handle resolver on a separate machine
- To install a Handle resolver on a separate machine using template handles
- Updating Existing Handle Prefixes

To install your Handle resolver on the host where DSpace runs

We recommend configuring your Handle server **without a passphrase**, as the current DSpace `start-handle-server` scripts do not yet support startup with a passphrase.

If you choose to set a passphrase, you may need to start the Handle Server via: `[dspace]\bin\dspace dsrun net.handle.server.Main [dspace]\handle-server`

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

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

   a. If you are using Windows, the proper command is:

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

      Ensure that *[dspace]/handle-server* matches whatever you have in *dspace.cfg* for the *handle.dir* property. You will need to answer a series of qestions to configure the server. For the most part, you can use the default options, except you should choose to **not** encrypt your certificates when prompted.
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"
   "enable_txn_queue" = "no"
   ```

   This tells the Handle server to get information about individual Handles from the DSpace code and to disable transaction replication. If you used the make-handle-config script, these should already be set in your config.dct file.
3. Once the configuration file has been generated, you will need to go to https://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/123456789"*. Replace 123456789 with the assigned naming authority prefix sent to you. Also change the value of `handle.prefix` in `[dspace]/config/local.cfg` from "123456789" to your assigned naming authority prefix, so that DSpace will use that prefix in assigning new Handles.
5. Now start your handle server (as the dspace user):

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

   a. If you are using Windows, there is a corresponding 'start-handle-server.bat' script:

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

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.

To install a Handle resolver on a separate machine

This works with DSpace 7.4 and later.

The Handle server you use must be dedicated to resolve Handles from DSpace. You cannot use a Handle server that is in use with other software already. You can use CNRI's Handle Software -- all you have to do is to add to it a plugin that is provided by DSpace. The following instructions were tested with CNRI's Handle software version 9.1.0. You can do the following steps on another machine than the machine DSpace runs on, but you have to copy some files from the machine on which DSpace is installed.

1. Set the following two configuration properties for every DSpace backend that your are running:

DSpace backend configuration to activate the endpoints used by the remote handle resolver

```
handle.remote-resolver.enabled = true
handle.hide.listhandles = false
```

2. Download the CNRI Handle Software: http:s//www.handle.net/download.html. In the tarball you'll find an `README.txt` with installation instructions -- follow it.
3. After installing the CNRI Handle Software you should have two directories: once that contains the CNRI software and one that contains the configuration of you local Handle Server. For the rest of this instruction we assume that the directory containing the CNRI Software is `/hs/handle-9.1.0` and the directory containing the configuration of your local server is `/hs/srv_1`. (We use the same paths here as CNRIs README.txt.)
4. Download the plugin from https://github.com/DSpace/Remote-Handle-Resolver/releases. Select a release. You can get the source and build it yourself, or just use the JAR file included in the release. In either case, once you have a `dspace-remote-handle-resolver-VERSION.jar`, copy it to the directory containing the CNRI software (`/hs/handle-9.1.0/lib`).
5. Create the directory `/hs/srv_1/logs`.
6. Create the following two files in `/hs/srv_1`.

log4j-handle-plugin.properties

```
log4j.rootCategory=INFO, A1
log4j.appender.A1=org.apache.log4j.DailyRollingFileAppender
log4j.appender.A1.File=/hs/srv_1/logs/handle-plugin.log
log4j.appender.A1.DatePattern= '.' yyyy-MM-dd
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%d %-5p %c @ %m%n
log4j.logger.org.apache.axis.handlers.http.HTTPAuthHandler=INFO
```

Change the path in the third line, if necessary. It must point to the DSpace 7 Rest API (as configured in $dspace.server.url).

handle-dspace-plugin.cfg

```
dspace.handle.endpoint1 = http: //dspace.example.org/server
```

If you run more than one DSpace Installation, you may add more DSpace Endpoints. Just increase the number at the end of the key for each: `endpoint2`, `endpoint3`....

7. Edit the file `/hs/srv_1/config.dct` to include the following lines in the `"server_config"` clause:

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

8. Edit `/hs/handle-9.1.0/bin/hdl`:
    a. Find a line that contains `exec java ... net.handle.server.Main ...`
    b. Add "`-Dlog4j.configuration=file:///hs/srv_1/log4j-handle-plugin.properties -Ddspace.handle.plugin.configuration=/hs/srv_1/handle-dspace-plugin.cfg`" right in front of `net.handle.server.Main`.
9. If your handle server is running, restart it.

Please note: The Handle Server will only start if it is able to connect to at least one running DSpace Installation. It only resolves the handles of the DSpace Installations that were running when it was started.

To install a Handle resolver on a separate machine using template handles

Instead of using the described plugin above, you can configure a Handle server (version 8+) to resolve handles based on a template. Template handle require less configuration than the plugin, and do not require an additional download. However, there are two things to keep in mind when using template handles:

1. Handles that don't exist will still generate a Handle record with a URL, even though resolving that URL will show an error page.
2. Handle records can only be generated based on the handle and the template. If you need to look up information in DSpace in or to geneate the correct url for a given handle, you will need to use a storage plugin instead.

The Handle server you use must be dedicated to resolve Handles from DSpace. You cannot use a Handle server that is in use with other software already. The following instructions were tested with CNRI's Handle software version 9.1.0.

1. Download the CNRI Handle Software: https://www.handle.net/download.html.
2. In the tarball you'll find an `README.txt` with installation instructions. Follow the directions to install and configure your Handle server. Importantly, make sure your prefixes are set correctly in the "auto_homed_prefixes" setting.
3. Edit `the server's config.dct` file to include the following line in the "`server_config`" clause:

```
"namespace" = "<namespace><template delimiter'/'><value type='URL' index='1' data='https://demo.dspace.org/handle/${handle}'/></template></namespace>"
```

    In the "namespace" section, replace "`https://demo.dspace.org/handle/`" with the url endpoint for your DSpace server. The "${handle}" part of the template will be replaced with the full handle to be resolved.
4. If your handle server is running, restart it.

This configuration is a minimal example of how to configure template handles for DSpace. For more details about configuring template handles, see the Handle Technical Manual, Chapter 11 (PDF download).

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.

Logical Item Filtering and DOI Filtered Provider for DSpace

- Section One: DSpace Logical Item filtering (org.dspace.content.logic.*)
  - LogicalStatement
  - Filters
  - Operators
  - Conditions
  - Configuring Filters in Spring
  - Running Tests on the Command Line
  - Using Filters in other Spring Services
- Section Two: DOI Filtered Provider
  - New FilteredProvider: DOIIdentifierProvider
  - Skip the filter

Section One: DSpace Logical Item filtering (org.dspace.content.logic.*)

Inspired by the powerful conditional filters in XOAI, this component offers a simple but flexible way to write logical statements and tests, and use the results of those tests in other services or DSpace code.

LogicalStatement

LogicalStatement is a simple interface ultimately implemented by all the other interfaces and classes described below. It just requires that a class implements a `Boolean getResult(context, item)` method.

Filters

Filters are at the root of any test definition, and it is the filter ID that is used to load up the filter in spring configurations for other services, or with DSpace Service Manager.

A filter bean is defined with a single “statement” property - this could be an Operator, to begin a longer logical statement, or a Condition, to perform a simple check.

There is one simple implementation of Filter included - DefaultFilter.

Operators

Operators are the basic logical building blocks that implement operations like AND, OR, NOT, NAND and NOR. An Operator can contain any number of other Operators or Conditions.

So statements like this can be created:

(**x** AND (**y** OR **z**) AND **a** AND (**b** OR NOT(**d**))

Conditions

Conditions are where the actual DSpace item evaluation code is written. A condition accepts a Map<String, Object> map of parameters. Conditions don’t contain any other LogicalStatement classes – the are at the bottom of the chain.

A condition could be something like MetadataValueMatchCondition, where a regex pattern and field name are passed as parameters, then tested against actual item metadata. If the regex matches, the boolean result is true.

Typically, commonly used Conditions will be defined as beans elsewhere in the spring config and then referenced inside Filters and Operators to create more complex statements.

Configuring Filters in Spring

Conditions, Operators and Filters are all defined in `${dspace}/config/spring/api/item-filters.xml`

Here’s a complete example of a filter definition that implements the same rules as the XOAI openAireFilter. As an exercise, some statements will be defined as beans externally, and some will be defined inline as part of the filter.

New Condition: driver-document-type_condition

This condition creates a new bean to test metadata values. In this case, we’re implementing “ends with” for a list of type patterns.

```
<!-- dc.type ends with any of the listed values, as per XOAI "driverDocumentTypeCondition" -->
    <bean id="driver-document-type_condition"
          class="org.dspace.content.logic.condition.MetadataValuesMatchCondition">
        <property name="parameters">
            <map>
                <entry key="field" value="dc.type" />
                <entry key="patterns">
                    <list>
                        <value>article$</value>
                        <value>bachelorThesis$</value>
                        <value>masterThesis$</value>
                        <value>doctoralThesis$</value>
                        <value>book$</value>
                        <value>bookPart$</value>
                        <value>review$</value>
                        <value>conferenceObject$</value>
                        <value>lecture$</value>
                        <value>workingPaper$</value>
                        <value>preprint$</value>
                        <value>report$</value>
                        <value>annotation$</value>
                        <value>contributionToPeriodical$</value>
                        <value>patent$</value>
                        <value>dataset$</value>
                        <value>other$</value>
                    </list>
                </entry>
            </map>
        </property>
    </bean>
```

New Condition: item-is-public_condition

This condition accepts group and action parameters, then inspects item policies for a match - if the supplied group can perform the action, the result is true.

```
<bean id="item-is-public_condition"
          class="org.dspace.content.logic.condition.ReadableByGroupCondition">
        <property name="parameters">
            <map>
                <entry key="group" value="Anonymous" />
                <entry key="action" value="READ" />
            </map>
        </property>
</bean>
```

New Filter: openaire_filter

Here is the full definition for the OpenAIRE filter.

The first statement is an And Operator, with many sub-statements – four Conditions, and an Or statement.

The first two statements in this Operator are simple Conditions defined in-line, and just check for a non-empty value in a couple of metadata fields.

The third statement is a reference to the document type Condition we made earlier:
`<ref bean="driver-document-type_condition" />`

The fourth statement is another Operator, in this case an Or Operator with two Conditions (the is-public Condition we defined earlier, and an in-line definition of as “is-withdrawn” Condition)

The fifth statement is an in-line definition of a Condition that checks dc.relation metadata for a valid OpenAIRE identifier.

So the full logic implemented is:

`(has-title AND has-author AND has-driver-type AND (is-public OR is-withdrawn) AND has-valid-relation)`

```
<!-- An example of an OpenAIRE compliance filter based on the same rules in xoai.xml
      some sub-statements are defined within this bean, and some are referenced from earlier definitions
-->
<bean id="openaire_filter" class="org.dspace.content.logic.DefaultFilter">
    <property name="statement">
        <bean class="org.dspace.content.logic.operator.And">
            <property name="statements">
                <list>
                    <!-- Has a non-empty title -->
                    <bean id="has-title_condition"
                          class="org.dspace.content.logic.condition.MetadataValueMatchCondition">
                        <property name="parameters">
                            <map>
                                <entry key="field" value="dc.title" />
                                <entry key="pattern" value=".*" />
                            </map>
                        </property>
                    </bean>
                    <!-- AND has a non-empty author -->
                    <bean id="has-author_condition"
                          class="org.dspace.content.logic.condition.MetadataValueMatchCondition">
                        <property name="parameters">
                            <map>
                                <entry key="field" value="dc.contributor.author" />
                                <entry key="pattern" value=".*" />
                            </map>
                        </property>
                    </bean>
                    <!-- AND has a valid DRIVER document type (defined earlier) -->
                    <ref bean="driver-document-type_condition" />
                    <!-- AND (the item is publicly accessible OR withdrawn) -->
                    <bean class="org.dspace.content.logic.operator.Or">
                        <property name="statements">
                            <list>
                                <!-- item is public, defined earlier -->
                                <ref bean="item-is-public_condition" />
                                <!-- OR item is withdrawn, for tombstoning -->
                                <bean class="org.dspace.content.logic.condition.IsWithdrawnCondition">
                                    <property name="parameters"><map></map></property>
                                </bean>
                            </list>
                        </property>
                    </bean>
                    <!-- AND the dc.relation is a valid OpenAIRE identifier
                          (starts with "info:eu-repo/grantAgreement/") -->
                    <bean id="has-openaire-relation_condition"
                          class="org.dspace.content.logic.condition.MetadataValueMatchCondition">
                        <property name="parameters">
                            <map>
                                <entry key="field" value="dc.relation" />
                                <entry key="pattern" value="^info:eu-repo/grantAgreement/" />
                            </map>
                        </property>
                    </bean>
                </list>
            </property>
        </bean>
    </property>
</bean>
```

Running Tests on the Command Line

There is a launcher command that can arbitrarily run tests on an item or all items, eg.

`${dspace}/bin/dspace dsrun org.dspace.content.logic.TestLogicRunner -f openaire_filter -i 123456789/100`

A simple `true` or `false` is printed for each item tested.

Using Filters in other Spring Services

The Filter beans can be referenced (or defined) in other services, for instance, here is adding the bean we configured earlier, as a `filterService` to a new `FilteredDOIIdentifierProvider`:

```
<bean id="org.dspace.identifier.DOIIdentifierProvider"
      class="org.dspace.identifier.FilteredDOIIdentifierProvider"
      scope="singleton">
    <property name="configurationService"
              ref="org.dspace.services.ConfigurationService" />
    <property name="DOIConnector"
              ref="org.dspace.identifier.doi.DOIConnector" />
    <property name="filterService"
              ref="openaire_filter"/>
</bean>
```

In the provider, we just define the property with the other services and class variables:

```
private Filter filterService;
```

And make sure there is a setter for it:

```
@Required
public void setFilterService(Filter filterService) {
    this.filterService = filterService;
}
```

Then you can actually run the tests with the service, like this:

```
try {
    Boolean result = filterService.getResult(context, (Item) dso);
    // do something with result
} catch(LogicalStatementException e) {
    // ... handle exception ...
}
```

In the TestLogicRunner, you can see a way to get the filters by name using the DSpaceServiceManager as well.

Section Two: DOI Filtered Provider

New FilteredProvider: DOIIdentifierProvider

DOIIdentifierProvider now extends a base FilteredIdentifierProvider, which looks for any configured filters and only allows minting DOIs for items where the filter returns **true**

This filter is always applied to the DOI consumer and other internal DOI service calls, and is applied by default to the `doi-organiser` tool (though it can be optionally skipped with a command-line argument)

The filter is a spring property configured in identifier-service.xml, in the provider bean declaration.

The filterService property is *optional*. If it is missing from spring configuration, all items will get DOIs minted as per normal and the provider's filter service will be null.

It is defined as follows:

```
<bean id="org.dspace.identifier.DOIIdentifierProvider"
    class="org.dspace.identifier.FilteredDOIIdentifierProvider"
    scope="singleton"> <property name="configurationService"
    ref="org.dspace.services.ConfigurationService" />
    <property name="DOIConnector" ref="org.dspace.identifier.doi.DOIConnector" />
    <property name="filterService" ref="openaire_filter"/>
</bean>
```

Where the "openaire_filter" reference is the ID of a filter bean defined in item-filters.xml

Skip the filter

In Edite Item administrators have a button to assign DOIs to item that don't have any yet. This skips the filters, as we assume administrators to know what they are doing. Since DSpace 7 there is a curation task that register DOIs for any item that does not have any. In the configuration of that curation task, you can specify whether filters should be skipped or respected. The curation task itself is configured in ${dspace}/config/modules/curate.cfg as 'registerdoi' with the label "Register DOI". There is a configuration file in ${dspace}/config/modules/doi-curation.cfg that can be used to customise the behaviour regarding filter skipping, and distribution over multiple items.

```
### DOI registration curation task configuration module


##
# Should any logical filters be skipped when registering DOIs? (ie. *always* register, never filter out the item)
# Default: true
#doi-curation.skip-filter = true


##
# Should we allow the curation task to be distributed over communities / collections of items or the whole
repository?
# This *could* be dangerous if run accidentally over more items than intended.
# Default: false
#doi-curation.distributed = false
```

Mediafilters for Transforming DSpace Content

1 MediaFilters: Transforming DSpace Content
- 1.1 Overview
- 1.2 Available Media Filters
- 1.3 Enabling/Disabling MediaFilters
- 1.4 Executing (via Command Line)
- 1.5 Creating Custom MediaFilters
  - 1.5.1 Creating a simple Media Filter
  - 1.5.2 Creating a Dynamic or "Self-Named" Format Filter
- 1.6 Configuration parameters

MediaFilters: Transforming DSpace Content

Overview

DSpace can apply filters or transformations to files/bitstreams, creating new content. Filters are included that extract text for **full-text searching**, and create **thumbnails** for items that contain images. The media filters are controlled by the `dspace filter-media` script which traverses the asset store, invoking all configured `MediaFilter` or `FormatFilter` classes on files/bitstreams (see Configuring Media Filters for more information on how they are configured).

Available Media Filters

Below is a listing of all currently available Media Filters, and what they actually do:

Name	Java Class	Function	Default input formats	Enabled by Default?
Text Extractor (7.3 or above)	`org.dspace.app.mediafilter.TikaTextExtractionFilter`	As of 7.3, all text extraction for Full text indexing takes place in a single filter. This filter uses Apache Tika which supports a wide variety of formats (e.g. Microsoft products, PDF, HTML, Text, etc). Additional formats may be configured from the Tika supported formats list at https://tika.apache.org/2.3.0/formats.html	Adobe PDF, Microsoft formats (Word, PPT, Excel), CSV, HTML, RTF, Text, OpenDocument formats (Text, Spreadsheet, Presentation)	yes
PDF Text Extractor (7.2 or below)	`org.dspace.app.mediafilter.PDFFilter`	extracts the full text of Adobe PDF documents (only if text-based or OCRed) for full text indexing. (Uses the Apache PDFBox tool)	Adobe PDF	yes
HTML Text Extractor (7.2 or below)	`org.dspace.app.mediafilter.HTMLFilter`	extracts the full text of HTML documents for full text indexing. (Uses Swing's HTML Parser)	HTML, Text	yes
Word Text Extractor (7.2 or below)	`org.dspace.app.mediafilter.PoiWordFilter`	extracts the full text of Microsoft Word and Microsoft Word XML documents for full text indexing. (Uses the "Apache POI" tools.)	Microsoft Word, Microsoft Word XML	yes
Excel Text Extractor (7.2 or below)	`org.dspace.app.mediafilter.ExcelFilter`	extracts the full text of Microsoft Excel documents for full text indexing. (Uses the "Apache POI" tools.)	Microsoft Excel, Microsoft Excel XML	yes
PowerPoint Text Extractor (7.2 or below)	`org.dspace.app.mediafilter.PowerPointFilter`	extracts the full text of slides and notes in Microsoft PowerPoint and PowerPoint XML documents for full text indexing. (Uses the Apache POI tools.)	Microsoft Powerpoint, Microsoft Powerpoint XML	yes
PDFBox JPEG Thumbnail	`org.dspace.app.mediafilter.PDFBoxThumbnail`	creates thumbnail images of the first page of PDF files	Adobe PDF	yes
JPEG Thumbnail	`org.dspace.app.mediafilter.JPEGFilter`	creates thumbnail images of GIF, JPEG and PNG files	BMP, GIF, JPEG, image/png	yes
Branded Preview JPEG	`org.dspace.app.mediafilter.BrandedPreviewJPEGFilter`	creates a branded preview image for GIF, JPEG and PNG files	BMP, GIF, JPEG, image/png	no

ImageMagick Image Thumbnail Generator	`org.dspace.app.mediafilter.ImageMagickImageThumbnailFilter`	Uses ImageMagick to generate thumbnails for image bitstreams. Requires installation of ImageMagick on your server. See ImageMagick Media Filters.	BMP, GIF, image/png, JPG, TIFF, JPEG, JPEG 2000	no
ImageMagick PDF Thumbnail Generator	`org.dspace.app.mediafilter.ImageMagickPdfThumbnailFilter`	Uses ImageMagick and Ghostscript to generate thumbnails for PDF bitstreams. Requires installation of ImageMagick and Ghostscript on your server. See ImageMagick Media Filters .	Adobe PDF	no

Please note that the `filter-media` script will automatically update the DSpace search index by default.

Enabling/Disabling MediaFilters

The media filter plugin configuration `filter.plugins` in `dspace.cfg` contains a list of all enabled media/format filter plugins (see Configuring Media Filters for more information). By modifying the value of `filter.plugins` you can disable or enable MediaFilter plugins. The `filter.plugins` setting can be set multiple times to enable multiple filters. Each filter must be enabled via its name (see "Name" column in the table above).

```
# Enable the default Text Extractor (for 7.3 or above)
filter.plugins = Text Extractor

# Enable the JPEG thumbnail creator
filter.plugins = JPEG Thumbnail

# Enable the PDF thumbnail creator
filter.plugins = PDFBox JPEG Thumbnail
```

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.
- **From Date mode (8.1 or above):** `[dspace]/bin/dspace filter-media -d 2020-12-30`
  - Only process items that have been modified/created after a specified date (provided in YYYY-MM-DD format).
  - This may be combined with other options, e.g. "-f -d YYYY-MM-DD" would use force mode on all Items modified after a specific date. However, it cannot be combined with "-i [identifier]" at this time.
- **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.
- **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 list, one identifier per line, within a separate file (e.g. *filter-skiplist.txt*). Use the following format to call the program.
    - `[dspace]/bin/dspace filter-media -s $(paste -sd, - < filter-skiplist.txt)`
- **Verbose mode** : `[dspace]/bin/dspace filter-media -v`
  - Print all extracted text and other filter details to STDOUT.

Creating Custom MediaFilters

Adding your own filters is done by creating a class which *implements* the `org.dspace.app.mediafilter.FormatFilter` interface. See the Creating a new Media/Format Filter topic and comments in the source file `FormatFilter.java` for more information. In theory filters could be implemented in any programming language (C, Perl, etc.) However, they need to be invoked by the Java code in the Media Filter class that you create.

Creating 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");
```

Configuration parameters

Property	**textextractor.max-chars** (only in 7.3 or above)
Example Value	textextractor.max-chars = 100000
Informational Note	By default, the "Text Extractor" only extracts the first 100,000 characters of text for full-text indexing. This setting allows you to increase or decrease that default. Set to -1 for no maximum. Keep in mind that larger values (or -1) are more likely to encounter OutOfMemoryException errors when extracting text from very large files. In those scenarios, you may wish to consider instead enabling "textextractor.use-temp-file" below to better control memory usage.
Property	**textextractor.max-array**
Example Value	textextractor.max-array = 250000000
Informational Note	By default, the "Text Extractor" can only buffer the first 100 million characters of a Microsoft Office input record. This setting allows you to increase that limit. Set this if you are seeing `RecordFormatException` calling out excessive array length from `dspace filter-media`. It is likely that you will need to increase the size of the Java heap if you greatly increase this value -- see JAVA_OPTS in 'bin/dspace' or 'bin/dspace.bat'.
Property	**textextractor.use-temp-file** (only in 7.3 or above)
Example Value	textextractor.use-temp-file = false
Informational Note	By default, the "Text Extractor" will perform all text extraction in memory (i.e. textextractor.use-temp-file=false). This ensures text extraction runs quickly, but it has the risk of hitting OutOfMemoryException errors if you either increase "textextractor.max-chars" or simply don't have much available memory on the server. In those scenarios, you can set "textextractor.use-temp-file=true" in order to tell the text extraction process to extract all text using a temporary file. This *decreases* the memory usage of the text extraction process, but will run slightly slower.
Property	**filter.org.dspace.app.mediafilter.publicPermission**
Example Value	filter.org.dspace.app.mediafilter.publicPermission = JPEGFilter
Informational Note	By default mediafilter derivatives / thumbnails inherit the permissions of the parent bitstream, but you can override this, in case you want to make publicly accessible derivative / thumbnail content, typically the thumbnails of objects for the browse list. List the MediaFilter names that would get public accessible permissions. Any media filters not listed will instead inherit the permissions of the parent bitstream.

ImageMagick Media Filters

ImageMagick Media Filters

1 ImageMagick Media Filters
- 1.1 Overview
- 1.2 Installation
- 1.3 DSpace Configuration
  - 1.3.1 Thumbnail Dimensions
  - 1.3.2 Conversion Utility Path
  - 1.3.3 Supported file formats
  - 1.3.4 Overwriting Existing Thumbnails
  - 1.3.5 Flatten
  - 1.3.6 ICC Profiles
  - 1.3.7 Override ImageMagick Default Density
- 1.4 Additional Customization
- 1.5 Possible Errors / Issues
  - 1.5.1 "convert.im6: not authorized" errors
  - 1.5.2 "convert-im6.q16: cache resources exhausted" errors

As of DSpace 7.6, the ImageMagick media filter also supports creating thumbnails of video (MP4) files, provided that "ffmpeg" is installed locally. See instructions below.

Overview

The ImageMagick Media Filters provide consistent, high quality thumbnails for image bitstreams, PDF bitstreams and video (MP4) bitstreams.

These filters require a separate software installation of the conversion utilities: ImageMagick, Ghostscript (to support PDF thumbnails) and/or ffmpeg (to support MP4 thumbnails).

The media filters use the library im4java to invoke the conversion utilities. This library constructs a conversion command launches a sub-process to perform the generation of media files.

Installation

Before ImageMagick Media Filters can be used, you must setup ImageMagick (and optionally Ghostscript) as follows:

1. Install ImageMagick on your server. The installation process differs based on your operating system. For example, on Debian/Ubuntu, it's similar to this:

```
apt-get install imagemagick
```

2. If you wish to generate PDF thumbnails, install Ghostscript on your server. The installation process differs based on your operating system. For example, on Debian/Ubuntu, it's similar to this:

```
apt-get install ghostscript
```

3. (New in 7.6) If you wish to generate MP4 (video) thumbnails, install FFmpeg on your server. The installation process differs based on your operating system. For example, on Debian/Ubuntu, it's similar to this:

```
apt-get install ffmpeg
```

4. The ImageMagick, Ghostscript, and FFmpeg executables should be accessible from the same directory (e.g. `/usr/bin`)
   a. This directory MUST be defined in the `org.dspace.app.mediafilter.ImageMagickThumbnailFilter.ProcessStarter` configuration as describe below.

DSpace Configuration

In the **filter.plugins** section of your **dspace.cfg** (or local.cfg) file, specify the ImageMagick media filters you wish to use.

local.cfg

```
# Make sure to always keep this plugin enabled if you want to support search within text documents
filter.plugins = Text Extractor

# NOTE: When "ImageMagick Image Thumbnail" is enabled, the default "JPEG Thumbnail" should NOT be enabled
filter.plugins = ImageMagick Image Thumbnail

# NOTE: When "ImageMagick PDF Thumbnail" is enabled, the default "PDFBox JPEG Thumbnail" should NOT be enabled
# Requires Ghostscript to also be installed
filter.plugins = ImageMagick PDF Thumbnail

# New in 7.6, this will generate thumbnails from video files.
# Requires ffmpeg to also be installed
filter.plugins = ImageMagick Video Thumbnail
```

This will activate the following settings which are already present in **dspace.cfg** (these do NOT need to be added, as they already exist)

```
plugin.named.org.dspace.app.mediafilter.FormatFilter = org.dspace.app.mediafilter.
ImageMagickImageThumbnailFilter = ImageMagick Image Thumbnail
plugin.named.org.dspace.app.mediafilter.FormatFilter = org.dspace.app.mediafilter.ImageMagickPdfThumbnailFilter
= ImageMagick PDF Thumbnail
plugin.named.org.dspace.app.mediafilter.FormatFilter = org.dspace.app.mediafilter.
ImageMagickVideoThumbnailFilter = ImageMagick Video Thumbnail
```

These media filters contain the several properties which can be configured.

Thumbnail Dimensions

The following properties are used to define the dimensions of the generated thumbnails:

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

Conversion Utility Path

The following property provides a path to the ImageMagick (convert), GhostScript (ghostscript), and ffmpeg utilities.

```
org.dspace.app.mediafilter.ImageMagickThumbnailFilter.ProcessStarter = /usr/bin
```

Supported file formats

Each of these ImageMagick filters has its own list of configurable file formats. The defaults are usually best, but may be updated if you have custom bitstream formats. These settings already exist in your dspace.cfg.

```
filter.org.dspace.app.mediafilter.ImageMagickImageThumbnailFilter.inputFormats = BMP, GIF, PNG, JPG, TIFF,
JPEG, JPEG 2000
filter.org.dspace.app.mediafilter.ImageMagickPdfThumbnailFilter.inputFormats = Adobe PDF
filter.org.dspace.app.mediafilter.ImageMagickVideoThumbnailFilter.inputFormats = Video MP4
```

Overwriting Existing Thumbnails

The ImageMagick media filters can differentiate thumbnails created by the DSpace default thumbnail generator and thumbnails that were manually uploaded by a user. The media filter reads the bitstream description field to make this determination. A regular expression can be provided to define the set of thumbnails that should be overwritten by the ImageMagick thumbnail generator. Thumbnail descriptions matching this pattern will be overwritten even if the -f option is not passed to the filter media process.

```
org.dspace.app.mediafilter.ImageMagickThumbnailFilter.replaceRegex = ^Generated Thumbnail$
```

The ImageMagick media filter will use the bitstream description field to identify bitstreams that it has created using the following setting. Bitstreams containing this label will be overwritten only if the -f filter is applied.

```
org.dspace.app.mediafilter.ImageMagickThumbnailFilter.bitstreamDescription = IM Thumbnail
```

Thumbnail descriptions that do not match either of the patterns listed above are presumed to be manually uploaded thumbnails. These thumbnails will not be replaced even if the -f option is passed to the filter media process.

Flatten

DSpace uses the JPEG format for thumbnails. While JPEG doesn't support transparency, PDF, PNG and other formats do. As those formats are used as outgoing material in DSpace, DSpace has to care about transparency during the generation of the thumbnails. In combinations of specific versions of ImageMagick and Ghostscript it may occur that completely transparent areas will become black. As a solution ImageMagick recommends to flatten images extracted from PDFs before they are stored as JPEG.

Since DSpace 5.2 the ImageMagick media filter flattens thumbnails extracted from PDFs. If you run into problems caused by flattening of the extracted images, you can switch the flattening off by setting the following property in **dspace.cfg** to false:

```
org.dspace.app.mediafilter.ImageMagickThumbnailFilter.flatten = false
```

ICC Profiles

PDFs optimized for physical printing often use the CMYK color space. On the web, however, the de facto color system is sRGB. By default, DSpace's ImageMagick-based thumbnailing system will create thumbnails that use the same color space as the source PDF. Most web browsers are not able to correctly display images that use the CMYK color space, which leads to images with visibly inaccurate colors.

If you are using Ghostscript version 9 or above, it is possible for DSpace to correctly convert images from CMYK to sRGB by providing it with appropriate ICC color profiles to use during thumbnail creation. Default ones are provided by most Ghostscript installations (version 9 or above). The following configuration options tell DSpace where those ICC profiles are located.

```
# org.dspace.app.mediafilter.ImageMagickThumbnailFilter.cmyk_profile = /usr/share/ghostscript/9.18/iccprofiles
/default_cmyk.icc
# org.dspace.app.mediafilter.ImageMagickThumbnailFilter.srgb_profile = /usr/share/ghostscript/9.18/iccprofiles
/default_rgb.icc
```

You may need to adjust those paths for your OS or the version of Ghostscript that you have.

Providing ICC profiles to ImageMagick is optional. If these configuration properties are unset, no profiles will be supplied to ImageMagick, and thumbnails produced from PDFs using the CMYK color space will also use CMYK. The transformation from CMYK to RGB is optional.

Override ImageMagick Default Density

It is possible to override ImageMagick's default density of 72 DPI when creating PDF thumbnails. This increases the quality of resulting thumbnails at the expense of slightly longer execution times and higher memory usage. Any integer over 72 will help, but recommend 144 for a "2x" supersample. See the following configuration option in `dspace.cfg` :

```
# org.dspace.app.mediafilter.ImageMagickThumbnailFilter.density = 144
```

The effect is most notable on PDFs with a lot of text, gradients, or curved lines. See the pull request implementing this feature for more information and comparisons.

Additional Customization

The ImageMagick conversion software provides a large number of conversion options. Subclasses of these media filters could be written to take advantage of the additional conversion properties available in the software.

Note: The PDF thumbnail generator is hard-coded to generate a thumbnail from the first page of the PDF.

Possible Errors / Issues

"convert.im6: not authorized" errors

On Ubuntu (possibly other OSes), you may see errors like these when attempting to generate PDF thumbnails:

```
ERROR filtering, skipping bitstream:
        Item Handle: 1234/5678
        Bundle Name: ORIGINAL
        File Size: 30406135
        Checksum: c1df4b3a4755e9bed956383b61fc5042 (MD5)
        Asset Store: 0
org.im4java.core.CommandException: org.im4java.core.CommandException: convert.im6: not authorized `/tmp
/impdfthumb6294641076817830415.pdf' @ error/constitute.c/ReadImage/454.

OR

org.im4java.core.CommandException: convert-im6.q16: attempt to perform an operation not allowed by the security
policy 'PDF' @ error/constitute.c/IsCoderAuthorized/421
```

These may be caused by a change in your ImageMagick policy configuration on your server.

In Ubuntu, the default "policy.xml" was recently updated to **exclude** all Ghostscript formats (including PDF, PS, etc). See this ticket: https://bugs.launchpad.net/ubuntu/+source/imagemagick/+bug/1796563

- This exclusion was implemented to workaround a security vulnerability in Ghostscript reported here: https://www.kb.cert.org/vuls/id/332928
- According to that vulnerability report, this was patched in Ghostscript v9.24 (or above)

The newly added lines in the `/etc/ImageMagick/policy.xml` are these ones:

```
<!-- disable ghostscript format types -->
<policy domain="coder" rights="none" pattern="PS" />
<policy domain="coder" rights="none" pattern="EPS" />
<policy domain="coder" rights="none" pattern="PDF" />
<policy domain="coder" rights="none" pattern="XPS" />
```

To fix the error above requires you to re-enable ImageMagick to process Ghostscript format types. That can be done by simply commenting out those new "policy" lines in the configuration file (surround them with <!-- and --> to comment out)

Be aware that you MUST ensure you are running Ghostscript v9.24 or later to ensure that you are not at risk for the above security vulnerability in older versions of Ghostscript.

"convert-im6.q16: cache resources exhausted" errors

On Debian, and possibly other OSes, you may see errors like these when attempting to generate video thumbnails (especially if video files are big):

```
File: video.mp4.jpg
ERROR filtering, skipping bitstream:
        Item Handle: 1234/5678
        Bundle Name: ORIGINAL
        File Size: 146761357
        Checksum: 735ceb1b6b249afc84a5bb1b87ae0c02 (MD5)
        Asset Store: 0
org.im4java.core.CommandException: convert-im6.q16: cache resources exhausted `/tmp/magick-64dziU-
1nQJjQHZYu4_R1fFP4l9en5iL.pam' @ error/cache.c/OpenPixelCache/4095.
```

These may be caused by too conservative resource policies in your `policy.xml` file. As an example, default values are located at `/etc/ImageMagick-6/policiy.xml` in Debian 11 (Bullseye), and are:

/etc/ImageMagick-6/policiy.xml

```
<policymap>
  <!-- <policy domain="resource" name="temporary-path" value="/tmp"/> -->
  <policy domain="resource" name="memory" value="256MiB"/>
  <policy domain="resource" name="map" value="512MiB"/>
  <policy domain="resource" name="width" value="16KP"/>
  <policy domain="resource" name="height" value="16KP"/>
  <!-- <policy domain="resource" name="list-length" value="128"/> -->
  <policy domain="resource" name="area" value="128MP"/>
  <policy domain="resource" name="disk" value="1GiB"/>
```

To avoid the `cache resources exhausted` error, try increasing the resource limits policies. You may want to start by increasing the `memory` and `disk` policies (disk cache is used when the memory limit is reached). The actual values have to be adjusted depending on the size of your video bitstreams and the actual resources available in your installation. For example:

/etc/ImageMagick-6/policiy.xml

```
<policymap>
  <!-- <policy domain="resource" name="temporary-path" value="/tmp"/> -->
  <policy domain="resource" name="memory" value="4GiB"/> <!-- memory limit increased from 256MiB to 4GiB -->
  <policy domain="resource" name="map" value="512MiB"/>
  <policy domain="resource" name="width" value="16KP"/>
  <policy domain="resource" name="height" value="16KP"/>
  <!-- <policy domain="resource" name="list-length" value="128"/> -->
  <policy domain="resource" name="area" value="128MP"/>
  <policy domain="resource" name="disk" value="4GiB"/> <!-- disk limit increased from 1GiB to 4GiB -->
```

For a detailed description of the ImageMagick limits, see https://imagemagick.org/script/command-line-options.php#limit.

Once the limits are properly set, a successful execution of the filter should show a message similar to:

```
File: video.mp4.jpg
FILTERED: bitstream 12345678-abcd-efgh-ijkl-1234567890ab (item: 1234/5678) and created 'video.mp4.jpg'
```

Performance Tuning DSpace

1 Bare Minimum Requirements
2 Blocking Aggressive Bots
3 Performance Tuning the Frontend (UI)
    3.1 Use "cluster mode" of PM2 to avoid Node.js using a single CPU
    3.2 Give Node.js more memory
    3.3 Limit which pages are processed via Server Side Rendering (SSR)
    3.4 Turn on (or increase) caching of Server Side Rendered pages
4 Performance Tuning the Backend (REST API)
    4.1 Give Tomcat More Memory
        4.1.1 Give Tomcat More Java Heap Memory
        4.1.2 Give Tomcat More Java PermGen Memory
        4.1.3 Choosing the size of memory spaces allocated to DSpace Backend
    4.2 Give the Command Line Tools More Memory
        4.2.1 Give the Command Line Tools More Java Heap Memory
        4.2.2 Give the Command Line Tools More Java PermGen Space Memory
5 Give PostgreSQL Database More Memory
6 Performance Tuning Solr

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.

Bare Minimum Requirements

As of this writing, DSpace is likely to require 4GB of memory at a *bare minimum*. However, with that little memory, you may quickly hit memory issues with any significant user activity or bulk uploading. So, *we recommend running DSpace with at least 8-16GB* (or more for very large or very active sites).

This minimum would roughly include...

- 2GB of memory for the Frontend (UI) / Node.js. Highly active sites will need more.
- 1GB of memory for the Backend (REST API) / JVM / Tomcat. Highly active sites will need more.
- 512MB of memory for PostgreSQL database. Highly active sites will need more.
- 512MB of memory for Solr. Highly active sites may need more.
- Extra memory may be required for command line scripts (which get kicked off in a separate JVM)

Keep in mind, because the frontend & backend can be run on separate servers, you can split this memory across two (or more) servers. You can even choose to run PostgreSQL or Solr either alongside the backend or on their own dedicated server.

The DSpace frontend (UI) will often require several CPUs, especially if you wish to use "cluster mode" (see below) to better scale your application. A smaller application may be able to use 4-6 CPU cores, while highly active sites may require additional CPU power. CPU is most often necessary for the frontend's Angular Serve Side Rendering (again see "cluster mode" notes below) and for any batch processing / command line scripts on backend.

Blocking Aggressive Bots

DSpace itself doesn't have built-in tools to allow you to block various aggressive bots. However, with the growth of Artificial Intelligence, many DSpace sites have experienced aggressive harvesting of their metadata / files by various AI-related bots. These aggressive bots can cause performance problems with DSpace sites if not dealt with in some way.

If you are experiencing this sort of aggressive harvesting, we'd recommend looking at the following resources (none of these are managed by DSpace, but all are community efforts by various library or repository-based developers):

- Aggressive AI Harvesting of Digital Resources
- https://wiki.code4lib.org/Blocking_Bots and the #bots channel on the code4lib Slack team.

Performance Tuning the Frontend (UI)

Use "cluster mode" of PM2 to avoid Node.js using a single CPU

If you are using PM2 to run the User Interface, you may want to start it using PM2's "Cluster Mode". This allows Node.js applications to be scaled across multiple CPUs by using the Node.js cluster module. See the PM2 Cluster Mode documentation at https://pm2.keymetrics.io/docs/usage/cluster-mode/

There are two ways to enable cluster mode. Choose one.

1. First, is by adding the "exec_mode" and "instances" settings to your JSON configuration as follows. You also may want to set the "max_memory_restart" option to avoid PM2 using too much memory. These three settings are described in more detail below. NOTE: make sure to start (or restart) your site to enable these settings (e.g. `pm2 start dspace-ui.json`)

dspace-ui.json

```
{
    "apps": [
        {
           "name": "dspace-ui",
           "cwd": "/full/path/to/dspace-ui-deploy",
           "script": "dist/server/main.js",
           "instances": "max",
           "exec_mode": "cluster",
           "env": {
              "NODE_ENV": "production"
           },
           "max_memory_restart": "500M"
        }
    ]
}
```

a. Setting "exec_mode" to "cluster" will enable cluster mode,
b. The "instances" setting allows you to customize how many CPUs are available to PM2 ("max" = all CPUs. But you also can specify a number like "8" = 8 CPUs.)
c. The "max_memory_restart" setting is *optional* but tells PM2 how much memory to allow **per instance.** The example above has a maximum of 500MB. If the number of 'instances' is 8, that would mean PM2 could use up to 8 x 500MB = 4GB of memory. *Therefore, you may wish to modify the values of "instances" and/or "max_memory_restart" to better control the memory available to PM2.*

2. Alternatively, you can use command line flags to specify the same settings described above. The "-i" flag enables cluster mode and specifies the number of instances. The "--max-memory-restart" flag limits the memory per instance.

```
# Start the "dspace-ui" app. Cluster it across all available CPUs with a maximum memory of 500MB per CPU.
# This command is equivalent to the example cluster settings in the "dspace-ui.json" file above.
pm2 start dspace-ui.json -i max --max-memory-restart 500M
```

Give Node.js more memory

On machines with >2GB of memory available, Node will only use a maximum of 2GB of memory by default (see https://github.com/nodejs/node/issues/28202). This 2GB of memory should be enough to build & run the User Interface, but it's possible that highly active sites may require 4GB or more.

If you want to increase the memory available to Node.js, you can set the NODE_OPTIONS environment variable:

```
# Increase memory limit to 4GB (4096MB) by setting "max-old-space-size"
# in your NODE_OPTIONS environment variable
export NODE_OPTIONS=--max-old-space-size=4096
```

Limit which pages are processed via Server Side Rendering (SSR)

While enabling Server Side Rendering (SSR) is extremely important for Search Engine Optimization, it can also be very resource intensive for large pages or highly active sites. Server Side Rendering involves building the entire HTML for the page in Node.js (on your server) before sending the page back to the client/user. Most humans only encounter SSR briefly, when they initially visit your site. However, bots *may only interact* with SSR, especially if they are unable to process Javascript. This is true even for Google Scholar, whose bots will only use SSR generated pages to index your site.

In order to maximum the performance of SSR, by default, DSpace will minimize the pages and Angular components that are processed during server side rendering. You may wish to review the default settings to ensure they are appropriate for your site. See the Server Side Rendering (SSR) Settings

Turn on (or increase) caching of Server Side Rendered pages

As of DSpace 7.5, we now provide basic, in-memory caching of server-side rendered (SSR) pages. Server-side rendering is used to pre-generate full HTML pages to pass back to users (primarily anonymous users and bots). This is necessary for Search Engine Optimization (SEO) as some web crawlers cannot use Javascript. It also can be used to immediately show the first HTML page to users while the Javascript app loads in the user's browser.

While server-side-rendering is highly recommended on all sites, it can result in Node.js having to pre-generate many HTML pages at once when a site has a large number of simultaneous users/bots. This may cause Node.js to spend a lot of time processing server-side-rendered content, slowing down the entire site.

Therefore, DSpace provides some basic caching of server-side rendered pages, which allows the same pre-generated HTML to be sent to many users /bots at once & decreases the frequency of server-side rendering.

These settings are documented at User Interface Configuration: Cache Settings - Server Side Rendering (SSR)

Performance Tuning the Backend (REST API)

Give Tomcat 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 Java Heap Memory to ensure optimal DSpace operation. Most larger sized or highly active DSpace installations however tend to allocate more like 1024MB (1GB) to 2048MB (2G) or more 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: https://frankkieviet.blogspot.com/2006/10/classloader-leaks-dreaded-permgen-space.html)

To increase the amount of PermGen memory available to Tomcat (default=64MB), use either the `JAVA_OPTS` or `CATALINA_OPTS` environment variable, e.g:

```
CATALINA_OPTS=-XX:MaxPermSize=128m
```

OR

```
JAVA_OPTS=-XX:MaxPermSize=128m
```

The above example sets the maximum PermGen memory to 128MB.

Difference between JAVA_OPTS and CATALINA_OPTS

You can use either environment variable. `JAVA_OPTS` is also used by other Java programs (besides just Tomcat). `CATALINA_OPTS` is *only used* by Tomcat. So, if you only want to tweak the memory available to Tomcat, it is recommended that you use `CATALINA_OPTS`. If you set **both** `CATALINA_OPTS` and `JAVA_OPTS`, Tomcat will default to using the settings in `CATALINA_OPTS`.

Please note that you can obviously set **both** Tomcat's Heap space and PermGen Space together similar to:
`CATALINA_OPTS=-Xmx512m -Xms512m -XX:MaxPermSize=128m`

On an Ubuntu machine (10.04) at least, the file `/etc/default/tomcat6` appears to be the best place to put these environmental variables.

Choosing the size of memory spaces allocated to DSpace Backend

psi-probe is a webapp that can be deployed in DSpace and be used to watch memory usage of the other webapps deployed in the same instance of Tomcat (in our case, the DSpace server webapp).

1. Download the latest version of psi-probe from https://github.com/psi-probe/psi-probe
2. Unzip probe.war into [dspace]/webapps/

```
cd [dspace]/webapps/
unzip ~/probe-3.1.0.zip
unzip probe.war -d probe
```

3. Add a Context element in Tomcat's configuration, and make it privileged (so that it can monitor the other webapps):
   EITHER in `$CATALINA_HOME/conf/server.xml`

```
<Context docBase="[dspace]/webapps/probe" privileged="true" path="/probe" />
```

   OR in `$CATALINA_HOME/conf/Catalina/localhost/probe.xml`

```
<Context docBase="[dspace]/webapps/probe" privileged="true" />
```

4. Edit `$CATALINA_HOME/conf/tomcat-users.xml` to add a user for logging into psi-probe (see more in https://github.com/psi-probe/psi-probe/wiki/InstallationApacheTomcat)

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <user username="admin" password="t0psecret" roles="manager" />
</tomcat-users>
```

5. Restart Tomcat
6. Open http://yourdspace.com:8080/probe/ (edit domain and port number as necessary) in your browser and use the username and password from tomcat-users.xml to log in.

In the "System Information" tab, go to the "Memory utilization" menu. Note how much memory Tomcat is using upon startup and use a slightly higher value than that for the `-Xms` parameter (initial Java heap size). Watch how big the various memory spaces get over time (hours or days), as you run various common DSpace tasks that put load on memory, including indexing, reindexing, importing items into the oai index etc. These maximum values will determine the `-Xmx` parameter (maximum Java heap size). Watching PS Perm Gen grow over time will let you choose the value for the `-XX:MaxPermSize` parameter.

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`

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

Performance Tuning Solr

Solr has it's own detailed documentation with recommendations for "Taking Solr to Production". We recommend following the recommendations from Solr, especially related to "Ulimit settings" (for Unix-based systems) and "Avoiding Swapping" (for Unix-based systems). See the Solr documentation for more details.

Ping or Healthcheck endpoints for confirming DSpace services are functional

For some installations of DSpace, it might be helpful to have a URL you can configure as a healthcheck for some sort of monitoring system (Monit, Eye). Some installations use load balancers, and those load balancers need a URL to check to confirm the system is functioning correctly. Here are some suggestions for you to use.

Frontend

```
/home
```

Be sure to append that path to the main URL of your DSpace instance's frontend URL. For example: https://demo7.dspace.org/home

Backend

```
/server/api/core/collections
```

```
/server/api/core/sites
```

Be sure to append these paths to the main URL of your DSpace instance's backend URL. For example: https://api7.dspace.org/server/api/core/collections

Both of those endpoints will throw an error if Solr is down or similar, and both are anonymously available (no login required).

Proxy Configuration

If your DSpace instance must access the internet through a proxy server (e.g., to connect to external services like CrossRef, PubMed, ORCID, etc.), you need to configure the proxy settings in the `dspace.cfgl`.

Where to Configure

The proxy configuration settings are located in the `dspace.cfg` file

You can override these settings in your `local.cfg` depending on your setup.

Required Settings

Here are the main proxy-related properties you can configure. For the full reference please check: Configuration Reference#ProxySettings

1. Enable Proxy Support

`useProxy` = `true`

Set this to `true` to enable proxy support globally in DSpace. This property is already set as true by default

2. Set Proxy Host

`http.proxy.host` = `<your.proxy.host>`

Specify the hostname or IP address of your proxy server.

3. Set Proxy Port

`http.proxy.port` = `<proxy port>`

Specify the port used by your proxy server.

4. Exclude specific hostnames from using the proxy

`http.proxy.hosts-to-ignore` = `<proxy port>`

As mentioned above, the proxy configuration is applied globally. Some connections are excluded by default—for example, SOLR connections, which are expected to occur over a private network.

In some cases, due to external host requirements or other specific needs, you may want to disable or restrict the use of the proxy for certain hostnames. This is where the **`http.proxy.hosts-to-ignore`** property becomes useful.

Scheduled Tasks via Cron

Several DSpace features **require** that a script is run regularly (via cron, or similar). Some of these features include:

- the e-mail subscription feature that alerts users of new items being deposited;
- the 'media filter' tool, that generates thumbnails of images and extracts the full-text of documents for indexing;
- the (optional) 'checksum checker' that tests the bitstreams in your repository for corruption;
- and the (optional) registration of DOIs using DataCite as registration agency.

There are some optional periodic tasks as well:

- Updating the geolocation database used to enrich usage statistics. At this writing, the database publisher issues monthly updates.

These regularly scheduled tasks should be setup via either cron (for Linux/Mac OSX) or Windows Task Scheduler (for Windows).

Recommended Cron Settings

If you are on Linux or Mac OSX, **you should add these cron settings under the OS account which is running Tomcat (and owns the [dspace] installation directory)**. For example, login as that user and type the following to edit the user's crontab.

```
crontab -e
```

Avoid scheduling cron tasks at midnight

Generally, we recommend avoiding scheduling any DSpace cron tasks to run at exactly midnight. DSpace's log4j settings default to rolling over the log file on a daily basis. Therefore any cron tasks which trigger at midnight have a risk of causing logging issues, or having their logs split over two log files.

While every DSpace installation is unique, in order to get the most out of DSpace, we highly recommend enabling these basic cron settings (the settings are described in the comments):

```
## SAMPLE CRONTAB FOR A PRODUCTION DSPACE
## You obviously may wish to tweak this for your own installation,
## but this should give you an idea of what you likely wish to schedule via cron.
##
## NOTE: You may also need to add additional sysadmin related tasks to your crontab
## (e.g. zipping up old log files, or even removing old logs, etc).

#-----------------
# GLOBAL VARIABLES
#-----------------
# Full path of your local DSpace Installation (e.g. /home/dspace or /dspace or similar)
# MAKE SURE TO CHANGE THIS VALUE!!!
DSPACE = [dspace]

# Shell to use
SHELL=/bin/sh

# Add all major 'bin' directories to path
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# Set JAVA_OPTS with defaults for DSpace Cron Jobs.
# Only provides 512MB of memory by default (which should be enough for most sites).
JAVA_OPTS="-Xmx512M -Xms512M -Dfile.encoding=UTF-8"

#--------------
# HOURLY TASKS (Recommended to be run multiple times per day, if possible)
# At a minimum these tasks should be run daily.
#--------------

# Send information about new and changed DOIs to the DOI registration agency
# NOTE: ONLY NECESSARY IF YOU REGISTER DOIS USING DATACITE AS REGISTRATION AGENCY (Disabled by default)
# 0 4,12,20 * * *  $DSPACE/bin/dspace doi-organiser -u -q ; $DSPACE/bin/dspace doi-organiser -s -q ; $DSPACE/bin
/dspace doi-organiser -r -q ; $DSPACE/bin/dspace doi-organiser -d -q

#----------------
# DAILY TASKS
# (Recommended to be run once per day. Feel free to tweak the scheduled times below.)
#----------------
```

```
# Update the OAI-PMH index with the newest content at 12:15am every day
# REQUIRED to update content available in OAI-PMH (However, it can be removed if you do not enable OAI-PMH)
15 0 * * * $DSPACE/bin/dspace oai import > /dev/null

# Clean and Update the Discovery indexes at 12:15am every day
# (This ensures that any deleted documents are cleaned from the Discovery search/browse index)
# RECOMMENDED to ensure your search/browse index stays fresh.
15 0 * * * $DSPACE/bin/dspace index-discovery > /dev/null

# run the index-authority script once a day at 12:45am to ensure the Solr Authority cache is up to date
45 0 * * * $DSPACE/bin/dspace index-authority > /dev/null

# Cleanup Web Spiders from DSpace Statistics Solr Index at 01:00 every day
# (This removes any known web spiders from your usage statistics)
# RECOMMENDED if you are running Solr Statistics.
0 1 * * * $DSPACE/bin/dspace stats-util -f

# Send out "daily" update subscription e-mails at 02:00 every day
# (This sends an email to any users who have "subscribed" to a Community/Collection, notifying them of newly
added content.)
# (This job must run before the `filter-media` task, because `filter-media` may update the Solr index, which
could delay sending subscription emails about the new content until the following day.)
# REQUIRED for daily "Email Subscriptions" to work properly.
0 2 * * * $DSPACE/bin/dspace subscription-send -f D

# Run the media filter at 03:00 every day.
# (This task ensures that thumbnails are generated for newly add images,
# and ensures full text search is available for newly added PDF/Word/PPT/HTML documents)
# REQUIRED for Thumbnails to be generated & full-text indexing to work.
0 3 * * * $DSPACE/bin/dspace filter-media

#----------------
# WEEKLY TASKS
# (Recommended to be run once per week, but can be run more or less frequently, based on your local needs
/policies)
#----------------
# Send out "weekly" update subscription e-mails at 02:00 every Sunday
# (This sends an email to any users who have "subscribed" to a Community/Collection, notifying them of newly
added content.)
# REQUIRED for weekly "Email Subscriptions" to work properly.
0 2 * * 0 $DSPACE/bin/dspace subscription-send -f W

# Run the checksum checker at 04:00 every Sunday
# By default it runs through every file (-l) and also prunes old results (-p)
# (This re-verifies the checksums of all files stored in DSpace. If any files have been changed/corrupted,
checksums will differ.)
# OPTIONAL, but useful if you want to enable regular regular checksum validation of files stored in DSpace.
# 0 4 * * 0 $DSPACE/bin/dspace checker -l -p
# NOTE: LARGER SITES MAY WISH TO USE DIFFERENT OPTIONS. The above "-l" option tells DSpace to check
*everything*.
# If your site is very large, you may need to only check a portion of your content per week. The below
commented-out task
# would instead check all the content it can within *one hour*. The next week it would start again where it
left off.
# 0 4 * * 0 $DSPACE/bin/dspace checker -d 1h -p
# Mail the results of the checksum checker (see above) to the configured "mail.admin" at 05:00 every Sunday.
# (This ensures the system administrator is notified whether any checksums were found to be different.)
# 0 5 * * 0 $DSPACE/bin/dspace checker-emailer

#----------------
# MONTHLY TASKS
# (Recommended to be run once per month, but can be run more or less frequently, based on your local needs
/policies)
#----------------
# Send out "monthly" update subscription e-mails at 02:00, on the first of every month
# (This sends an email to any users who have "subscribed" to a Community/Collection, notifying them of newly
added content.)
# REQUIRED for monthly "Email Subscriptions" to work properly.
0 2 1 * * $DSPACE/bin/dspace subscription-send -f M
```

```
# Permanently delete any bitstreams flagged as "deleted" in DSpace, on the first of every month at 01:00
# (This ensures that any files which were deleted from DSpace are actually removed from your local filesystem.
#  By default they are just marked as deleted, but are not removed from the filesystem.)
# REQUIRED to fully remove deleted content files from the "assetstore" folder
0 1 1 * * $DSPACE/bin/dspace cleanup > /dev/null
```

Search Engine Optimization

Please be aware that individual search engines also have their own guidelines and recommendations for inclusion. While the guidelines below apply to **most** DSpace sites, you may also wish to review these guidelines for specific search engines:

- "Indexing Repositories: Pitfalls and Best Practices" talk from Anurag Acharya (co-creator of Google Scholar) presented at the Open Repositories 2015 conference
- Google Scholar Inclusion Guidelines
- Bing Webmaster Guidelines

Ensuring your DSpace is indexed

Anyone who has analyzed traffic to their DSpace site (e.g. using Google Analytics or similar) will notice that a significant (and in many cases a majority) of visitors arrive via a search engine such as Google or other search engines. Hence, to help maximize the impact of content and thus encourage further deposits, it is important to ensure that your DSpace instance is indexed effectively.

DSpace comes with tools that ensure major search engines (Google, Bing, Google Scholar) are able to easily and effectively index all your content. However, many of these tools provide some basic setup. Here's how to ensure your site is indexed.

Basic SEO validation is now provided in the DSpace Admin User Interface

DSpace now has a basic Search Engine Optimization (SEO) validator which can provide you feedback on how well your site may align with the below SEO policies.

For the optimum indexing, you should:

1. Check SEO Validator status to detect any obvious issues
2. Keep your DSpace up to date. We are constantly adding new indexing improvements in new releases
3. Ensure your DSpace is visible to search engines.
4. Ensure your proxy is passing X-Forwarded headers to the User Interface
5. Ensure the user interface is using server-side rendering *(enabled by default)*
6. Ensure the sitemaps feature is enabled. *(enabled by default)*
7. Ensure your robots.txt allows access to item "splash" pages and full text.
8. Ensure item metadata appears in HTML headers correctly.
9. Avoid redirecting file downloads to Item landing pages
10. Turn OFF any generation of PDF cover pages
11. As an aside, it's worth noting that OAI-PMH is generally not useful to search engines. OAI-PMH has its own uses, but do not expect search engines to use it.

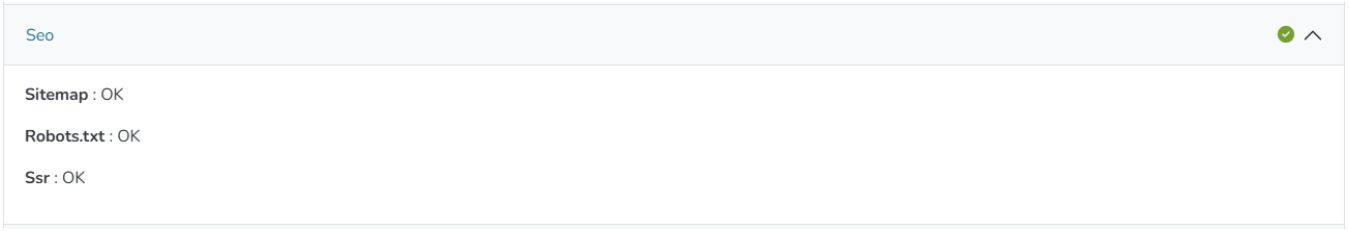
Check SEO Validator status

DSpace now has a basic Search Engine Optimization (SEO) validator which can provide you feedback on how well your site may align with the *some* of these Search Engine Optimization policies.

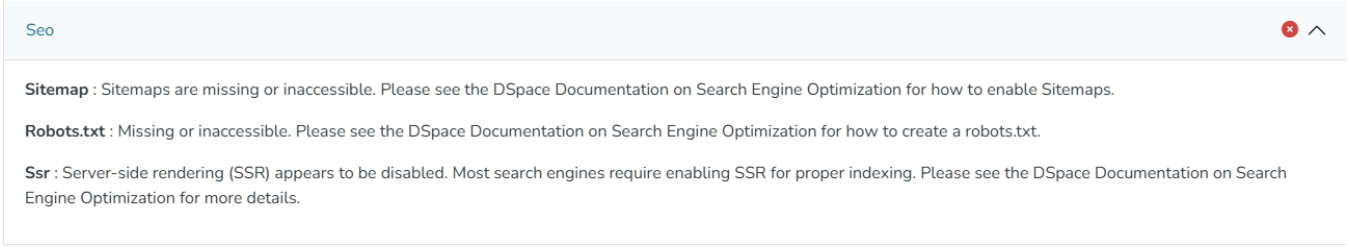
At this time, **this validation tool can only check three things:**

- Ensure the user interface is using server-side rendering
- Ensure the sitemaps feature is enabled and appear to be configured correctly
- Ensure your robots.txt exists and links correctly to the sitemaps (may also detect basic issues with a proxy not passing X-Forwarded headers)

This validation tool can be found in the Admin User Interface on the "Health" page. Look for the section named "SEO". If everything looks good, you'll see a green checkbox similar to this:



If there are issues detected, you'll see a red warning with details on what needs to be addressed.



If issues are detected, you should use the documentation on this wiki page to address the detected issues.

Even if you see a green checkmark on this page, you should **still** review all the Search Engine Optimization guidelines on this page. As noted above, **this validator cannot detect all possible SEO issues,** so manual verification is still required.

Keep your DSpace up to date

We are constantly adding new indexing improvements to DSpace. In order to ensure your site gets all of these improvements, you should strive to keep it up-to-date. For example:

- As of DSpace 9.0, 8.2, and 7.6.4, a basic SEO validator is now provided on the "Health" page.
- As of DSpace 7.0, Sitemaps are enabled by default (see below)
- As of DSpace 5.0, the DSpace robots.txt file now includes references to Sitemaps by default (see https://github.com/DSpace/DSpace/issues/5302) , and also blocks known bad bots (see https://github.com/DSpace/DSpace/issues/5701).
- As of DSpace 4.0, DSpace has provided several enhancements, which were requested by the Google Scholar team. These included providing users (and web indexers) a way to browse content by the date it was added to DSpace (see https://github.com/DSpace/DSpace/issues/4851), ensuring the "dc.date.issued" field is set more accurately (see https://github.com/DSpace/DSpace/issues/4850), and enhancing the logic behind the "citation_pdf_url" HTML <meta> tag (see https://github.com/DSpace/DSpace/issues/4852)
- As of DSpace 1.7, DSpace has improved how its Item-level metadata is made available to Google Scholar. For the 1.7.0 release, the DSpace Developers worked directly with the Google Scholar developers, to ensure DSpace is generating the "citation_*" HTML "<meta>" tags (i.e. Highwire Press tags) that Google Scholar recommends in their Indexing Guidelines.
- As of DSpace 1.5, DSpace has support for sitemaps (both simple HTML pages of links, as well as the sitemaps.org protocol). It also includes item metadata in the HTML HEAD element of item display pages, ensuring that the metadata can be effectively indexed no matter what changes you might have made to your DSpace's layout or style.
- As of DSpace 1.4, DSpace has support for the "if-modified-since" HTTP header. This basically means that if an item (or bitstream therein) has not changed since the last time a search engine's crawler indexed it, that item/bitstream does not have to be re-retrieved, sparing your server.

Additional minor improvements / bug fixes have been made to more recent releases of DSpace.

Ensure your DSpace is visible to search engines

First ensure your DSpace instance is visible, e.g. with: https://www.google.com/webmasters/tools/sitestatus

If your site is not indexed at all, all search engines have a way to add your URL, e.g.:

- Google: http://www.google.com/addurl
- Google Scholar: https://scholar.google.com/intl/en/scholar/inclusion.html
- Bing: https://www.bing.com/webmasters/about

Ensure your proxy is passing X-Forwarded headers to the User Interface

Some HTML tags important for SEO, such as the "citation_pdf_url" tag, require the full URL of your site. The DSpace user interface will automatically attempt to "discover" that URL using HTTP Headers.

Because most DSpace sites use some sort of proxy (e.g. Apache web server or Nginx or similar), this **requires** that the proxy be configured to pass along proper X-Forwarded-* headers, especially X-Forwarded-Host and X-Forwarded-Proto. For example in Apache HTTPD, you can do something like this:

```
# This lets DSpace know it is running behind HTTPS and what hostname is currently used
# (requires installing/enabling mod_headers)
RequestHeader set X-Forwarded-Proto https
RequestHeader set X-Forwarded-Host my.dspace.edu
```

Ensure the user interface is using server-side rendering

In DSpace, Angular's Server Side Rendering (SSR) feature is *enabled by default (only when running in production mode)*. However, it's important to ensure you do *not* disable it in production mode, as most search engine bots cannot index your site if SSR is disabled. Per the frontend Installation instructions, you MUST also be running your user interface in production mode (via either `npm run serve:ssr` or `npm start`).

Because the DSpace user interface is based on Angular.io (which is a javascript framework), you MUST have server-side rendering enabled (which is the default) for search engines to fully index your side. Server-side rendering allows your site to still function even when Javascript is turned *off* in a user's browser. Many web crawlers and bots do not support Javascript (e.g. Google Scholar), so they will only interact with this server-side rendered content.

If you are unsure if server-side rendering (SSR) is enabled, you can check to see if your site is accessible when Javascript is turned **off.** For example, in Chrome, you should be able to do the following:

1. Open your site in the Chrome browser
2. Turn off (disable) Javascript using the Chrome instructions: https://developer.chrome.com/docs/devtools/javascript/disable/
3. Click reload in your browser window to reload your site.
    a. If SSR is enabled, then you will still see your site's contents. You should be able to browse & search the site. (Keep in mind, pages may take longer to load because every request requires SSR.) However, all dynamic menus or actions obviously will not work, as all pages will be static HTML.
    b. If SSR is disabled, then you will see a blank white page. You will not be able to see any content on your site.
4. Don't forget to re-enable Javascript after you are done testing (see link above, or just close that window & reopen a new one)

DSpace use Angular's Server Side Rendering (SSR) for server-side rendering, and it's enabled by default in Production mode via our production environment initialization in `src/environments/environment.production.ts`:

```
// Angular Server Side Rendering Settings
ssr: {
  enabled: true,
  ...
},
```

SSR Performance Tuning

While enabling SSR is extremely important for Search Engine Optimization, it can also be very resource intensive for large pages or highly active sites. Server Side Rendering involves building the entire HTML for the page in Node.js (on your server) before sending the page back to the client /user. Most humans only encounter SSR briefly, when they initially visit your site. However, bots *may only interact* with SSR, especially if they are unable to process Javascript. This is true even for Google Scholar, whose bots will only use SSR generated pages to index your site.

In order to maximum the performance of SSR, by default, DSpace will minimize the pages and Angular components that are processed during server side rendering. You may wish to review the default settings to ensure they are appropriate for your site. See the "Server Side Rendering (SSR) settings" in User Interface Configuration

Ensure the sitemaps feature is enabled

As of DSpace 7, sitemaps are *enabled by default and automatically update on a daily basis.* This is the recommended setup to prefer proper indexing. So, there's nothing you need to do unless you wish to either change their schedule, or disable them.

In the dspace.cfg, the Sitemap generation schedule is controlled by this setting

```
# By default, sitemaps regenerate daily at 1:15am server time
sitemap.cron = 0 15 1 * * ?
```

You can modify this schedule by using the Cron syntax defined at https://www.quartz-scheduler.org/api/2.3.0/org/quartz/CronTrigger.html . Any modifications can be placed in your local.cfg.

If you want to disable this automated scheduler, you can either comment it out, or set it to a single "-" (dash) in your local.cfg

```
# This disables the automatic updates
sitemap.cron = -
```

Again, we **highly recommend** keeping them enabled. However, you may choose to disable this scheduler if you wish to define these in your local system cron settings.

Once you've enabled your sitemaps, they will be accessible at the following URLs:

- HTML Sitemaps: ${dspace.ui.url}/sitemap_index.html
- XML Sitemaps: ${dspace.ui.url}/sitemap_index.xml

So, for example, if your "dspace.ui.url = https://mysite.org" in your "dspace.cfg" configuration file, then the HTML Sitemaps would be at: "http://mysite.org /sitemap_index.html"

By default, the Sitemap URLs also will appear in your UI's `robots.txt` (in order to announce them to search engines):

```
# The URL to the DSpace sitemaps
# XML sitemap is listed first as it is preferred by most search engines
Sitemap: [dspace.ui.url]/sitemap_index.xml
Sitemap: [dspace.ui.url]/sitemap_index.html
```

The generate-sitemaps command

If you wanted to generate your sitemaps manually, you can use a commandline tool to do so.

WARNING: Keep in mind, you do NOT need to run these manually in most situations, as sitemaps are autoupdated on a regular schedule (see documentation above)

```
# Commandline option (run from the backend)
[dspace]/bin/dspace generate-sitemaps
```

This command accepts several options:

Option	meaning
-h --help	Explain the arguments and options.
-s --no_sitemaps	Do not generate a sitemap in sitemaps.org format.
-b -no_htmlmap	Do not generate a sitemap in htmlmap format.

You can configure the list of "all search engines" by setting the value of `sitemap.engineurls` in `dspace.cfg`.

Create a good robots.txt

As of 7.5, DSpace's robots.txt file can be found in the UI's codebase at "src/robots.txt.ejs". This is an "embedded javascript template" (ejs) file, which simply allows for us to insert variable values into the "robots.txt" at runtime. It can be edited as a normal text file.

The trick here is to minimize load on your server, but without actually blocking anything vital for indexing. Search engines need to be able to index item, collection and community pages, and all bitstreams within items – full-text access is critically important for effective indexing, e.g. for citation analysis as well as the usual keyword searching.

If you have restricted content on your site, search engines will not be able to access it; they access all pages as an anonymous user.

Ensure that your robots.txt file is at the top level of your site: i.e. at http://repo.foo.edu/robots.txt, and NOT e.g. http://repo.foo.edu/dspace/robots.txt. If your DSpace instance is served from e.g. http://repo.foo.edu/dspace/, you'll need to add /dspace to all the paths in the examples below (e.g. /dspace/browse-subject).

NEVER BLOCK THESE PATHS

Some URLs can be disallowed without negative impact, but be ABSOLUTELY SURE the following URLs can be reached by crawlers, i.e. DO NOT put these on Disallow: lines, or your DSpace instance might not be indexed properly.

- `/bitstreams`
- `/browse/*` (UNLESS USING SITEMAPS)
- `/collections`
- `/communities`
- `/community-list` (UNLESS USING SITEMAPS)
- `/entities/*`
- `/handle`
- `/items`

Example good robots.txt

DSpace 7 comes with an example robots.txt file (which is copied below). As of 7.5, this file can be found at "src/robots.txt.ejs" in the DSpace 7 UI. This is an "embedded javascript template" (ejs) file, which simply allows for us to insert variable values into the "robots.txt" at runtime. It can be edited as a normal text file.

The highly recommended settings are uncommented. Additional, optional settings are displayed in comments – based on your local configuration you may wish to enable them by uncommenting the corresponding "Disallow:" line.

```
# The URL to the DSpace sitemaps
# XML sitemap is listed first as it is preferred by most search engines
# NOTE: The <%= origin %> variables below will be replaced by the fully qualified URL of your site at runtime.
Sitemap: <%= origin %>/sitemap_index.xml
Sitemap: <%= origin %>/sitemap_index.html

##########################
# Default Access Group
# (NOTE: blank lines are not allowable in a group record)
##########################
User-agent: *
# Disable access to Discovery search and filters; admin pages; processes; submission; workspace; workflow &
profile page
Disallow: /search
Disallow: /admin/*
Disallow: /processes
Disallow: /submit
Disallow: /workspaceitems
Disallow: /profile
Disallow: /workflowitems

# Optionally uncomment the following line ONLY if sitemaps are working
# and you have verified that your site is being indexed correctly.
# Disallow: /browse/*
#
# If you have configured DSpace (Solr-based) Statistics to be publicly
# accessible, then you may not want this content to be indexed
# Disallow: /statistics
#
# You also may wish to disallow access to the following paths, in order
# to stop web spiders from accessing user-based content
# Disallow: /contact
# Disallow: /feedback
# Disallow: /forgot
# Disallow: /login
# Disallow: /register

# NOTE: The default robots.txt also includes a large number of recommended settings to avoid misbehaving bots.
# For brevity, they have been removed from this example, but can be found in src/robots.txt.ejs
```

WARNING: for your additional disallow statements to be recognized under the `User-agent: *` group, they *cannot be separated by white lines* from the declared `user-agent: *` block. A white line indicates the start of a new user agent block. Without a leading user-agent declaration on the first line, blocks are ignored. Comment lines are allowed and will not break the user-agent block.

This is OK:

```
User-agent: *
# Disable access to Discovery search and filters; admin pages; processes
Disallow: /search
Disallow: /admin/*
Disallow: /processes
```

This is **not OK**, as the two lines at the bottom will be completely ignored.

```
User-agent: *
# Disable access to Discovery search and filters; admin pages; processes
Disallow: /search

Disallow: /admin/*
Disallow: /processes
```

To identify if a specific user agent has access to a particular URL, you can use this handy robots.txt tester.

For more information on the robots.txt format, please see the Google Robots.txt documentation.

Ensure Item Metadata appears in the HTML HEAD

Google Scholar Metadata in HTML HEAD

In addition to Dublin Core <meta> tags in the HTML HEAD, DSpace also includes Google Scholar specific metadata fields in each item's HTML display page.

```
<meta property="citation_author" content="Tansley, Robert; Donohue, Timothy"/>
<meta property="citation_title" content="Ensuring your DSpace is indexed" />
```

These meta tags are the "Highwire Press tags" which Google Scholar recommends. If you have heavily customized your metadata fields, or wish to change the default "mappings" to these Highwire Press tags, you may do so by modifying https://github.com/DSpace/dspace-angular/blob/main/src/app/core/metadata/head-tag.service.ts (see for example the "setCitationAuthorTags()" method in that service class)

Much more information is available in the Configuration section on Google Scholar Metadata Mappings.

Avoid redirecting file downloads to Item landing pages

Make sure that you never redirect "direct file downloads" (i.e. users who directly jump to downloading a file, often from a search engine) to the associated Item's splash/landing page. In the past, some DSpace sites have added these custom URL redirects in order to facilitate capturing statistics via Google Analytics or similar.

While these URL redirects may seem harmless, they may be flagged as cloaking or spam by Google, Google Scholar and other major search engines. This may hurt your site's search engine ranking or even cause your entire site to be flagged for removal from the search engine.

If you have these URL redirects in place, it is highly recommended to remove them immediately. If you created these redirects to facilitate capturing download statistics in Google Analytics, you should consider upgrading to DSpace 5.0 or above, which is able to automatically record bitstream downloads in Google Analytics (see https://github.com/DSpace/DSpace/issues/5454) without the need for any URL redirects.

Turn OFF any generation of PDF cover pages

While DSpace offers a PDF Citation Cover Page option, this option may affect your content's visibility in search engines like Google Scholar. Google Scholar (and possibly other search engines) specifically extracts metadata by analyzing the contents of the first page of a PDF. Dynamically inserting a custom cover page can break the metadata extraction techniques of Google Scholar and may result in all or much of your site being dropped from the Google Scholar search engine.

For more information, please see the "Indexing Repositories: Pitfalls and Best Practices" talk from Anurag Acharya (co-creator of Google Scholar) presented at the Open Repositories 2015 conference.

In general, OAI-PMH is not useful to Search Engines

Feel free to support OAI-PMH, but be aware that in general it is not useful for search engines:

- No reliable way to determine OAI-PMH base URL for a DSpace site.
- No standard or predictable way to get to item display page or full text from an OAI-PMH record, making effective indexing and presenting meaningful results difficult.
- In most cases provides only access to simple Dublin Core, a subset of available metadata.
- **NOTE:** Back in 2008, Google officially announced they were retiring support for OAI-PMH based Sitemaps. So, OAI-PMH will no longer help you get better indexing through Google. Instead, you should be using the DSpace 'generate-sitemaps' feature described above.

T

Google Scholar Metadata Mappings

While DSpace 7.0 supports Google Scholar `meta` tags, they are no longer configurable & are currently hardcoded into the User Interface codebase. Configurability may be coming back in a later 7.x release (based on user feedback), see https://github.com/DSpace/dspace-angular/issues/1198

Google Scholar, in crawling sites, prefers Highwire Press tags. This schema contains names which are all prefixed by the string "citation_", and provide various metadata about the article/item being indexed.

In DSpace, there is a mapping facility to connect metadata fields with these citation fields in HTML. In order to enable this functionality, the switch needs to be flipped in dspace.cfg:

```
google-metadata.enable = true
```

Once the feature is enabled, the mapping is configured by a separate configuration file located here:

```
[dspace]/config/crosswalks/google-metadata.properties
```

This file contains name/value pairs linking meta-tags with DSpace metadata fields. E.g…

```
google.citation_title = dc.title
google.citation_publisher = dc.publisher
google.citation_author = dc.author | dc.contributor.author | dc.creator
```

There is further documentation in this configuration file explaining proper syntax in specifying which metadata fields to use. If a value is omitted for a meta-tag field, the meta-tag is simply not included in the HTML output.

The values for each item are interpolated when the item is viewed, and the appropriate meta-tags are included in the HTML head tag, on both the Brief Item Display and the Full Item Display in the UI.

Note: In DSpace 5, the field google.citation_authors was changed to google.citation_author.

Troubleshooting Information

You can quickly get some basic information about the DSpace version and the products supporting it by using the `[dspace]/bin/dspace version` command.

```
$ bin/dspace version
DSpace version:  4.0-SNAPSHOT
  SCM revision:  da53991b6b7e9f86c2a7f5292e3c2e9606f9f44c
    SCM branch:  UNKNOWN
            OS:  Linux(amd64) version 3.7.10-gentoo
Discovery enabled.
Lucene search enabled.
           JRE:  Oracle Corporation version 1.7.0_21
   Ant version:  Apache Ant(TM) version 1.8.4 compiled on June 25 2012
 Maven version:  3.0.4
   DSpace home:  /home/dspace
$
```

To troubleshoot a specific error, see our Troubleshoot an error guide

Validating CheckSums of Bitstreams

1 Checksum Checker
- 1.1 Checker Execution Mode
- 1.2 Checker Results Pruning
- 1.3 Checker Reporting
- 1.4 Cron or Automatic Execution of Checksum Checker
- 1.5 Automated Checksum Checkers' Results
- 1.6 Database Query

Checksum Checker

As of DSpace 9.2, the "-a" argument was renamed to "-i". However, the longer form is still "--handle". See the full list of arguments below.

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
`-i` 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 -i` Checker will only check bitstreams within the Community, Community or the item itself. Example: `[dspace]/bin/dspace checker -i 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!

Note that options `-c`, `-d`, `-b`, `-i`, `-l`, and `-L` are mutually exclusive. For example, it is not possible limit the running time or bitstream count of looping modes.

Checker Results Pruning

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

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

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

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

Checker Reporting

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

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

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

Cron or Automatic Execution of Checksum Checker

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

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

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

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

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

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

(This command should appear on a single line).

Automated Checksum Checkers' Results

Optionally, you may choose to receive automated emails listing the Checksum Checkers' results to the email address specified in the `mail.admin` configu ration property. Schedule it to run **after** the Checksum Checker has completed its processing (otherwise the email may not contain all the results). As of DSpace 4.1, an email is only generated if the selected report contains at least one bitstream needing attention.

Command used:	`[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 `--unchecked`	Send the Unchecked bitstream report.
`-n` or `--not-processed`	Send E-mail report for all bitstreams set to longer be processed for today.
`-h` or `--help`	Help

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

Cron. Follow the same steps above as you would running checker in cron. Change the time but match the regularity. Remember to schedule this **after** Checksum Checker has run. For an example cron setup, see Scheduled Tasks via Cron.

Database Query

A query like the following can be used to check the results of the checker (Postgres):

```
SELECT *
FROM checksum_history
WHERE date_trunc('day', process_start_date) = CURRENT_DATE
AND result != 'CHECKSUM_MATCH'
AND result != 'BITSTREAM_MARKED_DELETED';
```

Example of a more detailed query:

```
SELECT
    ch.process_start_date,
    ch.process_end_date,
    ch.result,
    ch.checksum_expected,
    ch.checksum_calculated,
    b.bitstream_id,
    bfr.short_description,
    b.store_number,
    substring(b.internal_id for 2) || '/' || substring(b.internal_id from 3 for 2) || '/' || substring(b.
internal_id from 5 for 2) || '/' || b.internal_id AS bitstream_path,
    hi.handle AS item_handle,
    hc.handle AS collection_handle
FROM checksum_history ch
JOIN bitstream b
ON ch.bitstream_id = b.uuid
JOIN bitstreamformatregistry bfr
ON b.bitstream_format_id = bfr.bitstream_format_id
LEFT JOIN bundle2bitstream bb
ON b.uuid = bb.bitstream_id
LEFT JOIN item2bundle ib
ON bb.bundle_id = ib.bundle_id
LEFT JOIN item i
ON ib.item_id = i.uuid
LEFT JOIN handle hi
ON i.uuid = hi.resource_id
AND hi.resource_type_id = 2
LEFT JOIN handle hc
ON i.owning_collection = hc.resource_id
AND hc.resource_type_id = 3
WHERE ch.result != 'CHECKSUM_MATCH'
AND date_trunc('day', process_start_date) = CURRENT_DATE
ORDER BY ch.check_id DESC;
```

DSpace Development

This section contains information on how to modify, extend and customize the DSpace source code.

- Advanced Customisation
- Batch Processing
- Curation Tasks
- Development Tools Provided by DSpace
- Menu providers (WIP)
- REST API
- Services to support Alternative Identifiers
- User Interface Design Principles & Accessibility
- Workflow

Advanced Customisation

If you are looking for ways to override specific classes or resources in DSpace (specifically in the backend), this page provides a guide for how to do so.

1 Additions module
2 Server Webapp Overlay
3 Rest (Deprecated) Webapp Overlay

If you are looking for ways to customize the content or layout of the user interface, see User Interface Customization.

Additions module

Location: `[dspace-source]/dspace/modules/additions/`

This module may be used to store dspace-api changes, custom plugins, etc. Classes placed in `[dspace-source]/dspace/modules/additions` will override those located in the `[dspace-source]/dspace-api`

This module may be used to override classes across *all* webapps located in `[dspace-source]/dspace/modules/` directory, as well as in the command line interface. Therefore, this modules is for global overrides only. If you have overrides specific to a single webapp, use the "Maven WAR Overlays" option below.

Server Webapp Overlay

Location: `[dspace-source]/dspace/modules/server/`

This module overlay directory allows you to override any classes, resources or files available (by default) in the Server Webapp. This includes overriding files of any of the following source directories:

- `[dspace-source]/dspace-oai/` (Bundled into the Server Webapp as a JAR)
- `[dspace-source]/dspace-rdf/` (Bundled into the Server Webapp as a JAR)
- `[dspace-source]/dspace-server-webapp/` (The Server Webapp itself)
- `[dspace-source]/dspace-sword/` (Bundled into the Server Webapp as a JAR)
- `[dspace-source]/dspace-swordv2/` (Bundled into the Server Webapp as a JAR)

Java classes place in `[dspace-source]/dspace/modules/server/` will override classes (of the same path/name) in any of the above modules.

You can also override resources (i.e. any files under a /src/main/resources/ directory) which are embedded in one of the JARs by putting them under `[dspace-source]/dspace/modules/server/src/main/resources/`. For example, to override the "`[dspace-source]/dspace-oai/src/main/resources/templates/index.twig.html`" file embedded in the `dspace-oai.jar`, you'd place your own version at `[dspace-source]/dspace/modules/server/src/main/resources/templates/index.twig.html`. This results in the resource/file being copied over into the `WEB-INF/classes/` subdirectory of the "server" webapp, and in that location it will override any file of the same name embedded in a JAR (per Servlet Spec 3.0).

Rest (Deprecated) Webapp Overlay

Location: `[dspace-source]/dspace/modules/rest`

If you have chosen to install the deprecated REST API v6 webapp, you can similar override any classes/files of that separate webapp by just placing those files in the `[dspace-source]/dspace/modules/rest/` directory

DSpace Service Manager

1 Introduction
2 Configuration
    2.1 Configuring Addons to Support Spring Services
    2.2 Configuration Priorities
        2.2.1 Configuring a new Addon
            2.2.1.1 Addon located as resource in jar
            2.2.1.2 Addon located in the [dspace]/config/spring directory
        2.2.2 The Core Spring Configuration
        2.2.3 Utilizing Autowiring to minimize configuration complexity.
    2.3 Accessing the Services Via Service Locator / Java Code
3 Architectural Overview
    3.1 Service Manager Startup in Webapplications and CLI
4 Tutorials

Introduction

The DSpace Spring Service Manager supports overriding configuration at many levels.

Configuration

Configuring Addons to Support Spring Services

Configuring Addons to support Spring happens at two levels. Default Spring configuration is available in the DSpace JAR or WAR resources directory and allows the addon developer to inject configuration into the service manager at load time. The second level is in the deployed [dspace]/config/spring directory where configurations can be provided on a addon module by addon module basis.

This latter method requires the addon to implement a SpringLoader to identify the location to look for Spring configuration and a place configuration files into that location. This can be seen inside the current [dspace-source]/config/modules/spring.cfg

Configuration Priorities

The ordering of the loading of Spring configuration is the following:

1. configPath = "spring/spring-dspace-applicationContext.xml" relative to the current classpath
2. addonResourcePath = "classpath*:spring/spring-dspace-addon-*-services.xml" relative to the current classpath
3. coreResourcePath = "classpath*:spring/spring-dspace-core-services.xml" relative to the current classpath
4. Finally, an array of SpringLoader API implementations that are checked to verify "config/spring/module" can actually be loaded by its existence on the classpath. The configuration of these SpringLoader API classes can be found in dspace.dir/config/modules/spring.cfg.

Configuring a new Addon

There are 2 ways to create a new Spring addon: a new Spring file can be located in the resources directory or in the configuration [dspace]/config/spring directory. A Spring file can also be located in both of these locations but the configuration directory gets preference and will override any configurations located in the resources directory.

Addon located as resource in jar

In the resources directory of a certain module, a Spring file can be added if it matches the following pattern: "spring/spring-dspace-addon-*-services.xml". An example of this can be found in the dspace-discovery-solr block in the DSpace trunk. (`spring-dspace-addon-discovery-services.xml`) Wherever this jar is loaded in a Maven module, 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

The reason why there is a separate directory is that if a service cannot be loaded, 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 "spring.springloader.modules" property located in the [dspace]/config/modules/spring.cfg.
The Spring service manager will check this property to ensure that only the interface implementations which it can find the class for are loaded in.

By doing this way we give some flexibility to the developers so that they can always create their own Spring modules and then Spring will not crash when it can't find a certain class.

The Core Spring Configuration

Utilizing Autowiring to minimize configuration complexity.

Please see the following tutorials:

- DSpace Spring Services Tutorial
- The TAO of DSpace Services

Accessing the Services Via Service Locator / Java Code

Please see the following tutorials:

- DSpace Spring Services Tutorial
- The TAO of DSpace Services

Architectural Overview

Please see Architectural Overview here: DSpace Services Framework

Service Manager Startup in Webapplications and CLI

Please see the DSpace Services Framework

Tutorials

Several good Spring / DSpace Services Tutorials are already available:

- DSpace Spring Services Tutorial
- The TAO of DSpace Services

Batch Processing

In the current DSpace design, the database transactions are in most of the cases relatively long: from Context creation to the moment the Context is completed. Especially when doing batch processing, that transaction can become very long. The new data access layer introduced in DSpace 6 which is based on Hibernate has built-in cache and auto-update mechanisms. But these mechanisms do not work well with long transactions and even have an exponentially adverse-effect on performance.

Therefore we added a new method `enableBatchMode()` to the DSpace Context class which tells our database connection that we are going to do some batch processing. The database connection (Hibernate in our case) can then optimize itself to deal with a large number of inserts, updates and deletes. Hibernate will then not postpone update statements anymore which is better in the case of batch processing. The method `isBatchModeEnabled()` lets you check if the current Context is in "batch mode".

When dealing with a lot of records, it is also important to deal with the size of the (Hibernate) cache. A large cache can also lead to decreased performance and eventually to "out of memory" exceptions. To help developers to better manage the cache, a method `getCacheSize()` was added to the DSpace Context class that will give you the number of database records currently cached by the database connection. Another new method `uncacheEntity(ReloadableEntity entity)` will allow you to clear the cache (of a single object) and free up (heap) memory. The `uncacheEntity()` method may be used to immediately remove an object from heap memory once the batch processing is finished with it. Besides the `uncacheEntity()` method, the `commit()` method in the DSpace Context class will also clear the cache, flush all pending changes to the database and commit the current database transaction. The database changes will then be visible to other threads.

BUT `uncacheEntity()` and `commit()` come at a price. After calling this method all previously fetched entities (hibernate terminology for database record) are "detached" (pending changes are not tracked anymore) and cannot be combined with "attached" entities. If you change a value in a detached entity, Hibernate will not automatically push that change to the database. If you still want to change a value of a detached entity or if you want to use that entity in combination with attached entities (e.g. adding a bitstream to an item) after you have cleared the cache, you first have to reload that entity. Reloading means asking the database connection to re-add the entity from the database to the cache and get a new object reference to the required entity. From then on, it is important that you use that new object reference. To simplify the process of reloading detached entities, we've added a `reloadEntity(ReloadableEntity entity)` method to the DSpace Context class with a new interface `ReloadableEntity`. This method will give the user a new "attached" reference to the requested entity. All DSpace Objects and some extra classes implement the `ReloadableEntity` interface so that they can be easily reloaded.

Examples on how to use these new methods can be found in the `IndexClient` class. But to summarize, when batch processing it is important that:

1. You put the Context into batch processing mode using the method:

```
boolean originalMode = context.isBatchModeEnabled();
context.enableBatchMode(true);
```

2. Perform necessary batch operations, being careful to call `uncacheEntity()` whenever you complete operations on each object. Alternatively, you can `commit()` the context once the object cache reaches a particular size (see `getCacheSize()`). Remember, once an object is "uncached", you will have to reload it (see `reloadEntity()`) before you can work with it again:

```
final Iterator<Item> itemIterator = itemService.findByCollection(context, collection);

// Loop through all items
while (itemIterator.hasNext()) {
  // Get access to next Item
  Item item = itemIterator.next();

  ... do something with Item ...

  // To prevent memory issues, discard Item from the cache after processing
  context.uncacheEntity(item);
}

// Remember: calling commit() will decache all objects
context.commit();

// So, if you need to reuse your Collection *post* commit(), you'd have to reload it
Collection collection = context.reloadEntity(collection);
```

3. When you're finished with processing the records, you put the context back into its original mode:

```
context.enableBatchMode(originalMode);
```

Curation Tasks

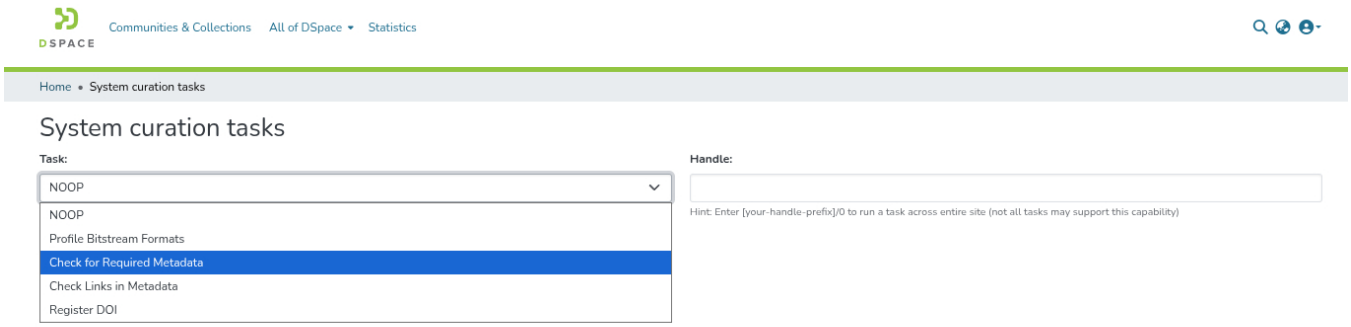
1 What is a Curation Task?
2 Curation Task Configuration (curate.cfg)
3 Writing your own tasks
4 Task Output and Reporting
    4.1 Status Code
    4.2 Result String
    4.3 Reporting Stream
    4.4 Accessing task output in calling code
5 Task Properties
6 Task Annotations
7 Scripted Tasks
    7.1 Interface
        7.1.1 performDso() vs. performId()

This documentation provides a guide for how to programmatically create Curation Tasks. For more information configuring Curation Tasks, see the Curation System section of the documentation

What is a Curation Task?

A **Curation Task** is a pluggable unit of backend logic in DSpace used to **inspect**, **validate**, **report on**, or **modify** repository content. Typically they are used for batch maintenance, metadata validation, content transformation, and repository-wide cleanup or enrichment operations. Tasks operate on DSpace objects (Items, Collections, Communities, or the entire Site) and can be executed from the **command line**, the **administrative UI**, or as a part of **automated workflows**.

Technically, a Curation Task is a Java class that implements `org.dspace.curate.CurationTask` (or `org.dspace.curate.ScriptedTask` for non-Java JVM implementations, see Curation System - Scripted Tasks for more information). Each task returns a standardized status code and may also produce result messages or detailed reports.



Curation task in the administrative UI of DSpace

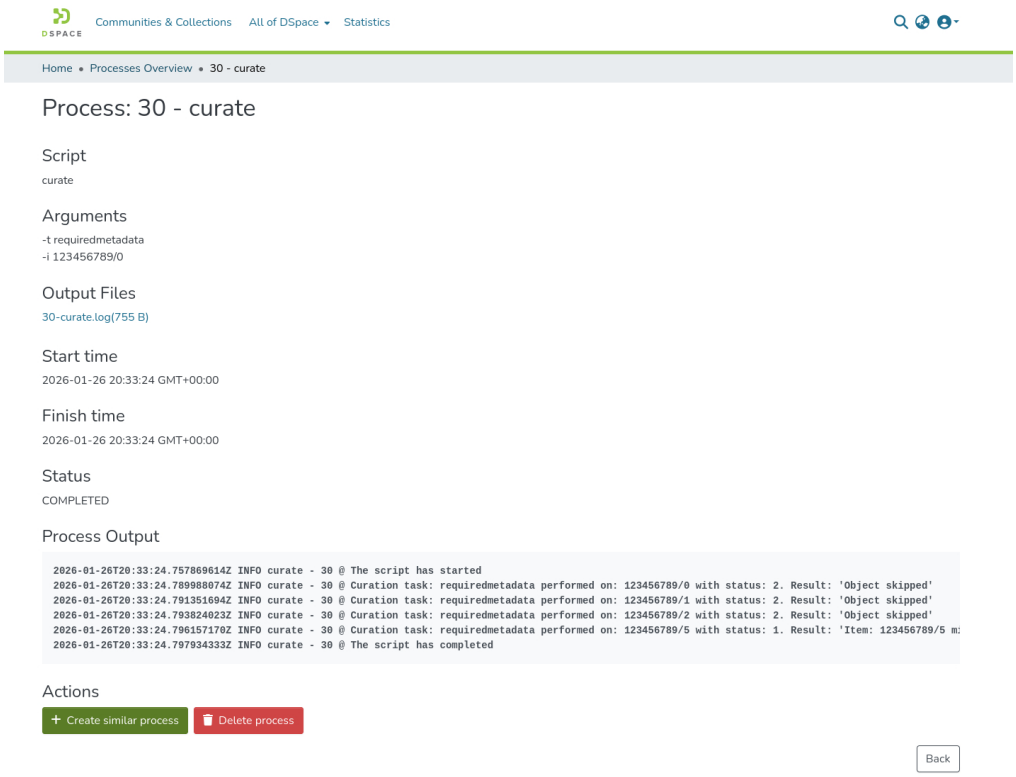
Curation Task Configuration (curate.cfg)

Available Curation Tasks are registered in DSpace backend's `[dspace]/config/modules/curate.cfg` using the DSpace plugin system:

```
plugin.named.org.dspace.curate.CurationTask = org.dspace.ctask.general.NoOpCurationTask = noop
plugin.named.org.dspace.curate.CurationTask = org.dspace.ctask.general.ProfileFormats = profileformats
#plugin.named.org.dspace.curate.CurationTask = org.dspace.ctask.general.ClamScan = vscan
#...
```

Only tasks listed and uncommented in this file are available for execution. Commenting out an entry disables it system-wide without requiring code changes.

The logical task name (e.g. "noop", "profileformats") is what administrators use when invoking the task via the UI or CLI; not the Java class name.



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.

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 is returned to CS by any of a task's `perform` methods. The complete list of values, defined in `Curator`, is:

value	symbol	meaning

-3	CURATE_NOTASK	CS could not find the requested task
-2	CURATE_UNSET	task did not return a status code because it has not yet run
-1	CURATE_ERROR	task could not be performed
0	CURATE_SUCCESS	task performed successfully
1	CURATE_FAIL	task performed, but failed
2	CURATE_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 in Curation System) for display.

Result String

The task may set a string indicating details of the outcome:

```
curator.setResult("Item " + item.getID() + " was painted " + color);
```

CS does not interpret or assign result strings; the task does it. A task may choose not to assign a result, but the "best practice" for tasks is to assign one whenever possible. Code which invokes `Curator.getResult()` may use the result string for display or any other purpose.

Reporting Stream

For very fine-grained information, a task may write to a *reporting* stream. Unlike the result string, there is no limit to the amount of data that may be pushed to this stream.

```
curator.report("Lorem ipsum dolor sit amet,\n");
curator.report("consectetur adipiscing elit,\n");
curator.report("sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.\n");
```

Accessing task output in calling code

The status code, reporting stream, and the result string are accessed (or set) by methods on the Curator object:

```
Curator curator = new Curator();
curator.setReporter(new OutputStreamWriter(System.out));
curator.addTask("vscan").curate(coll);
int status = curator.getStatus("vscan");
String result = curator.getResult("vscan");
```

Task Properties

Task code may configure itself using ConfigurationService in the normal manner, or by the use of "task properties". See Curation System - Task Properties for discussion of the issues for which task properties were invented. Any code which extends AbstractCurationTask has access to its configured task properties.

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);
```

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.

Scripted Tasks

DSpace 1.8 introduced 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. See Curation System - Scripted Tasks for information on configuring and running scripted tasks.

Interface

Scripted tasks must implement a slightly different interface than the **CurationTask** interface used for Java tasks. The appropriate interface for scripting tasks is **ScriptedTask** and has the following methods:

```
public void init(Curator curator, String taskId) throws IOException;
public int performDso(DSpaceObject dso) throws IOException;
public int performId(Context ctx, String id) throws IOException;
```

The difference is that `ScriptedTask` has separate `perform` methods for DSO and identifier. The reason for that is that some scripting languages (e.g. Ruby) don't support method overloading.

performDso() vs. performId()

You may have noticed that the `ScriptedTask` interface has both `performDso()` and `performId()` methods, but only performDso is ever called when curator is launched from command line.

There are a class of use-cases in which we want to construct or create new DSOs (DSpaceObject) given an identifier in a task. In these cases, there may be no live DSO to pass to the task.
You actually **can** get curation system to call `performId()` if you queue a task then process the queue - when reading the queue all CLI has is the handle to pass to the task.

Curation tasks in Jython

As mentioned in the "Scripted Tasks" chapter of Curation Tasks, you can write your curation tasks in several languages, including Jython (a flavour of Python running on JVM).

Instructions are outdated and unproven in DSpace 7.x

Setting up scripted tasks in Jython

1. Download the latest Jython installer jar (e.g. jython-installer-2.7.1.jar) from http://www.jython.org/downloads.html
2. Get `jython.jar` and the `Lib` directory.
    a. either unzip the installer jar:
    ```
    unzip -d [dspace]/lib/ jython-installer-2.7.1.jar jython.jar Lib/
    unzip -d [dspace]/webapps/server/WEB-INF/lib/ jython-installer-2.7.1.jar jython.jar Lib/
    ```
    b. or use it to install Jython:
    ```
    java -jar jython-installer-2.7.1.jar --console
    ```
    Note: Installation location doesn't matter, this is not necessary for DSpace. You can safely delete it after you retrieve `jython.jar` and `Lib`.
3. Install Jython to DSpace classpaths (step 2a already did this for you):
    a. The goal is to put `jython.jar` and the jython `Lib/` directory into **every** DSpace classpath you intend to use, so it must be installed in **both** `[dspace]/lib` and the webapp that deploys to Tomcat (if you want to run from the UI) - `[dspace]/webapps/server/WEB-INF/lib/`. There are no special maven/pom extensions - just copy in the jar and `Lib/`.
    b. You **can** use symlinks if you wish as long as allowLinking (Tomcat <=7, Tomcat 8) is set to true in that context's configuration. However, be warned that Tomcat documentation lists allowLinking="true" as a possible security concern.
    c. Note: Older versions of Jython mention the need for jython-engine.jar to implement JSR-223. Don't worry about that, new Jython versions, e.g. 2.7.1 don't require this.
4. Configure the curation framework to be aware of your new task(s):
    a. set up the location of scripted tasks in the curation system. This means simply adding a property to `[dspace]/config/modules/curate.cfg`:
    ```
    script.dir=${dspace.dir}/ctscripts
    ```
    b. in this directory, create a text file named "`task.catalog`". This is a Java properties file where lines beginning with '#' are comments. Add a line for each task you write. The syntax is following:
    ```
    # logical task name = script engine name|file name|constructor invocation
    mytask=python|mytask.py|MyTask()
    ```
    Notes:
        - **don't** put spaces around the pipe character or you'll get an error similar to this one:
        `ERROR org.dspace.curate.TaskResolver @ Script engine: 'python ' is not installed`
        - The "script engine name" is whatever name (or alias) jython registers in the JVM. You can use both "python" and "jython" as engine name (tested on jython 2.7.1).
        - The logical task name can't conflict with existing (java) task names, but otherwise any single-word token can be used.
        - The file name is just the script file name in the `script.dir` directory
        - "constructor invocation" is the language specific way to create an object that implements the task interface - it's ClassName() for Python
    c. If you want pretty names in the UI, configure other curate.cfg properties - see "`ui.tasknames`" (or groups etc)
5. Write your task.
    In the directory configured above, create your task (with the name configured in "`task.catalog`").
    The basic requirement of any scripted task is that it implements the ScriptedTask Java interface.
    So for our example, the `mytask.py` file might look like this:

```
from org.dspace.curate import ScriptedTask

class MyTask(ScriptedTask):
        def init(self, curator, taskName):
                print "initializing with Jython"

        def performDso(self, dso):
                print "perform on dso"
                return 0

        def performId(self, context, id):
                print "perform on id %s" % (id)
                return 0
```

6. Invoke the task.
   You can do this the same way you would invoke any task (from command line, in the admin UI, etc). The advantage of scripting is that you do not need to restart your servlet container to test changes; each task's source code is reloaded when you launch the task, so you can just put the updated script in place.
   Example of invocation from command line:

   ```
   [dspace]/bin/dspace curate -t mytask -i 123456789/123 -r -
   ```

   Note: "`-r -`" means that the script's standard output will be directed to the console. You can read more details in the "On the command line" chapter of the Curation Tasks page.

See also

- Curation Tasks page in the official documentation
- Nailgun - for speeding up repeated runs of a dspace command from the command line
  Note: since DSpace 4.0, there's a solution for running dspace CLI commands in batch: Executing streams of commands
- Jython webapp for DSpace - general purpose (not curation task) webapp written in Jython, optionally with access to DSpace API

Development Tools Provided by DSpace

Date parser tester

Some parts of DSpace use a custom date/time parser (`org.dspace.util.MultiFormatDateParser`) which is driven by a table of regular expressions, so it can match any of a variety of formats. The table is found in `config/spring/api/discovery-solr.xml`. To test new and altered rules, you can use the DSpace command line tool's `validate-date` command. You can simply pass it a date/time string on the command line (`dspace validate-date` 01-01-2015). You can pipe a stream of strings to be validated, one per line (`dspace validate-date < test.data`). Or you can have it prompt you for each string to be tested (`dspace validate-date`).

Menu providers (WIP)

This documentation is still a work-in-progress.
PR: https://github.com/DSpace/dspace-angular/pull/3994

The menu sections will be generated by multiple “Providers” which will each add a section or list of related sections. These providers will then be added to the list of providers per menu defined in the `app.menus.ts` file.

```
export const MENUS = buildMenuStructure({
  [MenuID.PUBLIC]: [
    CommunityListMenuProvider,
    BrowseMenuProvider,
    StatisticsMenuProvider,
  ],
  [MenuID.ADMIN]: [
    NewMenuProvider,
    EditMenuProvider,
    ...
    SystemWideAlertMenuProvider,
  ],
  [MenuID.DSO_EDIT]: [
    DsoOptionMenuProvider.withSubs([
      SubscribeMenuProvider.onRoute(
        MenuRoute.SIMPLE_COMMUNITY_PAGE,
        MenuRoute.SIMPLE_COLLECTION_PAGE,
      ),
      ...
      ClaimMenuProvider.onRoute(
        MenuRoute.SIMPLE_ITEM_PAGE,
        MenuRoute.FULL_ITEM_PAGE,
      ),
    ]),
  ],
});
```

This approach makes it way more intuitive to see what kind of sections there are and to add new sections in between already existing ones.

This new version fully removes the old way of adding route based data (e.g. statistics) and replaces it with a new way where you can easily add routes to specified providers and/or retrieve route data in the provider to be used for its sections. It also adds support to separately define a parent sections with various child sections. The latter will be mostly useful for creating sections on the DSO pages where you can now separately define the parent section through a provider and then add child providers for the various different sections.

For example, in the current implementation all the buttons on the DSO pages have been added to a single parent parent section resulting in a single button on the DSO pages with the relevant sections in the dropdown. In case sections need to be moved, the provider can simply be moved to a different location such as before the provider responsible for providing the button (`DsoOptionMenuProvider`).

Finally, since the menus have been moved from the resolvers to the `InitService`, the menus will not delay the page from loading when the menu sections need some time. The menus will simply appear when they are fully loaded. To prevent "blinking" when changing pages, the sections that need to be added/removed are all done at the same time.

Provider/section properties and helper providers

Provider properties

The following properties can be set on the level of a provider when creating a new one:

- `shouldPersistOnRouteChange` - When true, the sections of this provider will be retained when swapping routes. This is useful for route independent menus such as the community-list link, browse options and admin sidebar menu sections
    - This is by default set on `true`.
    - The default behaviour is overwritten when the `onRoute` static method is used, or the `AbstractRouteContextMenuProvider` is extended, then it is automatically set to `false` since the section are dependent on the route. See below for more information.
- `alwaysRenderExpandable` - When true, this will render the sections of this provider as something that is expected to be expandable and have children. When a section of this provider has no children, it will be hidden. When false, the section will always be rendered, even if no children are visible.
    - This is by default set to `false`.
    - The default value will be set to `true` when `AbstractExpandableMenuProvider` is extended and will be applied to the top section for this provider.

    - Example of a manual overwrite can be found in `DsoOptionMenuProvider`. This is done to ensure the "option" button is hidden when no entries are present in the dropdown.

Section properties

The sections still contain the same configuration properties as before the refactor, however a lot of the properties will be already set by the providers. There are however some parts that still need to be provided per sections:

- `visible` - Whether the section should be visible. This is most often determined based on an authorization call
- `model` - The model on how the section needs to be rendered (text, link, function etc)
- `icon` (optional) - The icon to be rendered with the section if necessary.

The following properties can also be set on the sections to overwrite the default behaviour added by the provider:

- `active` - This is by default set to `true` and indicates the section is active
- `shouldPersistOnRouteChange` - By default this will be set to the same value as the corresponding property in the provider. It can be set on the section in case the provider behaviour needs to be overwritten.
- `alwaysRenderExpandable` - By default this will be set to the same value as the corresponding property in the provider. It can be set on the section in case the behaviour needs to be overwritten.

The following two properties can also be set but should be avoided since these are set automatically and may cause confusion when overwritten manually:

- `id` - This will be set automatically when the providers are processed based on the menu and its index in the list of providers and its index in the list of sections in the provider.
    - The `id` can be manually set when needed to interact with this menu, however, you can also use one of the methods present to retrieve the automatic id instead by using the `getAutomatedSectionId` method which takes the relative index of the section inside the provider as argument. When extending the `AbstractExpandableMenuProvider` provider, the following two methods can be used instead: `getAutomatedSectionIdForTopSection` and `getAutomatedSectionIdForSubsection` where the latter also takes the relative index of the sub section inside the provider. An example of where one of these methods is used, is in the
- `parentID` - This will be set automatically when the `AbstractExpandableMenuProvider` or the `withSubs` method is used. It is advised to use one of these two options to add parent and child menu sections. In other cases, the parentID can be set in combination with the id to make sure they match, however, great care needs to be taken when doing this.

Helper providers

To make it easier to create new providers, several "helper" providers have been created that will pre-configure some aspects of the provider and two static methods can be used on any type of provider to change the behaviour of the provider.

Static methods

The following two methods can be added to a provider when defining it in the `app.menus.ts` file:

`onRoute`
To allow people to easily determine routes on which they want to add the sections from this provider, a new route ID has been added to various routes. All of the existing route IDs can be found in `MenuRoute` . These IDs can then be added to the route in the following way:

```
{
  path: '',
  component: ThemedCommunityPageComponent,
  pathMatch: 'full',
  data: {
    menuRoute: MenuRoute.SIMPLE_COMMUNITY_PAGE,
  },
}
```

On a provider, the `onRoute` static method can be used which takes a list of route IDs as argument and will only add menus on the routes which contain the provided route ID in its route data.

The `shouldPersistOnRouteChange` will be automatically set to `false` since the sections of this provider depends on the route.

`withSubs`
This method takes a list of child providers as argument.

When this method is called on a provider, this provider will act as a parent provider to all the child providers that have been provided as argument. The `parentID` will be automatically set for all the child providers.

When using the `withSubs` method, avoid adding multiple sections to the parent provider since this can cause unnecessary confusion. In case multiple sections have been added to the parent, the sub provider sections will be added to the first section

Note:
These two methods are mutually exclusive. In case you would want to combine the `withSubs` and `onRoute`, the `onRoute` method can instead be used on the child providers. If `alwaysRenderExpandable` is set to `true` for the parent provider, the section will be hidden when no child sections are visible!

Extendable helper providers

The following abstract providers have been added that can be extended to more easily create certain menu section structures.

The following abstract providers exist:

`AbstractRouteContextMenuProvider<T>`
This provider contains some extra support to retrieve information from the route through the use of additional (abstract) methods:

- `getRouteContext` should be implemented to retrieve the object of type <T> you want from the route.
- `getSectionsForContext` should be implemented to create the sections. This method will already provide the object retrieved by `getRouteContext` to make the section creation more easy.
- `isApplicable` is a method that can be used to further distinguish whether the sections of this provider should be created for the object of type <T>. This method returns `true` by default.

`DSpaceObjectPageMenuProvider`
This provider extends the `AbstractRouteContextMenuProvider` with <T> being a `DSpaceObject`. It can be extended to easily create sections for various DSpace objects or a specific kind of DSpace objects.

The provider already implemented the `getRouteContext` to retrieve the DSpace object from the route data (`route.data.dso`). When implementing the `getSectionsForContext` method, the DSpace object present in the route object will be provided so it can be used to easily create the sections.

The provider extends `isApplicable` to return true only when a DSpace object was retrieved from the route. This can be further extended though in case only a specific type or entity needs to be supported.

`AbstractExpandableMenuProvider`
This provider contains support to easily create an expandable section with subsections.

It adds two abstract methods to be implemented:

- `getTopSection` - Should return the menu section to act like the top section
- `getSubSections` - Should return the list of sub menu sections for the parent

The provider will then combine these sections and assign the correct parent IDs to the children. To manually interact with the ids of the parent and the child sections, the following two methods have been added to retrieve the sections:

- `getAutomatedSectionIdForTopSection` - Retrieves the ID of the top section
- `getAutomatedSectionIdForSubsection` - Retrieves the ID for a child section. If no argument is provided, the ID of the first child is retrieved. If a different section ID needs to be retrieved, a relative index of the child section can be provided.

How to migrate

The main idea of adding new sections to the existing menus is to separate the sections into different providers and then list these providers in the `app.menus.ts` file. Multiple sections can be provided by a single provider, but this should only be done if those sections are related to each other and are easy to add to the same provider, e.g. `NewMenuProvider` which contains the "New" section in the admin side bar and all its children. However, the various DSO pages have their menus split over separate providers, using the `.withSubs` method. In case more menus need to be added in the future, they can be easily added to the already existing dropdown, or added as separate buttons.

Below the practical steps are explained on how to migrate the sections.

Step 1

Choose how you want to migrate a section or a couple of sections that belong together, what helper provider you want to start from and what static methods you want to use if any.

For example:

Do you have just one section without route dependent info?
-> Extend the basic `AbstractMenuProvider` class. See `SystemWideAlertMenuProvider` for an example of this.

Do you have a simple expandable list without route dependent info?
-> Extend the `AbstractExpandableMenuProvider`. See `ImportMenuProvider` for an example of this.

Do you have a more complex expandable list with route dependent sections that need to be displayed on the DSpace object pages?
-> Create a separate parent provider extending the `DSpaceObjectPageMenuProvider`. Use the `.withSubs` method on this provider and create separate child providers that can also extend `DSpaceObjectPageMenuProvider` provider. On the child providers, the `onRoute` method can be used to determine on what routes these should be present.
See the `DsoOptionMenuProvider` and its children for an example of this.

Step 2

After having determined what abstract provider to start from, you can check if you need to overwrite some of the default properties of the provider that will be used for all sections added in this provider (`shouldPersistOnRouteChange` and `alwaysRenderExpandable`).

For example, in the `DsoOptionMenuProvider`, the `alwaysRenderExpandable` is set to true since the parent menu should be hidden when none of the children are present on the route.

Next, the relevant "getSections" method needs to be implemented (e.g. `getSections`, `getSectionsForContext`, `getTopSection` and `getSubSections`, ...)

These methods should return and observable of section(s). It is advised to only add the minimal information for the sections since a lot of the information will be automatically set.

The following properties are required:

- `visible`
- `model`
- `icon`

The following can be set if the behaviour of the provider needs to be overwritten on section level:

- `alwaysRenderExpandable`
- `shouldPersistOnRouteChange`

Avoid manually setting the `parentID` and `id` properties!
The `parentID` will be set automatically when using `.withSubs` or `AbstractExpandableMenuProvider`. Use the `getAutomatedSectionId` or `getAutomatedSectionIdForTopSection`/`getAutomatedSectionIdForSubsection` methods when interactions are needed with a specific menu.
See `ClaimMenuProvider` for an example of the usage of `getAutomatedSectionId`.

Step 3

The final step is to add the new provider to the list of providers in the `app.menus.ts` file. The position of the provider will determine the order of display in the actual menu.

REST API

- Overview
- REST Contract / Documentation
- Finding which REST API Endpoint to use
- REST Configuration
    - REST Spring Boot Configuration
- Technical Design
- DSpace Demo REST-API HAL Browser
- DSpace Python REST Client Library

Overview

The REST API for DSpace is provided as part of the "server" webapp (`[dspace-source]/dspace-server-webapp/`). It is available on the `/api/` subpath of that webapp (i.e. `${dspace.server.url}/api/`), though a human browseable/searchable interface (using the HAL Browser) is also available at the root path (i.e. `${dspace.server.url}`).

The REST API only responds in JSON at this time.

REST Contract / Documentation

The REST Contract is maintained in GitHub at https://github.com/DSpace/RestContract/blob/main/README.md

This contract provides detailed information on how to interact with the API, what endpoints are available, etc. All features/capabilities of the DSpace UI are available in this API.

Finding which REST API Endpoint to use

When first trying the use the DSpace REST API, it can be difficult to determine where to begin. This brief guide provides a few hints on where to start.

First, it's important to be aware that **every single action in the User Interface can be done in the REST API.** So, if you can achieve something in the User Interface, then it's also possible to do via the REST API.

A few key endpoints to be aware of:

- Authentication: https://github.com/DSpace/RestContract/blob/main/authentication.md
- CSRF Tokens (required for all non-GET requests): https://github.com/DSpace/RestContract/blob/main/csrf-tokens.md
- Submission via REST API: https://github.com/DSpace/RestContract/blob/main/submission.md
- Search via REST API (across all object types): https://github.com/DSpace/RestContract/blob/main/search-endpoint.md
    - Some endpoints also provide a "/search" subpath: https://github.com/DSpace/RestContract/blob/main/search-rels.md

How to find which endpoint(s) to use for any feature or action:

1. Open the DSpace User Interface in your browser window. You can even use our Demo Site (https://demo.dspace.org/) if you don't have the User Interface installed or running locally.
2. In your Browser, open the "Developer Tools"
    a. In Chrome, go to "More Tools Developer Tools"
    b. In Firefox, go to "Web Developer Web Developer Tools".
    c. In Microsoft Edge, go to "More Tools Developer Tools".
3. Once in "Developer Tools", open the "Network" tab. This tab will provide information about *every single call that the User Interface makes to the REST API.*
4. Now, perform an action or use a feature in the User Interface in your browser window.
5. Analyze what calls were just sent to the REST API in your "Network" tab. Those are the exact REST API endpoints that were used to perform that action.
    a. NOTE: Some actions may use *multiple endpoints.*
6. Finally, lookup the documentation for those endpoint(s) in the REST Contract / Documentation (see link above)

REST Configuration

The following REST API configurations are provided in `[dspace]/config/rest.cfg` and may be overridden in your local.cfg

Property:	`rest.cors.allowed-origins`
Example Value:	`rest.cors.allowed-origins = ${dspace.ui.url}`

Informational Note:	Allowed Cross-Origin-Resource-Sharing (CORS) origins (in "Access-Control-Allow-Origin" header). Only these origins (client URLs) can successfully authenticate with your REST API *via a web browser*. Defaults to `${dspace.ui.url}` if unspecified (as the UI must have access to the REST API). If you customize that setting, MAKE SURE TO include `${dspace.ui.url}` in that setting if you wish to continue trusting the UI. Multiple allowed origin URLs may be comma separated (or this configuration can be defined multiple times). Wildcard value (*) is NOT SUPPORTED. Keep in mind any URLs added to this setting must be *an exact match with the origin*: mode (http vs https), domain, port, and subpath(s) all must match.. So, for example, these URLs are all considered different origins: "http://mydspace.edu", "http://mydspace.edu:4000" (different port), "https://mydspace.edu" (http vs https), "https://myapp.mydspace.edu" (different domain), and "https://mydspace.edu/myapp" (different subpath). *NOTE #1:* Development or command-line tools may not use CORS and may therefore bypass this configuration. CORS does not provide protection to the REST API / server webapp. Instead, its role is to protect browser-based clients from cookie stealing or other Javascript-based attacks. All modern web browsers use CORS to protect their users from such attacks. Therefore DSpace's CORS support is used to protect users who access the REST API via a web browser application, such as the DSpace UI or custom built Javascript tools/scripts. *NOTE #2:* If you modify this value to allow additional UIs (or Javascript tools) to access your REST API, then you may also need to modify `proxies.trusted.ipranges` to trust the IP address of each UI. Modifying trusted proxies is only necessary if the `X-FORWARDED-FOR` header must be trusted from each additional UIs. (The DSpace UI currently requires the `X-FORWARDED-FOR` header to be trusted). By default, `proxies.trusted.ipranges` will only trust the IP address of the `${dspace.ui.url}` configuration. *NOTE #3*: Although subpath must match, the Origin header itself sent from Angular will never contain a subpath. So if the dspace.ui.url config property ever changes to include a subpath like /myapp, then the expected origin will need to be added to `rest.cors.allowed-origins` , ie. the URL **without** the subpath. (Requires reboot of servlet container, e.g. Tomcat, to reload)
Property:	`rest.cors.allow-credentials`
Example Value:	`rest.cors.allow-credentials = true`
Informational Note:	Whether or not to allow credentials (e.g. cookies) sent by the client/browser in CORS requests (in "Access-Control-Allow-Credentials" header). For DSpace, this MUST be set to "true" to support CSRF checks (which use Cookies) and external authentication via Shibboleth (and similar). Defaults to "true" if unspecified. (Requires reboot of servlet container, e.g. Tomcat, to reload)
Property:	`rest.projections.full.max`
Example Value:	`rest.projections.full.max = 2`
Informational Note:	This property determines the max embeddepth for a FullProjection. This is also used by the SpecificLevelProjection as a fallback in case the property is defined on the bean. Usually, this should be kept as-is for best performance.
Property:	`rest.projection.specificLevel.maxEmbed`
Example Value:	`rest.projection.specificLevel.maxEmbed = 5`

Informational Note:	This property determines the max embed depth for a SpecificLevelProjection. Usually, this should be kept as-is for best performance.
Property:	`rest.properties.exposed`
Example Value:	`rest.properties.exposed = plugin.named.org.dspace.curate.CurationTask` `rest.properties.exposed = google.analytics.key`
Informational Note:	Define which configuration properties are exposed through the `http://<dspace.server.url>/api/config/properties/` REST API endpoint. If a rest request is made for a property which exists, but isn't listed here, the server will respond that the property wasn't found. This property can be defined multiple times to allow access to multiple configuration properties. Generally, speaking, it is ONLY recommended to expose configuration settings where they are necessary for the UI or client, as exposing too many configurations could be a security issue. This is why we only expose the two above settings by default.

REST Spring Boot Configuration

Because the REST API is a Spring Boot web application, you can also configure or override *any* Spring Boot settings in your local.cfg. DSpace's default Spring Boot configuration can be found in `[src]/dspace-server-webapp/src/main/resources/application.properties`. A few common settings from Spring Boot which you may wish to override in your local.cfg include:

Property:	`spring.servlet.multipart.max-file-size`
Example Value:	`spring.servlet.multipart.max-file-size = 512MB`
Informational Note:	Per Spring Boot docs, this setting specifies the maximum size of file that can be uploaded via Spring Boot (and therefore via the DSpace REST API). A value of "-1" removes any limit. DSpace sets this to 512MB by default.
Property:	`spring.servlet.multipart.max-request-size`
Example Value:	`spring.servlet.multipart.max-request-size = 512MB`
Informational Note:	Per Spring Boot docs, this setting specifies the maximum size of a single request via Spring Boot (and therefore via the DSpace REST API). That means if multiple files are uploaded at once, this is the maximum total size of all files. A value of "-1" removes any limit. DSpace sets this to 512MB by default.

Technical Design

The REST API & Server Webapp are built on Spring Boot and Spring HATEOAS, using Spring Security. It also aligns with Spring Data REST (though at this time it doesn't use it directly because of incompatibility with the DSpace data model).

The REST API is stateless, aligns with HATEOAS (Hypertext as the Engine of Application State) principles, returning HAL formatted JSON. This allows the REST API to be easily browsable/interactable via third-party tools that understand HAL & HATEOAS, such as the HAL Browser. JSON Web Tokens (JWT) are used to store state/session information between requests.

For better security, the REST API requires usage of CSRF tokens for all modifying requests.

More information can be found in the REST Contract at https://github.com/DSpace/RestContract/blob/main/README.md#rest-design-principles

DSpace Demo REST-API HAL Browser

- https://demo.dspace.org/server/

DSpace Python REST Client Library

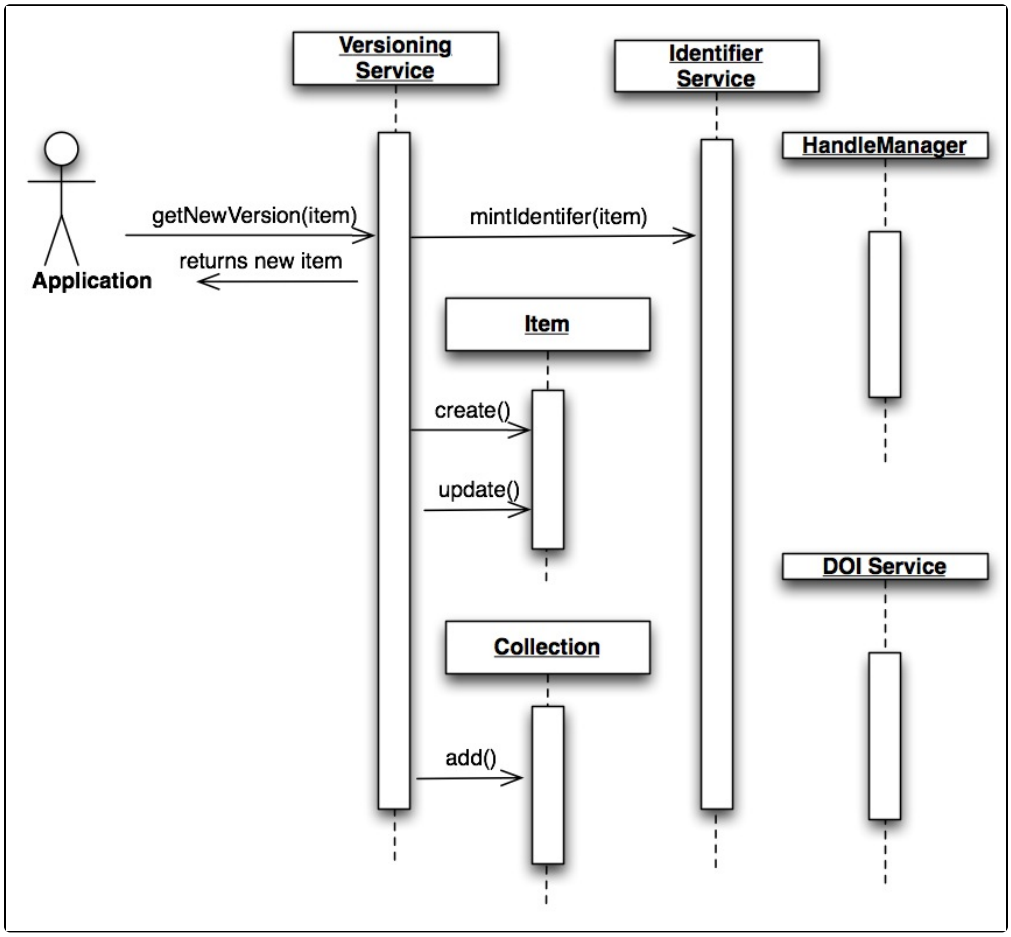
The Library Code is a DSpace Platinum certified Service Provider. They created a python library to use the REST-API of DSpace 7 and upwards out of python: https://github.com/the-library-code/dspace-rest-python.

Services to support Alternative Identifiers

Together with the Item Level Versioning an Identifier Service was introduced that make it possible to integrate new Identifiers. Currently the Identifier Service is used for Items only, but this may be changed in future versions of DSpace. Identifiers used for different versions are an very important point as part of an versioning strategy. The following documentation describes the Identifier Service in the context of Item Level Versioning, nevertheless the Identifier Service is also used for Items when the Item Level Versioning is switched off.

Versioning and Identifier Service

DSpace Item Versioning is encapsulated as an Extensible Service that may be reimplemented by the local repository maintainers to produce alternate versioning behaviors and Identifier Schemes. Versioning Services layer on top of IdentifierServices dedicated to Encoding, Resolution, Minting and Registration of Identifiers for specific DSpace Items. It is through this highly extensible layering of functionality where local developers can alter the versioning behavior and introduce their own local enhancements. The DSpace Service Manager, based on the Spring Framework, provides the key leverage for this flexibility.



Versioning Service

The *Versioning* Service will be responsible for the replication of one or more Items when a new version is requested. The new version will not yet be preserved in the Repository, it will be preserved when the databases transactional window is completed, thus when errors arise in the *versioning* process, the database will be properly kept in its original state and the application will alert that an exception has occurred that is in need of correction.

The *Versioning* Service will rely on a generic *IdentifierService* that is described below for minting and registering any identifiers that are required to track the revision history of the Items.

```
public interface VersioningService {

    Version createNewVersion(Context c, int itemId);

    Version createNewVersion(Context c, int itemId, String summary);

    void removeVersion(Context c, int versionID);

    void removeVersion(Context c, Item item);

    Version getVersion(Context c, int versionID);

    Version restoreVersion(Context c, int versionID);

    Version restoreVersion(Context c, int versionID, String summary);

    VersionHistory findVersionHistory(Context c, int itemId);

    Version updateVersion(Context c, int itemId, String summary);

    Version getVersion(Context c, Item item);
}
```

Identifier Service

The Identifier Service maintains an extensible set of *IdentifierProvider* services that are responsible for two important activities in Identifier management:

1. Resolution: *IdentifierService* act in a manner similar to the existing HandleManager in DSpace, allowing for resolution of DSpace Items from provided identifiers.
2. Minting: Minting is the act of reserving and returning an identifier that may be used with a specific DSpaceObject.
3. Registering: Registering is the act of recording the existence of a minted identifier with an external persistent resolver service. These services may reside on the local machine (HandleManager) or exist as external services (PURL or EZID DOI registration services)

```
public interface IdentifierService {

    /**
     *
     * @param context
     * @param dso
     * @param identifier
     * @return
     */
    String lookup(Context context, DSpaceObject dso, Class<? extends Identifier> identifier);

    /**
     *
     * This will resolve a DSpaceObject based on a provided Identifier.  The Service will interrogate
the providers in
     * no particular order and return the first successful result discovered.  If no resolution is
successful,
     * the method will return null if no object is found.
     *
     * TODO: Verify null is returned.
     *
     * @param context
     * @param identifier
     * @return
     * @throws IdentifierNotFoundException
     * @throws IdentifierNotResolvableException
     */
    DSpaceObject resolve(Context context, String identifier) throws IdentifierNotFoundException,
IdentifierNotResolvableException;

    /**
     *
     * Reserves any identifiers necessary based on the capabilities of all providers in the service.
     *
     * @param context
     * @param dso
```

```
     * @throws org.dspace.authorize.AuthorizeException
     * @throws java.sql.SQLException
     * @throws IdentifierException
     */
    void reserve(Context context, DSpaceObject dso) throws AuthorizeException, SQLException,
IdentifierException;

    /**
     *
     * Used to Reserve a Specific Identifier (for example a Handle,  hdl:1234.5/6) The provider is
responsible for
     * Detecting and Processing the appropriate identifier, all Providers are interrogated, multiple
providers
     * can process the same identifier.
     *
     * @param context
     * @param dso
     * @param identifier
     * @throws org.dspace.authorize.AuthorizeException
     * @throws java.sql.SQLException
     * @throws IdentifierException
     */
    void reserve(Context context, DSpaceObject dso, String identifier) throws AuthorizeException,
SQLException, IdentifierException;

    /**
     *
     * @param context
     * @param dso
     * @return
     * @throws org.dspace.authorize.AuthorizeException
     * @throws java.sql.SQLException
     * @throws IdentifierException
     */
    void register(Context context, DSpaceObject dso) throws AuthorizeException, SQLException,
IdentifierException;

    /**
     *
     * Used to Register a Specific Identifier (for example a Handle,  hdl:1234.5/6) The provider is
responsible for
     * Detecting and Processing the appropriate identifier, all Providers are interrogated, multiple
providers
     * can process the same identifier.
     *
     * @param context
     * @param dso
     * @param identifier
     * @return
     * @throws org.dspace.authorize.AuthorizeException
     * @throws java.sql.SQLException
     * @throws IdentifierException
     */
    void register(Context context, DSpaceObject dso, String identifier) throws AuthorizeException,
SQLException, IdentifierException;

    /**
     * Delete (Unbind) all identifiers registered for a specific DSpace item. Identifiers are "unbound"
across
     * all providers in no particular order.
     *
     * @param context
     * @param dso
     * @throws org.dspace.authorize.AuthorizeException
     * @throws java.sql.SQLException
     * @throws IdentifierException
     */
    void delete(Context context, DSpaceObject dso) throws AuthorizeException, SQLException,
IdentifierException;

    /**
```

```
     * Used to Delete a Specific Identifier (for example a Handle,  hdl:1234.5/6) The provider is
responsible for
     * Detecting and Processing the appropriate identifier, all Providers are interrogated, multiple
providers
     * can process the same identifier.
     *
     * @param context
     * @param dso
     * @param identifier
     * @throws org.dspace.authorize.AuthorizeException
     * @throws java.sql.SQLException
     * @throws IdentifierException
     */
    void delete(Context context, DSpaceObject dso, String identifier) throws AuthorizeException,
SQLException, IdentifierException;

}
```

User Interface Design Principles & Accessibility

These guidelines help ensure that all DSpace components have a consistent layout and follow the essential Web Content Accessibility Guidelines (WCAG). These guidelines MUST be followed by anyone who wants to contribute to the project. See also our Code Contribution Guidelines

- Overview
    - Terminology used in this page
- Guiding Principles
- User Interface Design Guidelines
- User Interface Accessibility Guidelines

Overview

These guidelines apply primarily to the "Base Theme" for the DSpace User Interface.

- **Base Theme** (`/src/app/` directories): The primary look & feel of DSpace (e.g. HTML layout, header/footer, etc) is defined by the HTML5 templates under this directory. Each HTML5 template is stored in a subdirectory named for the Angular component where that template is used. The base theme includes very limited styling (CSS, etc), based heavily on *default Bootstrap (4.x) styling*, and only allowing for minor tweaks to improve accessibility (e.g. default Bootstrap's color scheme does not have sufficient color contrast)
- Two additional themes are provided with DSpace out-of-the-box
    - **Custom Theme** (`/src/themes/custom` directories): This directory acts as the scaffolding or template for creating a new custom theme. It provides (empty) Angular components/templates which allow you to change the theme of individual components. Since all files are empty by default, if you enable this theme (without modifying it), it will look *identical* to the Base Theme.
    - **DSpace Theme** (`/src/themes/dspace` directories): This is the default theme for DSpace 7. It is a very simple *example theme* providing a custom color scheme & homepage on top of the Base Theme.
- More information on themes (in general) can be found in the User Interface Customization documentation

Terminology used in this page

The following terms are used frequently in this page, and this is a quick reference to what we mean by these terms:

- template = Angular template
- component = Angular component
- style = CSS/SCSS styles
- theme = a set of templates that work together (along with the styling/CSS) to comprise the look & feel of the site.

Guiding Principles

- All templates in the Base Theme (*/src/app* directories) should only use default Bootstrap styling. Documentation at: *https://getbootstrap.com/docs/4.6/getting-started/introduction/*
    - Exceptions may be made for accessibility purposes. For example, Bootstrap notes their default color scheme does not always have sufficient color contrast
- When Bootstrap Components (accordion, dropdown, etc …) are required you MUST use the included ng-bootstrap library. Documentation at: *https://ng-bootstrap.github.io/#/components/accordion/examples*

The use of the Bootstrap framework can help in achieving some WCAG goals such as ‘Visual Presentation’ (AAA), 'Parsing' (A), ‘Orientation’ (AA), ‘Reflow’ (AA) and ‘Text Spacing’ (AA). See the Bootstrap chapter ‘*Accessibility*’ for an explanation of WCAG and where to find additional information.

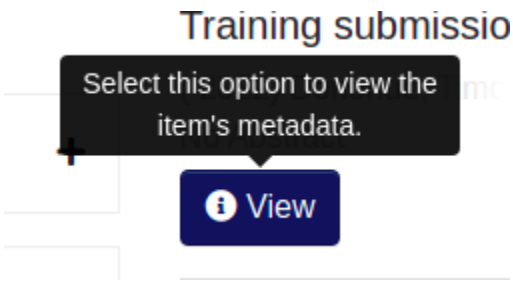
User Interface Design Guidelines

This section provides basic guidelines on User Interface layout/design and the elements (or components) used by the DSpace User Interface. DSpace developers strive to meet these guidelines in order to ensure consistent behavior/layout on all pages. If you find a guideline is not met on a particular page, please report a bug ticket for that user interface page.

- All the buttons should have a text description and an icon:



  If that it’s not possible (e.g. a small button with an icon) always use the ‘name’ and ‘title’ properties.
- Use the tooltip component when you need a better explanation of a button functionality. For example:



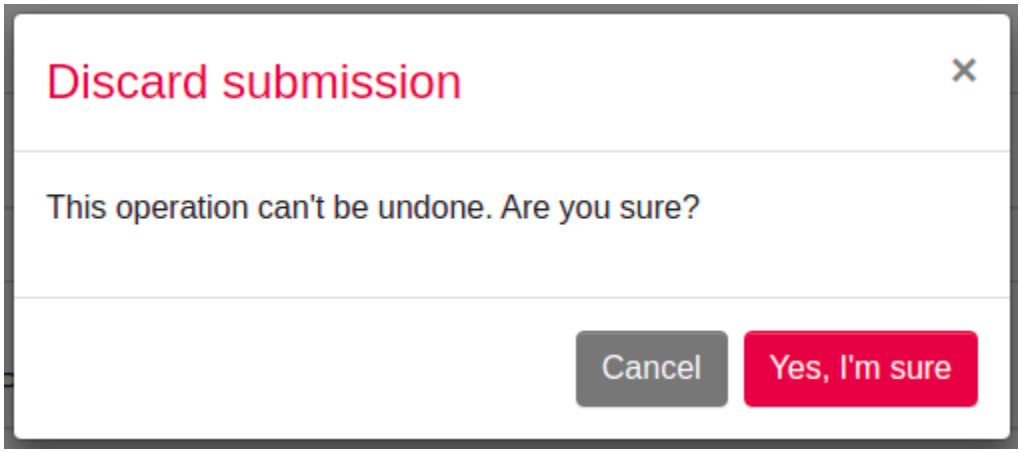
- For UI elements on public pages that are only visible to users with elevated privileges use an inverted color scheme `btn-dark`.
- For the 'anchor' (the '<a>' element) that uses the 'btn' Bootstrap CSS class always use `btn-outline-primary`.
- For the main action button use the Bootstrap CSS class `btn-primary`.
- For buttons like 'Cancel' or 'Back' use che Bootstrap CSS class `btn-outline-secondary`.
- For buttons that open a dropdown list use the Bootstrap CSS class `btn-secondary`.
- In a button series, or group, only one '<button>' has the Bootstrap CSS class `btn-primary`.
- The buttons order, inside a group or a series, should follow the kind of action it performs. At the left side the most 'light' action, like 'Back' or 'Cancel', at the right side the most 'changing' action like 'Delete' or 'Remove'. So, the order is:

```
Back -> Cancel -> Submit/Edit/Save/Save for later/Deposit -> Discard/Delete/Remove
```

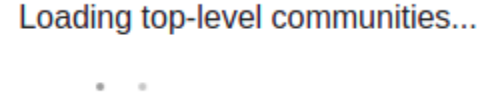
Here an example:



- Where possible, align the buttons to the right.
- Every macro-function inside a page shall be a primary button.
- If, after an action, there is a waiting time, e.g. for the server response, the control (e.g. the button) that launched the action must become disabled and show an animated waiting icon. All other associated controls must be disabled (property '*disabled="true"*').
- Action confirmation should occur on cancellations or non-reversible operations; a modal should appear with a message giving the possibility to confirm or cancel the requested action:

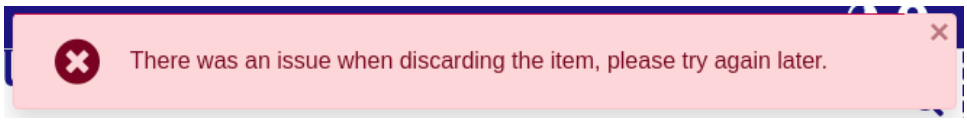


- To inform the user that a certain section of the page is about to be modified as a result of an action (e.g. a page change to move forward in a list) it's necessary to make an animated waiting icon appear:



- All searches that do not return a result must report their absence via a message within a block (e.g. '* *') with the Bootstrap CSS class 'alert-info'.
- The items in the horizontal top navigation menu are links to pages always available to the user (logged or not).
- The items in the left vertical menu are links concerning management, administration and creation or modification of DSpace items. The list can be different according to the permissions of the logged user.

- If a page topic has more logical subdivisions, it's opportune to separate them in more tabs (e.g. the 'Edit collection' page where you can edit metadata, roles and policies).
- Inside the Extended Footer you can insert information about the institution, partnerships, social links, external links or legal information.
- Notifications
    - All the notifications with the whole page scope should be placed on the top right side of the page, hovering on the elements beneath it:



    - Notifications types and color schemes: the corresponding Bootstrap CSS class must be applied:

```
Notification type   Boostrap Class   Description
-------------------------------------------------------------------------------
success             alert-success    to indicate the successful execution of an action
warning             alert-warning    to warn of a possible change to the standard behavior
                                     of some controls (button or action)
info                alert-info       to provide any additional information to the user
                                     (e.g.: no search results)
danger              alert-danger     to inform that an error has occurred
```

    - Besides warnings about changes in behaviors, all the notifications must have a closing button to remove them.
    - The timed notifications are allowed when the notification is purely informational and there is no possibility of interaction (e.g. presence of buttons or forms within the notification).
- For the most common actions and alerts, the following free FontAwesome icons are recommended:

```
Information: fas fa-info-circle
Search     : fas fa-search
Import     : fas fa-file-import
Export     : fas fa-file-export
Add/New    : fas fa-plus
Edit       : fas fa-edit
Save       : fas fa-save
Delete     : fas fa-trash
Cancel     : fas fa-times
Undo       : fas fa-undo
Back       : fas fa-arrow-left

Login      : fas fa-sign-in-alt
Logout     : fas fa-sign-out-alt
```

User Interface Accessibility Guidelines

The DSpace User Interface strives to align with all WCAG AA criteria. Some AAA criteria may also be supported. For more information see Accessibility documentation.

Here are specific guidelines we strive to follow in the "Base Theme":

- Web pages have titles that describe their topic or purpose.
- Use the Breadcrumb component on all pages.
- Use headers and labels to describe topics or purposes in order to provide support for the user to navigate, find content, and determine his location. Always provide labels or instructions when content requires user input actions, e.g. forms.
- Elements have full opening and closing tags, are nested according to their own specifications, contain no duplicate attributes, and all IDs are unique.
- Always use ALT properties within the IMAGE tag to describe the content of the images.
- The default language setting of each Web page can be determined programmatically with:

```
<html lang="en">
...
</html>
```

- The text is never justified.
- Text, with the exception of subtitles and images containing text, can be resized up to 200 percent without the aid of assistive technologies and without loss of content or functionality.
- Use '' elements to emphasize something rather than '<b>'. For example, required fields in a form.
- The website operation does not depend on the screen orientation.
- To create Web pages that look and feel predictable, components that have the same functionality within a set of Web pages should be uniquely identified. This means, for example, avoiding buttons with similar functions but different descriptions or using similar icons for different functions. Wherever possible, on different pages or different tabs / page sections, place controls in the same position.
- ARIA Role, ARIA label and ARIA labelledby: use the 'aria-label' and 'role' properties to identify the regions of a page. Ex. to identify two kinds of menu:

```
<div id="leftnav" role="navigaton" aria-label="Primary">
   ...
</div>
<div id="rightnav" role="navigation" aria-label="Secondary">
   ...
</div>
```

Other regions of a page can be the header, footer, the page content, etc. Remember that ‘aria-label’ should be used only when there is no other element in the HTML page that can describe better the element itself. In that case use the ‘aria-labelledby’:

```
<div id="leftnav" role="navigaton" aria-labelledby="menuTitle">
   <h4 id="menuTitle">This is the primary menu</h4>
   ...
</div>
```

It can be also used on a simple text field to provide a label in a situation where there is no text available for a dedicated label but there is other text on the page that can be used to accurately label the control. Ex.:

```
<input name="searchText" type="text" aria-labelledby="searchButton">
<input name="searchButton" id="searchButton" type="submit" value="Search">
```

It’s possible to use them to provide labels to user interface controls (ex.: buttons or inputs in a form).

- For all User Interface components, ‘name’ and ‘role’ must be determined programmatically. To do this use:
    - label elements to associate text labels with form controls
    - ‘aria-labelledby’ and ‘aria-label
- Always prefer the following hierarchy of choices when trying to describe topics or purposes:
    - Plain text with a full description where possible. This will help people with cognitive disabilities who may not immediately know the purpose of the field because the label used by the author is not familiar to them;
    - Label element;
    - ARIA label and ARIA labelledby.
- Identify programmatically the purpose of the inputs using the guidelines described above and the attribute ‘autocomplete’:

```
<input id="fname" type="text" autocomplete="given-name" ... >
```

This property is useful to browsers / user agents to identify the content and provide auto-fill capabilities. The values you can use with 'autocomplete' are described here:

    - https://www.w3.org/TR/WCAG21/#input-purposes
- Including the text of the visible label as part of the accessible name. When speech recognition software processes speech input and looks for matches, it uses the ‘accessible name' of controls, so it’s important that what the user reads in label or description is, at least partially, what is defined in the ‘accessible name' like ‘aria-label’ or ‘aria-labelledby’. E.g. if a button has a visible value of ‘search’ and its ‘aria-label’ has ‘go’ a problem can occur when the user says 'click Search’ :

```
<button aria-label="Go">Search</button>
```

So, if you have an ‘accessible name' available, you can expand it using the label text inside it. All of the following examples are valid:

```
<button>Search</button>
<button aria-label="Search for matches"><i class="fa fa-search"></i></button>

<h4 id="buttonTitle">Search for matches</h4>
<button aria-labelledby="buttonTitle">Search</button>
```

- Order of focus: For example, in a form, use ‘tabindex’ logically (e.g. street number after street name).
- Change the color of an element when it receives FOCUS: e.g. CSS can be used to apply a different color when link elements receive focus.
- Ensure that the information conveyed by color differences is also available in the text; e.g. links also underlined or mandatory form fields highlighted with an asterisk (*);
- Error identification: The element in error is identified and described by text even with client-side controls. Use the property ‘*aria-invalid=”true”*’ inside that element. For example, within a form, apply client-side validations to the input fields and make sure that any error message is comprehensive; where possible, suggestions on how to correct the error, should be provided to the user.



- Provide users with sufficient time to read and use the content. For example, inside the timed notifications (error or success messages) provide a button to stop the timer.

Workflow

Configuration

Main workflow configuration

As of DSpace 7, the `workflow.xml` configuration file has been migrated to use Spring Bean syntax (instead of a custom XML format). The structure of this XML has changed. If you need help migrating your old `workflow.xml` file (which started with a `<wf-config>` tag) to the new format (using `<bean>` tags), an XSLT script is available: workflow-migration.xsl

The workflow main configuration can be found in the workflow.xml file, located in **`[dspace]/config/spring/api/workflow.xml`** . An example of this workflow configuration file can be found below.

```
<beans>
    <bean class="org.dspace.xmlworkflow.XmlWorkflowFactoryImpl">
        <property name="workflowMapping">
            <util:map>
                <entry key="defaultWorkflow" value-ref="defaultWorkflow"/>
<!--            <entry key="123456789/4" value-ref="selectSingleReviewer"/>-->

<!--            <entry key="123456789/5" value-ref="scoreReview"/>-->
            </util:map>
        </property>
    </bean>

    <!--Standard DSpace workflow-->
    <bean name="defaultWorkflow" class="org.dspace.xmlworkflow.state.Workflow">

        <property name="firstStep" ref="reviewstep"/>
        <property name="steps">
            <util:list>
                <ref bean="reviewstep"/>
                <ref bean="editstep"/>
                <ref bean="finaleditstep"/>
            </util:list>
        </property>
    </bean>

    <bean id="{workflow.id}"
          class="org.dspace.xmlworkflow.state.Workflow">
        <!-- Another workflow configuration-->
    </bean>

    <!-- Role beans.  See below. -->

    <!-- Step beans.  See below. -->

</beans>
```

workflowFactory bean (org.dspace.xmlworkflow.XmlWorkflowFactoryImpl)

The workflow map contains a mapping between collections in DSpace and a workflow configuration, and is defined by the `workflowMapping` property of the workflow factory. Similar to the configuration of the submission process, the mapping can be done based on the handle of the collection. The mapping with "defaultWorkflow" 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 "`entry`" element with two attributes:

- key: can either be a collection handle or "defaultWorkflow"
- value-ref: the value of this attribute points to one of the workflow configurations defined by the "Workflow" beans

workflow beans (org.dspace.xmlworkflow.state.Workflow)

The workflow bean is a repeatable XML element and represents one workflow process. It requires the following:

- "`name`" attribute: a unique name used for the identification of the workflow and used in the workflow to collection mapping
- "`firstStep`" property: the identifier of the first step of the workflow. This step 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 "`firstStep`" property will he the first step the item will go through.
- "`steps`" property: a list of all steps within this workflow (in the order they will be processed).

role beans (org.dspace.xmlworkflow.Role)

Each workflow step has defined "role" property. A role represents one or more existing 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" bean and has the following:

- "id" attribute: a unique identifier (in one workflow process) for the role
- "description" property: optional attribute to describe the role
- "scope" property: optional attribute that is used to find our group and must have one of the following values, which are defined as constant fields of org.dspace.xmlworkflow.Role.Scope:
    - 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" property: The name specified in the name attribute of a role will be used to lookup an eperson group 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.

```
<bean id="reviewer" class="org.dspace.xmlworkflow.Role">
    <property name="scope" value="#{ T(org.dspace.xmlworkflow.Role.Scope).COLLECTION}"/>
    <property name="name" value="Reviewer"/>
    <property name="description" value="The people responsible for this step are able to edit the metadata of
incoming submissions, and then accept or reject them."/>
</bean>
```

step beans (org.dspace.xmlworkflow.state.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 in order to further configure it:

- "name" attribute: The name attribute specifies a unique identifier for the step. This identifier 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.
- "userSelectionMethod" property: This attribute defines the UserSelectionAction that will be used to determine how to attach users to this step for a workflow-item. The value of this attribute must refer to the identifier of an action bean in the workflow-actions.xml. Examples of the user attachment to a step are the currently used system of a task pool or as an alternative directly assigning a user to a task.
- "role" property: 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

```
<bean name="reviewstep" class="org.dspace.xmlworkflow.state.Step">
    <property name="userSelectionMethod" ref="claimaction"/>
    <property name="role" ref="reviewer"/>
    <property name="outcomes">
        <util:map>
            <entry key="#{ T(org.dspace.xmlworkflow.state.actions.ActionResult).OUTCOME_COMPLETE}"

                   value-ref="editstep"/>
        </util:map>
    </property>
    <property name="actions">
        <util:list>
            <ref bean="reviewaction"/>
        </util:list>
    </property>
</bean>
```

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 "outcomes". 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 "outcomes" 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.

```
<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="pr
ototype">
         <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 configuration refers to a bean definition in the `workflow-actions.xml` file. In order to define a new user selection action, the following XML code is used:

```
<bean id="{action.api.id.2}" class="org.dspace.xmlworkflow.state.actions.UserSelectionActionConfig" scope="prototy
pe">
    <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 of the task. This is the same as the `id` attribute of the bean and is used by the workflow configuration 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 similarly 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.

Authorizations

Currently, the authorizations are always granted and revoked based on the tasks that are available for certain users and groups. The types of authorization policies that is granted for each of these is always the same:

- READ
- WRITE
- ADD
- DELETE

Database

The workflow uses a separate metadata schema named `workflow`. The fields this schema contains can be found in the `[dspace]/config/registries` directory and in the file `workflow-types.xml`. This schema is only used when using the score reviewing system at the moment, but one could always use this schema if metadata is required for custom workflow steps.

The following tables have been added to the DSpace database. All tables are prefixed with 'cwf_' to avoid any confusion with the existing workflow related database tables:

cwf_workflowitem

The cwf_workflowitem table contains the different workflowitems in the workflow. This table has the following columns:

- workflowitem_id: The identifier of the workflowitem and primary key of this table
- item_id: The identifier of the DSpace item to which this workflowitem refers.
- collection_id: The collection to which this workflowitem is submitted.
- multiple_titles: Specifies whether the submission has multiple titles (important for submission steps)
- published_before: Specifies whether the submission has been published before (important for submission steps)
- multiple_files: Specifies whether the submission has multiple files attached (important for submission steps)

cwf_collectionrole

The cwf_collectionrole table represents a workflow role for one collection. This type of role is the same as the roles that existed in the original workflow meaning that for each collection a separate group is defined to 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

DSpace Reference

- Architecture
  - Application Layer
  - Business Logic Layer
  - DSpace Services Framework
  - Storage Layer
- Configuration Reference
- Directories and Files
- DSpace Item State Definitions
- Metadata and Bitstream Format Registries
- History
  - Changes in 10.x
  - Changes in Older Releases

Architecture

1 Overview

Overview

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

- **Application Layer** - All external/public facing interfaces/tools. These include the Web User Interface, REST API, OAI-PMH, RDF, and SWORD (v and v2) interfaces. Also includes the Command Line interface, and various tools that can be used to import/export data to/from DSpace.
- **Business Logic Layer** - Primarily the Java API layer ([dspace-source]/dspace-api and dspace-services), which provides the core business logic for all the various application interfaces.
- **Storage Layer** - A subset of the dspace-api (*org.dspace.storage.* classes)* whose role is to manage all content storage (metadata, relationships, bitstreams) for all business layer objects. This layer is provides access to a relational database (Postgres or Oracle usually) via Hibernate ORM & using FlywayDB for migrations/updates. It also defines a custom BitStoreService for storing files (bitstreams) via storage plugins (currently supporting filesystem storage or Amazon S3 storage).

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 Java API (in the case of the business logic layer), and the DSpace REST API (in the case of the application layer). In the Application Layer, it's worth noting that the Web User Interface only accesses DSpace via the REST API.

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 Java 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.

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.

The REST API provides not only JavaDocs, but also a public contract. See REST API.

Each layer is described in a separate section:

- Storage Layer
  - RDBMS
  - Bitstream Store
- Business Logic Layer
  - Core Classes
  - Content Management API
  - Workflow System
  - Administration Toolkit
  - E-person/Group Manager
  - Authorisation
  - Handle Manager/Handle Plugin
  - Search
  - Browse API
  - History Recorder
  - Checksum Checker
- Application Layer
  - Web User Interface
  - OAI-PMH Data Provider
  - Item Importer and Exporter
  - Transferring Items Between DSpace Instances
  - Registration
  - METS Tools
  - Media Filters

- Sub-Community Management

Application Layer

The following explains the components of the Application Layer.

1 Web User Interface
    1.1 Web UI Files
2 REST API
3 OAI-PMH Data Provider
4 RDF / Linked Data Provider
5 SWORD v1 Service / Server
6 SWORD v2 Service / Server
7 DSpace Command Line Launcher
    7.1 Command Launcher Structure

Web User Interface

The DSpace Web UI is the largest and most-used component in the application layer. As of DSpace 7, it has been rebuilt on Angular.io communicating via the REST API to the rest of DSpace.

Web UI Files

The web User Interface code is managed in a separate GitHub Project:

https://github.com/DSpace/dspace-angular/

Quick setup and configuration instructions can be found in the README of that project.

REST API

This component defines the main public API of the Application Layer. See REST API section of the documentation.

OAI-PMH Data Provider

See OAI section of the documentation

RDF / Linked Data Provider

See Linked (Open) Data section of the documentation

SWORD v1 Service / Server

See SWORDv1 Server section of the documentation

SWORD v2 Service / Server

See SWORDv2 Server section of the documentation

DSpace Command Line Launcher

The DSpace Command Launcher brings together the various command and scripts into a standard-practice for running CLI runtime programs. See Command Line Operations

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>*

See Command Line Operations for additional details.

Business Logic Layer

1 Core Classes
- 1.1 The Configuration Service
- 1.2 Constants
- 1.3 Context
- 1.4 Email
- 1.5 LogManager
- 1.6 Utils

2 Content Management API
- 2.1 Other Classes
- 2.2 Modifications
- 2.3 What's In Memory?
- 2.4 Dublin Core Metadata
- 2.5 Support for Other Metadata Schemas
- 2.6 Packager Plugins

3 Plugin Service
- 3.1 Concepts
- 3.2 Using the Plugin Service
  - 3.2.1 Types of Plugin
  - 3.2.2 Self-Named Plugins
  - 3.2.3 Obtaining a Plugin Instance
  - 3.2.4 Lifecycle Management
  - 3.2.5 Getting Meta-Information
- 3.3 Implementation
  - 3.3.1 LegacyPluginServiceImpl Class
  - 3.3.2 SelfNamedPlugin Class
  - 3.3.3 Errors and Exceptions
- 3.4 Configuring Plugins
  - 3.4.1 Configuring Singleton (Single) Plugins
  - 3.4.2 Configuring Sequence of Plugins
  - 3.4.3 Configuring Named Plugins
- 3.5 Use Cases
  - 3.5.1 Managing the MediaFilter plugins transparently
  - 3.5.2 A Singleton Plugin
  - 3.5.3 Plugin that Names Itself
  - 3.5.4 Stackable Authentication

4 Workflow System
5 Administration Toolkit
6 E-person/Group Manager
7 Authorization
- 7.1 Special Groups
- 7.2 Miscellaneous Authorization Notes

8 Handle Manager/Handle Plugin
9 Search
- 9.1 Harvesting API

10 Browse API
- 10.1 Using the API

11 Checksum checker
12 OpenSearch Support
13 Embargo Support
- 13.1 What is an Embargo?
- 13.2 Embargo Model and Life-Cycle

Core Classes

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

The Configuration Service

The configuration service 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 *ConfigurationService* 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.

For many more details on configuration in DSpace, see Configuration Reference

Constants

This class contains constants that are used to represent types of object and actions in the database. For example, authorization policies can relate to objects of different types, so the *resourcepolicy* table has columns *resource_id*, which is the internal ID of the object, and *resource_type_id*, which indicates whether the object is an item, collection, bitstream etc. The value of *resource_type_id* is taken from the *Constants* class, for example *Constants.ITEM*.

Here are a some of the most commonly used constants you might come across:

DSpace types

- Bitstream: 0
- Bundle: 1
- Item: 2
- Collection: 3
- Community: 4
- Site: 5
- Group: 6
- Eperson: 7

DSpace actions

- Read: 0
- Write: 1
- Delete: 2
- Add: 3
- Remove: 4

Refer to the org.dspace.core.Constants for all of the Constants.

Context

The *Context* class is central to the DSpace operation. Any code that wishes to use the any API in the business logic layer must first create itself a *Context* object. This is akin to opening a connection to a database (which is in fact one of the things that happens.)

A context object is involved in most method calls and object constructors, so that the method or object has access to information about the current operation. When the context object is constructed, the following information is automatically initialized:

- A connection to the database. This is a transaction-safe connection. i.e. the 'auto-commit' flag is set to false.
- A cache of content management API objects. Each time a content object is created (for example *Item* or *Bitstream*) it is stored in the *Context* object. If the object is then requested again, the cached copy is used. Apart from reducing database use, this addresses the problem of having two copies of the same object in memory in different states.
  The following information is also held in a context object, though it is the responsibility of the application creating the context object to fill it out correctly:

- The current authenticated user, if any
- Any 'special groups' the user is a member of. For example, a user might automatically be part of a particular group based on the IP address they are accessing DSpace from, even though they don't have an e-person record. Such a group is called a 'special group'.
- Any extra information from the application layer that should be added to log messages that are written within this context. For example, the Web UI adds a session ID, so that when the logs are analyzed the actions of a particular user in a particular session can be tracked.
- A flag indicating whether authorization should be circumvented. This should only be used in rare, specific circumstances. For example, when first installing the system, there are no authorized administrators who would be able to create an administrator account!As noted above, the public API is *trusted*, so it is up to applications in the application layer to use this flag responsibly.
  Typical use of the context object will involve constructing one, and setting the current user if one is authenticated. Several operations may be performed using the context object. If all goes well, *complete* is called to commit the changes and free up any resources used by the context. If anything has gone wrong, *abort* is called to roll back any changes and free up the resources.

You should always *abort* a context if *any* error happens during its lifespan; otherwise the data in the system may be left in an inconsistent state. You can also *commit* a context, which means that any changes are written to the database, and the context is kept active for further use.

Email

Sending e-mails is pretty easy. Just use the configuration manager's *getEmail* method, set the arguments and recipients, and send.

The e-mail texts are stored in `[dspace]/config/emails`. They are processed by the standard *java.text.MessageFormat*. At the top of each e-mail are listed the appropriate arguments that should be filled out by the sender. Example usage is shown in the *org.dspace.core.Email* Javadoc API documentation.

LogManager

The log manager consists of a method that creates a standard log header, and returns it as a string suitable for logging. Note that this class does not actually write anything to the logs; the log header returned should be logged directly by the sender using an appropriate Log4J call, so that information about where the logging is taking place is also stored.

The level of logging can be configured on a per-package or per-class basis by editing `[dspace]/config/log4j.properties`. You will need to stop and restart Tomcat for the changes to take effect.

A typical log entry looks like this:

2002-11-11 08:11:32,903 INFO org.dspace.app.webui.servlet.DSpaceServlet @ anonymous:session_id=BD84E7C194C2CF4BD0EC3A6CAD0142BB: view_item:handle=1721.1/1686

This is breaks down like this:

Date and time, milliseconds	*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.

Content Management API

The content management API package *org.dspace.content* contains Java classes for reading and manipulating content stored in the DSpace system. This is the API that components in the application layer will probably use most.

Classes corresponding to the main elements in the DSpace data model (*Community*, *Collection*, *Item*, *Bundle* and *Bitstream*) are sub-classes of the abstract class *DSpaceObject*. The *Item* object handles the Dublin Core metadata record.

Each class generally has one or more static *find* methods, which are used to instantiate content objects. Constructors do not have public access and are just used internally. The reasons for this are:

- "Constructing" an object may be misconstrued as the action of creating an object in the DSpace system, for example one might expect something like:

  ```
  Context dsContent = new Context();
  Item myItem = new Item(context, id)
  ```

  to construct a brand new item in the system, rather than simply instantiating an in-memory instance of an object in the system.
- *find* methods may often be called with invalid IDs, and return *null* in such a case. A constructor would have to throw an exception in this case. A *null* return value from a static method can in general be dealt with more simply in code.
- If an instantiation representing the same underlying archival entity already exists, the *find* method can simply return that same instantiation to avoid multiple copies and any inconsistencies which might result.

Collection, *Bundle* and *Bitstream* do not have *create* methods; rather, one has to create an object using the relevant method on the container. For example, to create a collection, one must invoke *createCollection* on the community that the collection is to appear in:

```
Context context = new Context();
Community existingCommunity = Community.find(context, 123);
Collection myNewCollection = existingCommunity.createCollection();
```

The primary reason for this is for determining authorization. In order to know whether an e-person may create an object, the system must know which container the object is to be added to. It makes no sense to create a collection outside of a community, and the authorization system does not have a policy for that.

Items are first created in the form of an implementation of InProgressSubmission. An *InProgressSubmission* represents an item under construction; once it is complete, it is installed into the main archive and added to the relevant collection by the *InstallItem* class. The *org.dspace.content* package provides an implementation of *InProgressSubmission* called *WorkspaceItem*; this is a simple implementation that contains some fields used by the Web submission UI. The *org.dspace.workflow* also contains an implementation called *WorkflowItem* which represents a submission undergoing a workflow process.

In the previous chapter there is an overview of the item ingest process which should clarify the previous paragraph. Also see the section on the workflow system.

Community and *BitstreamFormat* do have static *create* methods; one must be a site administrator to have authorization to invoke these.

Other Classes

Classes whose name begins *DC* are for manipulating Dublin Core metadata, as explained below.

The *FormatIdentifier* class attempts to guess the bitstream format of a particular bitstream. Presently, it does this simply by looking at any file extension in the bitstream name and matching it up with the file extensions associated with bitstream formats. Hopefully this can be greatly improved in the future!

The *ItemIterator* class allows items to be retrieved from storage one at a time, and is returned by methods that may return a large number of items, more than would be desirable to have in memory at once.

The *ItemComparator* class is an implementation of the standard *java.util.Comparator* that can be used to compare and order items based on a particular Dublin Core metadata field.

Modifications

When creating, modifying or for whatever reason removing data with the content management API, it is important to know when changes happen in-memory, and when they occur in the physical DSpace storage.

Primarily, one should note that no change made using a particular *org.dspace.core.Context* object will actually be made in the underlying storage unless *complete* or *commit* is invoked on that *Context*. If anything should go wrong during an operation, the context should always be aborted by invoking *abort*, to ensure that no inconsistent state is written to the storage.

Additionally, some changes made to objects only 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 Metadatum class is a simple container that represents a single Dublin Core-like element, optional qualifier, value and language. Note that since DSpace 1.4 the *MetadataValue* and associated classes are preferred (see Support for Other Metadata Schemas). The other classes starting with *DC* are utility classes for handling types of data in Dublin Core, such as people's names and dates. As supplied, the DSpace registry of elements and qualifiers corresponds to the Library Application Profile for Dublin Core. It should be noted that these utility classes assume that the values will be in a certain syntax, which will be true for all data generated within the DSpace system, but since Dublin Core does not always define strict syntax, this may not be true for Dublin Core originating outside DSpace.

Below is the specific syntax that DSpace expects various fields to adhere to:

Element	Qualifier	Syntax	Helper Class
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 Metadatum class. Finally the *MetadataSchema* class is used to describe supported schemas. The DC schema is supported by default. Refer to the javadoc for method details.

Packager Plugins

The Packager plugins let you *ingest* a package to create a new DSpace Object, and *disseminate* a content Object as a package. A package is simply a data stream; its contents are defined by the packager plugin's implementation.

To ingest an object, which is currently only implemented for Items, the sequence of operations is:

1. Get an instance of the chosen *PackageIngester* plugin.
2. Locate a Collection in which to create the new Item.
3. Call its *ingest* method, and get back a *WorkspaceItem*.
   The packager also takes a *PackageParameters* object, which is a property list of parameters specific to that packager which might be passed in from the user interface.

Here is an example package ingestion code fragment:

```
Collection collection =  find target collection
     InputStream source = ...;
     PackageParameters params = ...;
     String license = null;

    PackageIngester sip = (PackageIngester) PluginManager
            .getNamedPlugin(PackageIngester.class, packageType);

    WorkspaceItem wi = sip.ingest(context, collection, source, params, license);
```

Here is an example of a package dissemination:

```
OutputStream destination = ...;
     PackageParameters params = ...;
     DSpaceObject dso = ...;

     PackageIngester dip = (PackageDisseminator) PluginManager
             .getNamedPlugin(PackageDisseminator.class, packageType);

     dip.disseminate(context, dso, params, destination);
```

Plugin Service

In DSpace 6, the old "PluginManager" was replaced by `org.dspace.core.service.PluginService` which performs the same activities/actions.

The PluginService 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 Service

Types of Plugin

The Plugin Service 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 `getAllPluginNames()` 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 Service 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 Service 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 *PluginService* fulfills a request for a plugin, a new instance is always created.

Getting Meta-Information

The *PluginService* 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 getAllPluginNames() 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 *PluginService* 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.

LegacyPluginServiceImpl Class

The *LegacyPluginServiceImpl* class is the default PluginService implementation. While it is possible to implement your own version of PluginService, no other implementations are provided with DSpace

Here are the public methods, followed by explanations:

- `Object getSinglePlugin(Class interfaceClass)` - 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.
- `Object[] getPluginSequence(Class interfaceClass)` - Returns instances of all plugins that implement the interface *interfaceClass*, 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.
- `Object getNamedPlugin(Class interfaceClass, String name)` - Returns an instance of a plugin that implements the interface *interfaceClass* 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.

- `String[] getAllPluginNames(Class interfaceClass)` - Returns all of the names under which a named plugin implementing the interface *interfaceClass* 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).

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 Service's configuration comes from the DSpace Configuration Service (see Configuration Reference). 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.

Use Cases

Here are some usage examples to illustrate how the Plugin Service works.

Managing the MediaFilter plugins transparently

The MediaFilterService implementation relies heavily on the Plugin Service. The MediaFilter classes become plugins named in the configuration. Refer to the Configuration Reference 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 PluginService the names to which it answers instead of forcing someone to configure those names in two places (and try to keep them synchronized).

Here is the configuration file listing both the plugin's own configuration and the *PluginService* config line:

```
crosswalk.dissemination.stylesheet.DublinCore = xwalk/TESTDIM-2-DC_copy.xsl
crosswalk.dissemination.stylesheet.MODS = xwalk/mods.xsl

plugin.selfnamed.org.dspace.content.metadata.DisseminationCrosswalk = \
  org.dspace.content.metadata.XsltDisseminationCrosswalk
```

This look into the implementation shows how it finds configuration entries to populate the array of plugin names returned by the *getPluginNames()* method. Also note, in the *getStylesheet()* method, how it uses the plugin name that created the current instance (returned by *getPluginInstanceName()*) to find the correct stylesheet.

```
public class XsltDisseminationCrosswalk extends SelfNamedPlugin
{
    ....
    private final String prefix =
        "crosswalk.dissemination.stylesheet.";
    ....
    public static String[] getPluginNames()
    {
        List aliasList = new ArrayList();
        Enumeration pe = ConfigurationManager.propertyNames();

        while (pe.hasMoreElements())
        {
            String key = (String)pe.nextElement();
            if (key.startsWith(prefix))
                aliasList.add(key.substring(prefix.length()));
        }
        return (String[])aliasList.toArray(new
        String[aliasList.size()]);
    }

    // get the crosswalk stylesheet for an instance of the plugin:
    private String getStylesheet()
    {
        return ConfigurationManager.getProperty(prefix +
        getPluginInstanceName());
    }
}
```

Stackable Authentication

The Stackable Authentication mechanism needs to know all of the plugins configured for the interface, in the order of configuration, since order is significant. It gets a Sequence Plugin from the Plugin Manager. Refer to the Configuration Section on Stackable Authentication for further details.

Workflow System

The primary classes are:

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. WorkflowService	responds to events, manages the WorkflowItem states. There are two implementations, the traditional, default workflow (described below) and Configurable Workflow.
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 default 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. Optionally, you can also choose to enable the enhanced, Configurable Workflow, if you wish to have more control over your workflow steps/states. (*Note: the remainder of this description relates to the traditional, default workflow. For more information on the Configurable Workflow option, visit Configurable Workflow* .)

The WorkflowService is invoked by events. While an Item is being submitted, it is held by a WorkspaceItem. Calling the start() method in the WorkflowService 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 WorkflowService handles are advance(), which advances the WorkflowItem to the next state. If there are no further states, then the WorkflowItem is removed, and the Item is then archived. An EPerson performing one of the tasks can reject the Item, which stops the workflow, rebuilds the WorkspaceItem for it and sends a rejection note to the submitter. More drastically, an abort() event is generated by the admin tools to cancel a workflow outright.

Administration Toolkit

The *org.dspace.administer* package contains some classes for administering a DSpace system that are not generally needed by most applications.

The *CreateAdministrator* class is a simple command-line tool, executed via `[dspace]/bin/dspace create-administrator`, that creates an administrator e-person with information entered from standard input. This is generally used only once when a DSpace system is initially installed, to create an initial administrator who can then use the Web administration UI to further set up the system. This script does not check for authorization, since it is typically run before there are any e-people to authorize! Since it must be run as a command-line tool on the server machine, generally this shouldn't cause a problem. A possibility is to have the script only operate when there are no e-people in the system already, though in general, someone with access to command-line scripts on your server is probably in a position to do what they want anyway!

The *DCType* class is similar to the *org.dspace.content.BitstreamFormat* class. It represents an entry in the Dublin Core type registry, that is, a particular element and qualifier, or unqualified element. It is in the *administer* package because it is only generally required when manipulating the registry itself. Elements and qualifiers are specified as literals in *org.dspace.content.Item* methods and the *org.dspace.content.Metadatum* class. Only administrators may modify the Dublin Core type registry.

The *org.dspace.administer.RegistryLoader* class contains methods for initializing the Dublin Core type registry and bitstream format registry with entries in an XML file. Typically this is executed via the command line during the build process (see *build.xml* in the source.) To see examples of the XML formats, see the files in *config/registries* in the source directory. There is no XML schema, they aren't validated strictly when loaded in.

E-person/Group Manager

DSpace keeps track of registered users with the *org.dspace.eperson.EPerson* class. The class has methods to create and manipulate an *EPerson* such as get and set methods for first and last names, email, and password. (Actually, there is no *getPassword()* method, an MD5 hash of the password is stored, and can only be verified with the *checkPassword()* method.) There are find methods to find an EPerson by email (which is assumed to be unique,) or to find all 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 (for groups associated with workflows) 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.

Two specific groups are created when DSpace is installed: Administrator (which can bypass authorization) and Anonymous (which is assigned to all sessions that are not logged in). The code expects these groups to exist. They cannot be renamed or deleted.

Another kind of Group is also implemented in DSpace, special Groups. The *Context* object for each session carries around a List of Group IDs that the user is also a member of, currently the MITUser Group ID is added to the list of a user's special groups if certain IP address or certificate criteria are met.

Authorization

The primary classes are:

org.dspace.authorize.AuthorizeService	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.

Three new attributes have been introduced in the *ResourcePolicy* class as part of the DSpace Embargo Contribution:

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

While *rpname* and *rpdescription _are fields manageable by the users the _rptype* is a fields managed by the system. It represents a type that a resource policy can assume beteween the following:

- TYPE_SUBMISSION: all the policies added automatically during the submission process
- TYPE_WORKFLOW: all the policies added automatically during the workflow stage
- TYPE_CUSTOM: all the custom policies added by users
- TYPE_INHERITED: all the policies inherited by the *DSO* father.

An custom policy, created for the purpose of creating an embargo could look like:

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

The *AuthorizeService* class'
authorizeAction(Context, object, action) is the primary source of all authorization in the system. It gets a list of all of the ResourcePolicies in the system that match the object and action. It then iterates through the policies, extracting the EPerson Group from each policy, and checks to see if the EPersonID from the Context is a member of any of those groups. If all of the policies are queried and no permission is found, then an *AuthorizeException* is thrown. An *authorizeAction()* method is also supplied that returns a boolean for applications that require higher performance.

ResourcePolicies are very simple, and there are quite a lot of them. Each can only list a single group, a single action, and a single object. So each object will likely have several policies, and if multiple groups share permissions for actions on an object, each group will get its own policy. (It's a good thing they're small.)

Special Groups

All users are assumed to be part of the public group (ID=0.) DSpace admins (ID=1) are automatically part of all groups, much like super-users in the Unix OS. The Context object also carries around a List of special groups, which are also first checked for membership. These special groups are used at MIT to indicate membership in the MIT community, something that is very difficult to enumerate in the database! When a user logs in with an MIT certificate or with an MIT IP address, the login code adds this MIT user group to the user's Context.

Miscellaneous Authorization Notes

Where do items get their read policies? From the their collection's read policy. There once was a separate item read default policy in each collection, and perhaps there will be again since it appears that administrators are notoriously bad at defining collection's read policies. There is also code in place to enable policies that are timed, have a start and end date. However, the admin tools to enable these sorts of policies have not been written.

Handle Manager/Handle Plugin

The *org.dspace.handle* package contains two classes; *HandleService* 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.

HandleService 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.

In additional to Handles, DSpace also provides basic support for DOIs (Digital Object Identifiers). For more information visit DOI Digital Object Identifier.

Search

DSpace's search code is a simple, configurable API which currently wraps Apache Solr. See Discovery for more information on how to customize the default search settings, etc.

Harvesting API

The *org.dspace.search* package also provides a 'harvesting' API. This allows callers to extract information about items modified within a particular timeframe, and within a particular scope (all of DSpace, or a community or collection.) Currently this is used by the Open Archives Initiative metadata harvesting protocol application, and the e-mail subscription code.

The *Harvest.harvest* is invoked with the required scope and start and end dates. Either date can be omitted. The dates should be in the ISO8601, UTC time zone format used elsewhere in the DSpace system.

HarvestedItemInfo objects are returned. These objects are simple containers with basic information about the items falling within the given scope and date range. Depending on parameters passed to the *harvest* method, the *containers* and *item* fields may have been filled out with the IDs of communities and collections containing an item, and the corresponding *Item* object respectively. Electing not to have these fields filled out means the harvest operation executes considerable faster.

In case it is required, *Harvest* also offers a method for creating a single *HarvestedItemInfo* object, which might make things easier for the caller.

Browse API

The browse API uses the same underlying technology as the Search API (Apache Solr, see also Discovery). It 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 *B rowseScope* 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.

Checksum checker

Checksum checker is used to verify every item within DSpace. While DSpace calculates and records the checksum of every file submitted to it, the checker can determine whether the file has been changed. The idea being that the earlier you can identify a file has changed, the more likely you would be able to record it (assuming it was not a wanted change).

`org.dspace.checker.CheckerCommand` class, is the class for the checksum checker tool, which calculates checksums for each bitstream whose ID is in the *most_recent_checksum* table, and compares it against the last calculated checksum for that bitstream.

OpenSearch Support

DSpace is able to support the OpenSearch protocol. For those not acquainted with the standard, a very brief introduction, with emphasis on what possibilities it holds for current use and future development.

OpenSearch is a small set of conventions and documents for describing and using 'search engines', meaning any service that returns a set of results for a query. It is nearly ubiquitous, but also nearly invisible, in modern web sites with search capability. If you look at the page source of Wikipedia, Facebook, CNN, etc you will find buried a link element declaring OpenSearch support. It is very much a lowest-common-denominator abstraction (think Google box), but does provide a means to extend its expressive power. This first implementation for DSpace supports *none* of these extensions, many of which are of potential value, so it should be regarded as a foundation, not a finished solution. So the short answer is that DSpace appears as a 'search-engine' to OpenSearch-aware software.

Another way to look at OpenSearch is as a RESTful web service for search, very much like SRW/U, but considerably simpler. This comparative loss of power is offset by the fact that it is widely supported by web tools and players: browsers understand it, as do large metasearch tools.

How Can It Be Used

- Browser Integration: Many recent browsers (IE7+, FF2+) can detect, or 'autodiscover', links to the document describing the search engine. Thus you can easily add your or other DSpace instances to the drop-down list of search engines in your browser. This list typically appears in the upper right corner of the browser, with a search box. In Firefox, for example, when you visit a site supporting OpenSearch, the color of the drop-down list widget changes color, and if you open it to show the list of search engines, you are offered an opportunity to add the site to the list. IE works nearly the same way but instead labels the web sites 'search providers'. When you select a DSpace instance as the search engine and enter a search, you are simply sent to the regular search results page of the instance.
- Flexible, interesting RSS Feeds. Because one of the formats that OpenSearch specifies for its results is RSS (or Atom), you can turn any search query into an RSS feed. So if there are keywords highly discriminative of content in a collection or repository, these can be turned into a URL that a feed reader can subscribe to. Taken to the extreme, one could take any search a user makes, and dynamically compose an RSS feed URL for it in the page of returned results. To see an example, if you have a DSpace with OpenSearch enabled, try:

  ```
  # The Opensearch feature is available from the dspace.server.url
  [dspace.server.url]/opensearch/search?query=<your query>
  # e.g. https://demo.dspace.org/server/opensearch/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	UUID 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	metadata field to sort by, for example "dc.title" (see `webui.itemlist.sort-option` in `dspace.cfg`)
sort_direction	ASC or DESC (default ASC)

Multiple parameters may be specified on the query string, using the "&" character as the delimiter, e.g.:

```
https://demo.dspace.org/server/opensearch/search?query=<your query>&format=rss&scope=e9af9bd8-aca9-4efa-8bd8-1edb5fe1ac4c
```

- 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.
- When OpenSearch is enabled in DSpace, informational "link" tags will be embedded into the HTML of every page. These link tags will allow tools to easily discover the OpenSearch Service Document (`/service`) and RSS / Atom feeds. The links appear in the HTML head tag and look like this:

```
<link href="https://demo.dspace.org/server/opensearch/search?format=atom&query=*" type="application/atom+xml" rel="alternate" title="Sitewide Atom feed">
<link href="https://demo.dspace.org/server/opensearch/search?format=rss&query=*" type="application/rss+xml" rel="alternate" title="Sitewide RSS feed">
<link href="https://demo.dspace.org/server/opensearch/search/service" type="application/atom+xml" rel="search" title="DSpace">
```

Configuration is through the `dspace.cfg` file. See OpenSearch Support for more details.

Embargo Support

What is an Embargo?

An embargo is a temporary access restriction placed on content, commencing at time of accession. 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.

More Embargo Details



More details on Embargo configuration, including specific examples can be found in the Embargo section of the documentation.

DSpace Services Framework

1 Architectural Overview
- 1.1 DSpace Kernel
  - 1.1.1 Kernel registration
- 1.2 Service Manager

2 Basic Usage
- 2.1 Standalone Applications
- 2.2 Application Frameworks (Spring, Guice, etc.)
- 2.3 Web Applications

3 Providers and Plugins
- 3.1 Activators
- 3.2 Provider Stacks

4 Core Services
- 4.1 Caching Service
- 4.2 Configuration Service
- 4.3 EventService
- 4.4 RequestService
- 4.5 SessionService

5 Examples
- 5.1 Configuring Event Listeners

6 Tutorials

The DSpace Services Framework is a backporting of the DSpace 2.0 Development Group's work in creating a reasonable and abstractable "Core Services" layer for DSpace components to operate within. The Services Framework represents a "best practice" for new DSpace architecture and implementation of extensions to the DSpace application. DSpace Services are best described as a "Simple Registry" where plugins can be "looked up" or located. The DS2 (DSpace 2.0) core services are the main services that make up a DS2 system. These includes services for things like user and permissions management and storage and caching. These services can be used by any developer writing DS2 plugins (e.g. statistics), providers (e.g. authentication), or user interfaces.

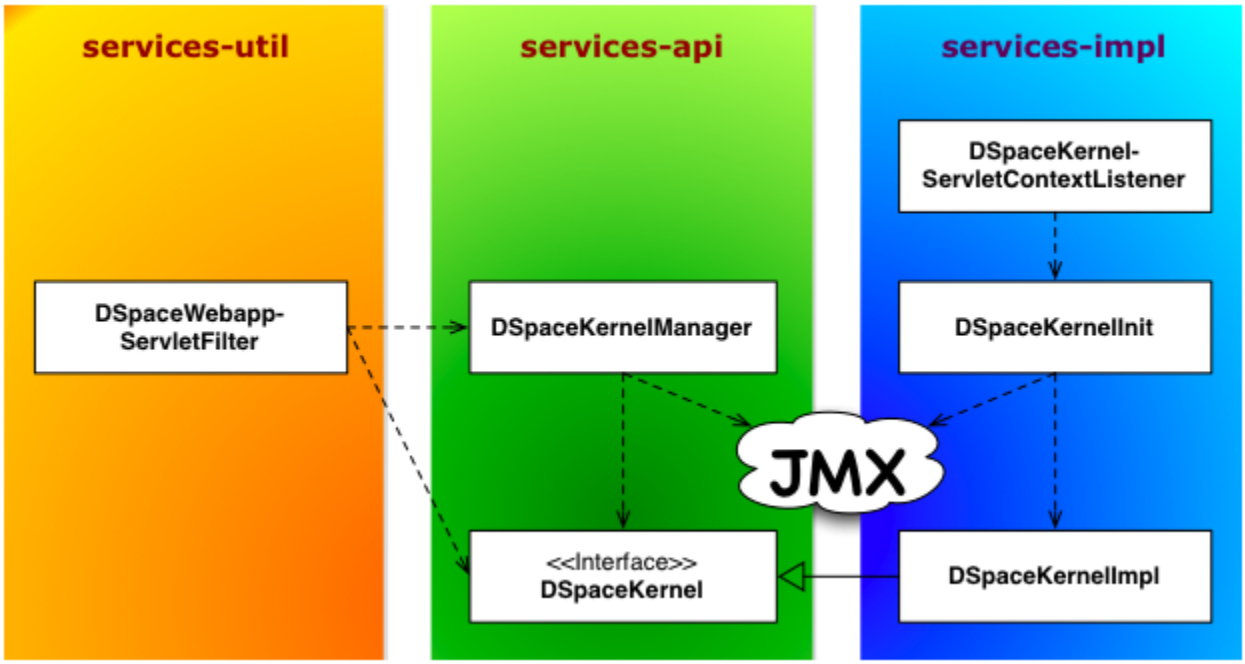
Architectural Overview

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 UI 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.

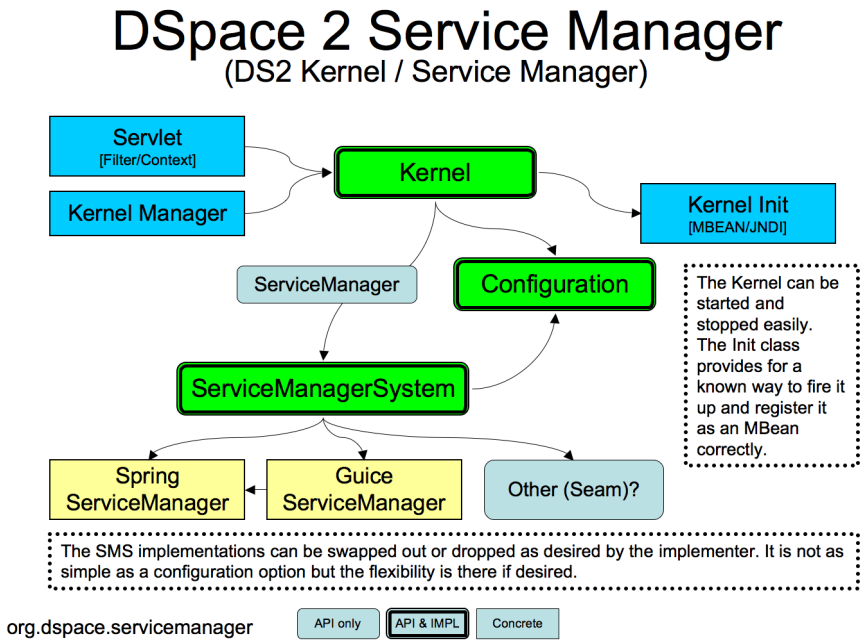
Kernel registration

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.

Basic Usage

To use the Framework you must begin by instantiating and starting a DSpaceKernel. The kernel will give you references to the ServiceManager and the ConfigurationService. The ServiceManager can be used to get references to other services and to register services which are not part of the core set.

Access to the kernel is provided via the Kernel Manager through the DSpace object, which will locate the kernel object and allow it to be used.

Standalone Applications

For standalone applications, access to the kernel is provided via the Kernel Manager and the DSpace object which will locate the kernel object and allow it to be used.

```
/* Instantiate the Utility Class */
DSpace dspace = new DSpace();


/* Access get the Service Manager by convenience method */
ServiceManager manager = dspace.getServiceManager();


/* Or access by convenience method for core services */
EventService service = dspace.getEventService();
```

The DSpace launcher (

```
bin/dspace
```

) initializes a kernel before dispatching to the selected command.

Application Frameworks (Spring, Guice, etc.)

Similar to Standalone Applications, but you can use your framework to instantiate an `org.dspace.utils.DSpace` object.

```
<bean id="dspace" class="org.dspace.utils.DSpace"/>
```

Web Applications

In web applications, the kernel can be started and accessed through the use of Servlet Filter/ContextListeners which are provided as part of the DSpace 2 utilities. Developers don't need to understand what is going on behind the scenes and can simply write their applications and package them as webapps and take advantage of the services which are offered by DSpace 2.

Providers and Plugins

For developers (how we are trying to make your lives easier): The DS2 ServiceManager supports a plugin/provider system which is runtime hot-swappable. The implementor can register any service/provider bean or class with the DS2 kernel ServiceManager. The ServiceManager will manage the lifecycle of beans (if desired) and will instantiate and manage the lifecycle of any classes it is given. This can be done at any time and does not have to be done during Kernel startup. This allows providers to be swapped out at runtime without disrupting the service if desired. The goal of this system is to allow DS2 to be extended without requiring any changes to the core codebase or a rebuild of the code code.

Activators

Developers can provide an activator to allow the system to startup their service or provider. It is a simple interface with 2 methods which are called by the ServiceManager to startup the provider(s) and later to shut them down. These simply allow a developer to run some arbitrary code in order to create and register services if desired. It is the method provided to add plugins directly to the system via configuration as the activators are just listed in the configuration file and the system starts them up in the order it finds them.

Provider Stacks

Utilities are provided to assist with stacking and ordering providers. Ordering is handled via a priority number such that 1 is the highest priority and something like 10 would be lower. 0 indicates that priority is not important for this service and can be used to ensure the provider is placed at or near the end without having to set some arbitrarily high number.

Core Services

The core services are all behind APIs so that they can be reimplemented without affecting developers who are using the services. Most of the services have plugin/provider points so that customizations can be added into the system without touching the core services code. For example, let's say a deployer has a specialized authentication system and wants to manage the authentication calls which come into the system. The implementor can simply implement an AuthenticationProvider and then register it with the DS2 kernel's ServiceManager. This can be done at any time and does not have to be done during Kernel startup. This allows providers to be swapped out at runtime without disrupting the DS2 service if desired. It can also speed up development by allowing quick hot redeploys of code during development.

Caching Service

Provides for a centralized way to handle caching in the system and thus a single point for configuration and control over all caches in the system. Provider and plugin developers are strongly encouraged to use this rather than implementing their own caching. The caching service has the concept of scopes so even storing data in maps or lists is discouraged unless there are good reasons to do so.

Configuration Service

The ConfigurationService controls the external and internal configuration of DSpace 2. It reads Properties files when the kernel starts up and merges them with any dynamic configuration data which is available from the services. This service allows settings to be updated as the system is running, and also defines listeners which allow services to know when their configuration settings have changed and take action if desired. It is the central point to access and manage all the configuration settings in DSpace 2.

Manages the configuration of the DSpace 2 system. Can be used to manage configuration for providers and plugins also.

EventService

Handles events and provides access to listeners for consumption of events.

RequestService

In DS2 a request is an atomic transaction in the system. It is likely to be an HTTP request in many cases but it does not have to be. This service provides the core services with a way to manage atomic transactions so that when a request comes in which requires multiple things to happen they can either all succeed or all fail without each service attempting to manage this independently. In a nutshell this simply allows identification of the current request and the ability to discover if it succeeded or failed when it ends. Nothing in the system will enforce usage of the service, but we encourage developers who are interacting with the system to make use of this service so they know if the request they are participating in with has succeeded or failed and can take appropriate actions.

SessionService

In DS2 a session is like an HttpSession (and generally is actually one) so this service is here to allow developers to find information about the current session and to access information in it. The session identifies the current user (if authenticated) so it also serves as a way to track user sessions. Since we use HttpSession directly it is easy to mirror sessions across multiple servers in order to allow for no-interruption failover for users when servers go offline.

Examples

Configuring Event Listeners

Event Listeners can be created by overriding the the EventListener interface:

In Spring:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>

    <bean id="dspace" class="org.dspace.utils.DSpace"/>

    <bean id="dspace.eventService"
          factory-bean="dspace"
          factory-method="getEventService"/>

    <bean class="org.my.EventListener">
         <property name="eventService" >
                   <ref bean="dspace.eventService"/>
            </property>
    </bean>
</beans>
```

(org.my.EventListener will need to register itself with the EventService, for which it is passed a reference to that service via the eventService property.)

or in Java:

```
DSpace dspace = new DSpace();

EventService eventService = dspace.getEventService();

EventListener listener = new org.my.EventListener();
eventService.registerEventListener(listener);
```

(This registers the listener externally – the listener code assumes it is registered.)

Tutorials

Several tutorials on Spring / DSpace Services are available:

- DSpace Spring Services Tutorial
- The TAO of DSpace Services

Storage Layer

In this section, we explain the storage layer: the database structure, maintenance, and the bitstream store and configurations. The bitstream store, also known as assetstore or bitstore, holds the uploaded, ingested, or generated files (documents, images, audio, video, datasets, ...), where as the database holds all of the metadata, organization, and permissions of content.

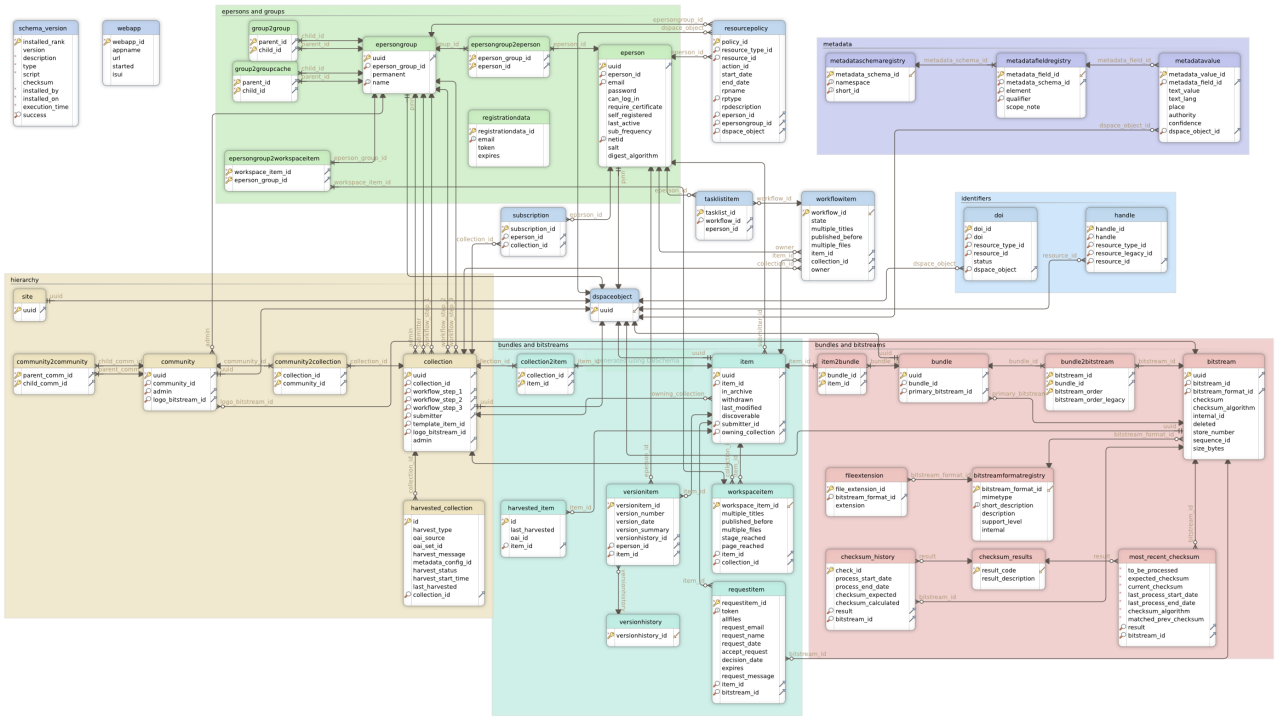
1 RDBMS / Database Structure
- 1.1 Maintenance and Backup
- 1.2 Configuring the RDBMS Component
- 1.3 Custom RDBMS tables, columns or views

2 Bitstream Store
- 2.1 Cleanup
  - 2.1.1 Cleanup Options
  - 2.1.2 Cleanup when copy requests are pending
- 2.2 Backup
- 2.3 Configuring the Bitstream Store
  - 2.3.1 Configuring Traditional Storage
  - 2.3.2 Configuring Amazon S3 Storage
  - 2.3.3 Configuring JCloudstore for Asset Storage
    - 2.3.3.1 Choosing a Provider and Authentication
    - 2.3.3.2 AWS S3 Configuration Example
    - 2.3.3.3 Third Party S3 Configuration
- 2.4 Migrate BitStores

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.

DSpace 6 database schema (Postgres). Right-click the image and choose "Save as" to save in full resolution. Instructions on updating this schema diagram are in How to update database schema diagram.



- DSpace uses FlywayDB to perform automated database initialization and upgrades. Flyway's role is to initialize the database tables (and default content) prior to Hibernate initialization.
  - The `org.dspace.storage.rdbms.DatabaseUtils` class manages all Flyway API calls, and executes the SQL migrations under the `org.dspace.storage.rdbms.sqlmigration` package and the Java migrations under the `org.dspace.storage.rdbms.migration` package.
  - Once all database migrations have run, a series of Flyway Callbacks are triggered to initialize the (empty) database with required default content. For example, callbacks exist for adding default DSpace Groups (`GroupServiceInitializer`), default Metadata & Format Registries (`DatabaseRegistryUpdater`), and the default Site object (`SiteServiceInitializer`). All Callbacks are under the `org.dspace.storage.rdbms` package.

    - While Flyway is automatically initialized and executed during startup, various Database Utilities are also available on the command line. These utilities allow you to manually trigger database upgrades or check the status of your database.
- DSpace uses Hibernate ORM as the object relational mapping layer between the DSpace database and the DSpace code.
    - The main Hibernate configuration can be found at `[dspace]/config/hibernate.cfg.xml`
    - Hibernate initialization is triggered via Spring (beans) defined `[dspace]/config/spring/api/core-hibernate.xml`. This Spring configuration pulls in some settings from DSpace Configuration, namely all Database (db.*) settings defined there.
    - All DSpace Object Classes provide a DAO (Data Access Object) implementation class that extends a GenericDAO interface defined in o `rg.dspace.core.GenericDAO` class. The default (abstract) implementation is in `org.dspace.core.AbstractHibernateDAO` cla ss.
    - The DSpace Context object (`org.dspace.core.Context`) provides access to the configured `org.dspace.core.DBConnection` (D atabase Connection), which is HibernateDBConnection by default. The `org.dspace.core.HibernateDBConnection` class provides access the the Hibernate Session interface (org.hibernate.Session) and its Transactions.
        - Each Hibernate Session opens a single database connection when it is created, and holds onto it until the Session is closed. A Session may consist of one or more Transactions. Sessions are NOT thread-safe (so individual objects cannot be shared between threads).
        - Hibernate will intelligently cache objects in the current Hibernate Session (on object access), allowing for optimized performance.
        - DSpace provides methods on the Context object to specifically remove (`Context.uncacheEntity()`) or reload (`Context. reloadEntity()`) objects within Hibernate's Session cache.
        - DSpace also provides special Context object "modes" to optimize Hibernate performance for read-only access (`Mode. READ_ONLY`) or batch processing (`Mode.BATCH_EDIT`). These modes can be specified when constructing a new Context object.

Most of the functionality that DSpace uses can be offered by any standard SQL database that supports transactions. However, at this time, DSpace only provides Flyway migration scripts for PostgreSQL and Oracle (and has only been tested with those database backends). Additional database backends should be possible, but would minimally require creating custom Flyway migration scripts for that database backend.

Maintenance and Backup

When using PostgreSQL, it's a good idea to perform regular 'vacuuming' of the database to optimize performance. By default, PostgreSQL performs autom atic vacuuming on your behalf. However, if you have this feature disabled, then we recommend scheduling the `vacuumdb` command to run on a regular basis.

```
# clean up the database nightly
40 2 * * * /usr/local/pgsql/bin/vacuumdb --analyze dspace > /dev/null 2>&1
```

Backups: The DSpace database can be backed up and restored using usual PostgreSQL Backup and Restore methods, for example with `pg_dump` and `p sql`. However when restoring a database, you will need to perform these additional steps:

- After restoring a backup, you will need to reset the primary key generation sequences so that they do not produce already-used primary keys. Do this by executing the SQL in `[dspace]/etc/postgres/update-sequences.sql`, for example with:

  ```
  psql -U dspace -f [dspace]/etc/update-sequences.sql
  ```

Configuring the RDBMS Component

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

`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.

Custom RDBMS tables, columns or views

When at all possible, we recommend creating custom database tables or views within a *separate schema* from the DSpace database tables. Since the DSpace database is initialized and upgraded automatically using Flyway DB, the upgrade process may stumble or throw errors if you've directly modified the DSpace database schema, views or tables. Flyway itself assumes it has full control over the DSpace database schema, and it is not "smart" enough to know what to do when it encounters a locally customized database.

That being said, if you absolutely need to customize your database tables, columns or views, it is possible to create *custom Flyway migration scripts* , which should make your customizations easier to manage in future upgrades. (Keep in mind though, that you may still need to maintain/update your

custom Flyway migration scripts if they ever conflict directly with future DSpace database changes. The only way to "future proof" your local database changes is to try and make them as independent as possible, and avoid directly modifying the DSpace database schema as much as possible.)

If you wish to add custom Flyway migrations, they may be added to the following locations:

- Custom Flyway SQL migrations may be added anywhere under the `org.dspace.storage.rdbms.sqlmigration` package (e.g. `[src]/dspace-api/src/main/resources/org/dspace/storage/rdbms/sqlmigration` or subdirectories)
- Custom Flyway Java migrations may be added anywhere under the `org.dspace.storage.rdbms.migration` package (e.g. `[src]/dspace-api/src/main/java/org/dspace/storage/rdbms/migration/` or subdirectories)
- Additionally, for backwards support, custom SQL migrations may also be placed in the `[dspace]/etc/[db-type]/` folder (e.g. `[dspace]/etc/postgres/` for a PostgreSQL specific migration script)

Adding Flyway migrations to any of the above location will cause Flyway to auto-discover the migration. It will be run in the order in which it is named. Our DSpace Flyway script naming convention follows Flyway best practices and is as follows:

- SQL script names: `V[version]_[date]__[description].sql`
  - E.g. `V5.0_2014.09.26__DS-1582_Metadata_For_All_Objects.sql` is a SQL migration script created for DSpace 5.x (`V5.0`) on Sept 26, 2014 (`2014_09_24`). Its purpose was to fulfill the needs of ticket DS-1582, which was to migrate the database in order to support adding metadata on all objects.
  - More examples can be found under the `org.dspace.storage.rdbms.sqlmigration` package
- Java migration script naming convention: `V[version]_[date]__[description].java`
  - E.g. `V5_0_2014_09_25__DS_1582_Metadata_For_All_Objects_drop_constraint.java` is a Java migration created for DSpace 5.x (`V5_0`) on Sept 25, 2014 (`2014_09_25`). Its purpose was to fulfill the needs of ticket DS-1582, specifically to drop a few constraints.
  - More examples can be found under the `org.dspace.storage.rdbms.migration` package
- Flyway will execute migrations in order, based on their Version and Date. So, V1.x (or V1_x) scripts are executed first, followed by V3.0 (or V3_0), etc. If two migrations have the same version number, the date is used to determine ordering (earlier dates are run first).

Bitstream Store

DSpace offers two means for storing content.

1. Storage in a mounted file system on the server (DSBitStore)
2. Storage using AWS S3 (Simple Storage Service), (S3BitStore).

Both are achieved using a simple, lightweight BitStore API, providing actions of Get, Put, About, Remove. Higher level operations include Store, Register, Checksum, Retrieve, Cleanup, Clone, Migrate. Digital assets are stored on the bitstores by being transferred to the bitstore when it is uploaded or ingested. The exception to this is for "registered" objects, that the assets are put onto the filesystem ahead of time out-of-band, and during ingest, it just maps the database to know where the object already resides. The storage interface is such that additional storage implementations (i.e. other cloud storage providers) can be added with minimal difficulty.

DSBitStore stores content on a path on the filesystem. This could be locally attached normal filesystem, a mounted drive, or a mounted networked filesystem, it will all be treated as a local filesystem. All DSpace needs to be configured with for a filesystem, is the filesystem path, i.e. /dspace/assetstore, /opt/data/assetstore. The DSBitStore uses a "Directory Scatter" method of storing an asset within 3 levels of subfolders, to minimize any single folder having too many objects for normal filesystem performance.

S3BitStore uses Amazon Web Services S3 (Simple Storage Service) to offer limitless cloud storage into a bucket, and each distinct asset will have a unique key. S3 is a commercial service (costs money), but is available at low price point, and is fully managed, content is automatically replicated, 99.999999999% object durability, integrity checked. Since S3 operates within the AWS network, using other AWS services, such virtual server on EC2 will provide lower network latency than local "on premises" servers. Additionally there could be some in-bound / out-bound bandwidth costs associated with DSpace application server outside of the AWS network communicating with S3, compared to AWS-internal EC2 servers. S3 has a checksum computing operation, in which the S3 service can return the checksum from the storage service, without having to shuttle the bits from S3, to your application server, and then computing the checksum. S3BitStore requires an S3 bucketName, accessKey, secretKey, and optionally specifying the AWS region, or a subfolder within the bucket.

There can be multiple bitstream stores. Each of these bitstream stores can be traditional storage or S3 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 filesystem and S3storage can be combined in one DSpace installation. Both filesystem and S3 storage are specified by configuration. 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. An example of having multiple asset stores configured is that assetstore0 is /dspace/assetstore, when the filesystem gets nearly full, you could then configure a second filesystem path assetstore1 at /data/assetstore1, later, if you wanted to use S3 for storage, assetstore2 could be s3://dspace-assetstore-xyz. In this example various bitstreams (database objects) refer to different assetstore for where the files reside. It is typically simplest to just have a single assetstore configured, and all assets reside in that one. If policy dictated, infrequently used masters could be moved to slower/cheaper disk, where as access copies are on the fastest storage. This could be accomplished through migrating assets to different stores.

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 storage. The first three pairs of digits are the directory path that the bitstream is stored under. The bitstream is stored in a file with the internal ID as the filename.

For example, a bitstream with the internal ID *12345678901234567890123456789012345678* is stored in the directory:

```
[dspace]/assetstore/12/34/56/12345678901234567890123456789012345678
```

The reasons for storing files this way are:

- Using a randomly-generated 38-digit number means that the 'number space' is less cluttered than simply using the primary keys, which are allocated sequentially and are thus close together. This means that the bitstreams in the store are distributed around the directory structure, improving access efficiency.
- The internal ID is used as the filename partly to avoid requiring an extra lookup of the filename of the bitstream, and partly because bitstreams may be received from a variety of operating systems. The original name of a bitstream may be an illegal UNIX filename.
- When storing a bitstream, the *BitstreamStorageService* DOES set the following fields in the corresponding database table row:
    - *bitstream_id*
    - *size*
    - *checksum*
    - *checksum_algorithm*
    - *internal_id*
    - *deleted*
    - *store_number*

- The remaining fields are the responsibility of the *Bitstream* content management API class.

The bitstream storage manager is fully transaction-safe. In order to implement transaction-safety, the following algorithm is used to store bitstreams:

1. A database connection is created, separately from the currently active connection in the current DSpace context.
2. An unique internal identifier (separate from the database primary key) is generated.
3. The bitstream DB table row is created using this new connection, with the *deleted* column set to *true*.
4. The new connection is _commit_ted, so the 'deleted' bitstream row is written to the database
5. The bitstream itself is stored in a file in the configured 'asset store directory', with a directory path and filename derived from the internal ID
6. The *deleted* flag in the bitstream row is set to *false*. This will occur (or not) as part of the current DSpace *Context*.

This means that should anything go wrong before, during or after the bitstream storage, only one of the following can be true:

- No bitstream table row was created, and no file was stored
- A bitstream table row with *deleted=true* was created, no file was stored
- A bitstream table row with *deleted=true* was created, and a file was stored
  None of these affect the integrity of the data in the database or bitstream store.

Similarly, when a bitstream is deleted for some reason, its *deleted* flag is set to true as part of the overall transaction, and the corresponding file in storage is *not* deleted.

Cleanup

The above techniques mean that the bitstream storage manager is transaction-safe. Over time, the bitstream database table and file store may contain a number of 'deleted' bitstreams. The *cleanup* method of *BitstreamStorageService* goes through these deleted rows, and actually deletes them along with any corresponding files left in the storage. It only removes 'deleted' bitstreams that are more than one hour old, just in case cleanup is happening in the middle of a storage operation.

This cleanup can be invoked from the command line via the *cleanup* command, which can in turn be easily executed from a shell on the server machine using `[dspace]/bin/dspace cleanup`. You might like to have this run regularly by *cron*, though since DSpace is read-lots, write-not-so-much it doesn't need to be run very often.

```
# Clean up any deleted files from local storage on first of the month at 2:40am
40 2 1 * * [dspace]/bin/dspace cleanup > /dev/null 2>&1
```

Cleanup Options

Command used:	`[dspace]/bin/dspace cleanup`
Java class:	org.dspace.storage.bitstore.Cleanup
Arguments short and (long) forms:	Description
`-l` or `--leave`	Leave database records but delete file from assetstore. Deletes files from storage system, but doesn't remove entries marked as deleted from table `bitstream`.
`-v` or `--verbose`	Provide verbose output.
`-h` or `--help`	Show usage information.

Cleanup when copy requests are pending

If a bitstream was deleted while a "request a copy" request was still pending for that bitstream, the cleanup command will fail unless the `-l` option was specified.

This is because the table `requestitem` table uses the bitstream's UUID as a foreign key constraint and thus the database prohibits deleting the bitstreams entry in table `bitstreams` .

Backup

The bitstreams (files) in traditional storage may be backed up very easily by simply 'tarring' or 'zipping' the `[dspace]/assetstore/` directory (or whichever directory is configured in *dspace.cfg*). Restoring is as simple as extracting the backed-up compressed file in the appropriate location.

It is important to note that since the bitstream storage manager holds the bitstreams in storage, and information about them in the database, that a database backup and a backup of the files in the bitstream store must be made at the same time; the bitstream data in the database must correspond to the stored files.

Of course, it isn't really ideal to 'freeze' the system while backing up to ensure that the database and files match up. Since DSpace uses the bitstream data in the database as the authoritative record, it's best to back up the database before the files. This is because it's better to have a bitstream in storage but not the database (effectively non-existent to DSpace) than a bitstream record in the database but not storage, since people would be able to find the bitstream but not actually get the contents.

With DSpace 1.7 and above, there is also the option to backup both files and metadata via the AIP Backup and Restore feature.

Configuring the Bitstream Store

Changed in DSpace 7.4

While the old **bitstore.xml** file (defined at `[dspace]/config/spring/api/bitstore.xml`) still exists in DSpace 7.4, a new configuration file has been added for configuring the Bitstream Store: `[dspace]/config/modules/assetstore.cfg`

If you have previously configured **bitstore.xml**, and only have a single assetstore, we recommend resetting **bitstore.xml** to the default configuration, and use the new **assetstore.cfg** file for configuring your Bitstream Store. This is of course very general advice, and your specific situation may require more care, particularly if you have more than one Bitstream Store (see below, and this likely doesn't apply to you, but it could).

BitStores (aka assetstores) are configured with `[dspace]/config/modules/assetstore.cfg`

Configuring Traditional Storage

By default, DSpace uses a traditional filesystem bitstore of `[dspace]/assetstore/`

To configure traditional filesystem bitstore, as a specific directory, configure the bitstore like this:

```
#---------------------------------------------------------------#
#-----------------STORAGE CONFIGURATIONS------------------------#
#---------------------------------------------------------------#
# Configuration properties used by the bitstore.xml config file #
#                                                               #
#---------------------------------------------------------------#

# assetstore.dir, look at DSPACE/config/spring/api/bitstore.xml for more options
assetstore.dir = ${dspace.dir}/assetstore

# Configures the primary store to be local or S3.
# This value will be used as `incoming` default store inside the `bitstore.xml`
# Possible values are:
#     - 0: to use the `localStore`;
#     - 1: to use the `s3Store`.
# If you want to add additional assetstores, they must be added to that bitstore.xml
# and new values should be provided as key-value pairs in the `stores` map of the
# `bitstore.xml` configuration.
assetstore.index.primary = 0
```

This would configure store number 0 named localStore, which is a `DSBitStore` (filesystem), at the filesystem path of `${dspace.dir}/assetstore` (i.e. `[dspace]/assetstore/`)

It is also possible to use multiple local filesystems. The following example is specific to the older **bitstore.xml** configuration, and it should still work, but is un-tested with DSpace 7.4. In the example below, key #0 is localStore at `${dspace.dir}/assetstore`, and key #1 is localStore2 at `/data/assetstore2`. Note that incoming is set to store "1", which in this case refers to localStore2. That means that any new files (bitstreams) uploaded to DSpace will be stored in localStore2, but some existing bitstreams may still exist in localStore.

```
<bean name="org.dspace.storage.bitstore.BitstreamStorageService" class="org.dspace.storage.bitstore.
BitstreamStorageServiceImpl">
    <property name="incoming" value="1"/>
    <property name="stores">
        <map>
            <entry key="0" value-ref="localStore"/>
            <entry key="1" value-ref="localStore2"/>
        </map>
```

```
    </property>
</bean>
<bean name="localStore" class="org.dspace.storage.bitstore.DSBitStoreService" scope="singleton">
    <property name="baseDir" value="${dspace.dir}/assetstore"/>
</bean>
<bean name="localStore2" class="org.dspace.storage.bitstore.DSBitStoreService" scope="singleton">
    <property name="baseDir" value="/data/assetstore2"/>
</bean>
```

Configuring Amazon S3 Storage

To use Amazon S3 as a bitstore, in **assetstore.cfg** point **assetstore.index.primary** to 1, set **assetstore.s3.enabled** to true, and configure it with **awsAccessKey**, **awsSecretKey**, and **bucketName**. NOTE: Before you can specify these settings, you obviously will have to create an account in the Amazon AWS console, and create an IAM user with credentials and privileges to an existing S3 bucket. You can also use an IAM Role, which would allow your Amazon EC2 instance to talk directly to your S3 bucket without using any credentials. If you do configure an IAM Role with AWS, you'll still need to provide values for **awsAccessKey** and **awsSecretKey**, but they are un-used by DSpace, and can be anything at all. In the example below, we assume you are using an IAM Role, and are providing the text "use-the-role-please" as the value for these two keys. This is of course not a valid AWS key, for either example.

You can customize download speed and memory footprint of **s3Store** by tuning these properties:

- **assetstore.s3.memoryUsageFactor** - Determines the percentage of **Xmx** parameter that can be used during the download of a bitstream (defaults to 10% - **change at your own risk**)
- **assetstore.s3.maxConcurrency** - Determines the number of threads that would be used during the download of a bitstream (defaults to CPU cores of the machine)
- **assetstore.s3.initialReadBufferSizeInBytes** - Determines the **bufferSize** that each thread can pre-allocate and use during the download of a bitstream (will be determined by a custom heuristic / AWS SDK)

Whenever the **assetstore.s3.memoryUsageFactor** is set, other parameters listed would be computed using a custom heuristic: https://github.com/DSpace/DSpace/issues/12468#issuecomment-4453332457

Whenever not specified, you can set the other two parameters or just rely on AWS SDK defaults.



We suggest to keep the default settings, since they've been tested on different Xmx sizes and performs well.

You can find some reference values in this issue: https://github.com/DSpace/DSpace/issues/12468#issuecomment-4459335016 and adapt those to your needs, just keep in mind that OOM can occur if not properly tuned.

```
# Configures the primary store to be local or S3.
# This value will be used as `incoming` default store inside the `bitstore.xml`
# Possible values are:
#     - 0: to use the `localStore`;
#     - 1: to use the `s3Store`.
# If you want to add additional assetstores, they must be added to that bitstore.xml
# and new values should be provided as key-value pairs in the `stores` map of the
# `bitstore.xml` configuration.
assetstore.index.primary = 1

#----------------------------------------------------------------#
#-------------- Amazon S3 Specific Configurations --------------#
#----------------------------------------------------------------#
# The below configurations are only used if the primary storename
# is set to 's3Store' or the 's3Store' is configured as a secondary store
# in your bitstore.xml

# Enables or disables the store initialization during startup, without initialization the store won't work.
# if changed to true, a lazy initialization will be tried on next store usage, be careful an excecption could
be thrown
assetstore.s3.enabled = true

# For using a relative path (xx/xx/xx/xxx...) set to true, default it false
# When true: it splits the path into subfolders, each of these
# are 2-chars (2-bytes) length, the last is the filename and could have
# at max 3-chars (3-bytes).
# When false: is used the absolute path using full filename.
assetstore.s3.useRelativePath = false

# S3 bucket name to store assets in. If unspecified, by default DSpace will
# create a bucket based on the hostname of `dspace.ui.url` setting.
assetstore.s3.bucketName = your-bucket-name-goes-here
```

```
# Subfolder to organize assets within the bucket, in case this bucket
# is shared. Optional, default is root level of bucket
assetstore.s3.subfolder = your-optional-subfolder-goes-here

# Optional custom S3 endpoint URI. Leave this blank / commented to use the Amazon default
# assetstore.s3.endpoint =

# please don't use root credentials in production but rely on the aws credentials default
# discovery mechanism to configure them (ENV VAR, EC2 Iam role, etc.)
# The preferred approach for security reason is to use the IAM user credentials, but isn't always possible.
# More information about credentials here: https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide
/credentials.html
# More information about IAM usage here: https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/java-dg-
roles.html
assetstore.s3.awsAccessKey = use-the-role-please
assetstore.s3.awsSecretKey = use-the-role-please

# If the credentials are left empty,
# then this setting is ignored and the default AWS region will be used.
# NOTE: AWS is funny about regions, it's probably best to explicitly provide the one you intend to use,
# if you want to keep track of where your bitstreams are
assetstore.s3.awsRegionName = us-west-2

# The target throughput for transfer requests in Gbps. Higher value means more connections will be established
with S3.
assetstore.s3.targetThroughputGbps = 10.0

# Sets the minimum part size for transfer parts. Decreasing the minimum part size causes multipart transfer to
be split
# into a larger number of smaller parts.
assetstore.s3.minPartSizeBytes = 8388608

# Specifies the maximum number of S3 connections that should be established during a transfer.
# If not provided, it will be based on targetThroughputGbps
assetstore.s3.maxConcurrency =

# The algorithm the S3 client will use to create a checksum when doing putObject.
assetstore.s3.s3ChecksumAlgorithm = CRC32

# The amount of maximum memory used during data transfers, as a percentage of the total available memory.
# Default is 0.1 (10% of total memory - Xmx). Adjust this value based on the available memory and expected
transfer sizes.
# This value should be set between 0 and 1, where 0 means no memory usage limit and 1 means using all available
memory.
# BE careful with this value, since it can cause OOM - especially for lower Xmx values ( 256m / 512m )
# If not set, there is no memory usage limit and the S3 client will use as much memory as needed for transfers,
# which may lead to OutOfMemory errors if the transfer sizes are large and the available memory is limited.
assetstore.s3.memoryUsageFactor=0.1
```

Configuring JCloudstore for Asset Storage

Unfortunately, as of June 18, 2025, Apache JClouds has been retired and is no longer under support. We therefore do not recommend using it in your DSpace site. See also https://github.com/DSpace/DSpace/issues/11294

JCloudstore allows you to store assets in various cloud storage providers like AWS S3, Azure Blob Storage, and MinIO. JCloudstore supports multiple providers, including `aws-s3`, `s3`, `filesystem`, `azureblob`, and `google-cloud-storage`. Providers that are supported are available in the JClouds site/documentation https://jclouds.apache.org/reference/providers/#blobstore

To enable it, set the following property:

```
### JCloudSettings
# Configuration for JCloudstore, see config/spring/api/bitstore.xml for more options

# Enables or disables the JCloudstore asset store.
# Default: false
assetstore.jcloud.enabled = true
```

By default, this is `false`, meaning JCloudstore is disabled unless explicitly enabled.

```
# The provider for the remote Blob service. See for additional services
# https://jclouds.apache.org/reference/providers/
# Note configuration may require additional jar dependencies be added to dspace-api/pom.xml
# Examples: aws-s3, s3, filesystem, azureblob, google-cloud-storage
# Default: aws-s3
# assetstore.jcloud.provider = aws-s3

# Identity or key for remote store
# Examples: awsAccessKey
# Default: (empty) requirement depends on service provider
# assetstore.jcloud.identity =

# Credentials or secret key for JCloudstore.
# Examples: awsSecretKey
# This key is used in conjunction with the identity to authenticate with remote service.
# Default: (empty) requirement depends on service provider
# assetstore.jcloud.credentials =

# The endpoint URL for the S3 service.
# Example: when using "s3" and third party provider such as Minio, https://s3.myprovider.com
# Default: (empty) requirement depends on service provider
# assetstore.jcloud.endpoint =

# The name of the container (S3 Bucket) to use for storing assets.
# Default: (empty) requirement depends on service provider
# assetstore.jcloud.container =

# Specifies a subfolder within the container for storing assets
# Default: (empty)
# assetstore.jcloud.subfolder =

# Determines whether to use relative hash paths (12/34/56/1234567..) for storing bitstreams
# Default: false
# assetstore.jcloud.useRelativePath = false

# The maximum counter value for operations before the BlobStoreContext is forcibly reset
# (sometimes beneficial in cases where connection pool errors start to happen with a provider).
# Default: -1 (unlimited) connection is never manually closed in BitstoreService
# assetstore.jcloud.maxCounter = -1
```

Choosing a Provider and Authentication

JCloudstore supports multiple providers, including `aws-s3`, `s3`, `filesystem`, `azureblob`, and `google-cloud-storage`. Providers that are supported are available in the JClouds site/documentation https://jclouds.apache.org/reference/providers/#blobstore

(Note, some providers like Azure or Google Cloud may require additional dependencies be added in the maven pom.xml)

AWS S3 Configuration Example

```
# Set the provider:
assetstore.jcloud.provider = aws-s3

# Configure AWS credentials:
assetstore.jcloud.identity = <your-aws-access-key>
assetstore.jcloud.credentials = <your-aws-secret-key>

# Define the S3 bucket
assetstore.jcloud.container = <your-bucket-name>
```

Third Party S3 Configuration

With the aws-s3 provider in jclouds, you typically don't need to specify an endpoint as jclouds uses the default AWS S3 endpoint. However, with thrid party vendors, it is more than likely that you will need to set one. MinIO is an example of an S3-compatible object storage that can be self-hosted. Below is an example of how to storage with MinIO as an example.

```
# Set the provider to s3:
assetstore.jcloud.provider = s3

# Configure Credentials
assetstore.jcloud.identity = <your-minio-access-key>
assetstore.jcloud.credentials = <your-minio-secret-key>

# Define the S3 bucket
assetstore.jcloud.container = <your-bucket-name>

# Specify the an endpoint
assetstore.jcloud.endpoint = https://minio.yourdomain.com

# Ensure that MinIO is running and accessible, and that the bucket has been created with the necessary
permissions.
```

Migrate BitStores

There is a command line migration tool to move all the assets within a bitstore, to another bitstore. `bin/dspace bitstore-migrate`

```
[dspace]/bin/dspace bitstore-migrate
usage: BitstoreMigrate
 -a,--source <arg>        Source assetstore store_number (to lose content). This is a number such as 0 or 1
 -b,--destination <arg>   Destination assetstore store_number (to gain content). This is a number such as 0 or
1.
 -d,--delete              Delete file from losing assetstore. (Default: Keep bitstream in old assetstore)
 -h,--help                Help
 -p,--print               Print out current assetstore information
 -s,--size <arg>          Batch commit size. (Default: 1, commit after each file transfer)

[dspace]/bin/dspace bitstore-migrate -p
store[0] == DSBitStore, which has 2 bitstreams.
store[1] == S3BitStore, which has 2 bitstreams.
Incoming assetstore is store[1]

[dspace]/bin/dspace bitstore-migrate -a 0 -b 1


[dspace]/bin/dspace bitstore-migrate -p
store[0] == DSBitStore, which has 0 bitstreams.
store[1] == S3BitStore, which has 4 bitstreams.
Incoming assetstore is store[1]
```

Configuration Reference

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

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

- General Configuration - addresses general conventions used with configuring the `local.cfg` file, `dspace.cfg` and other configuration files which use similar conventions.
- The local.cfg Configuration Properties File - describes how to use the `local.cfg` file to store all your locally customized configurations
- The dspace.cfg Configuration Properties File - specifies the basic `dspace.cfg` file settings (these settings specify the default configuration for DSpace)
- Optional or Advanced Configuration Settings - contain other more advanced settings that are optional in the `dspace.cfg` configuration file.

The full table of contents follows:

1 General Configuration
- 1.1 Configuration File Syntax
  - 1.1.1 Special Characters
  - 1.1.2 Specifying Multiple Values for Properties
  - 1.1.3 Including other Property Files
- 1.2 Configuration Scheme for Reloading and Overriding
- 1.3 Why are there multiple copies of some config files?

2 The local.cfg Configuration Properties File

3 The dspace.cfg Configuration Properties File
- 3.1 Main DSpace Configurations
- 3.2 General Solr Configuration
- 3.3 DSpace Database Configuration
  - 3.3.1 To provide the database connection pool externally
- 3.4 DSpace Email Settings
  - 3.4.1 Wording of E-mail Messages
    - 3.4.1.1 Templates can set message headers
      - 3.4.1.1.1 Pre-requesites
      - 3.4.1.1.2 Enable headers containing hyphens
      - 3.4.1.1.3 Setting the header
      - 3.4.1.1.4 Headers subject and charset are special
- 3.5 File Storage
- 3.6 Logging Configuration
- 3.7 General Plugin Configuration
- 3.8 Configuring the Search Engine
- 3.9 Handle Server Configuration
- 3.10 Delegation Administration: Authorization System Configuration
- 3.11 Inheritance of collection default policy (since 7.1)
- 3.12 Login as feature
- 3.13 Restricted Item Visibility Settings
- 3.14 Restricted Bundle Settings
- 3.15 Proxy Settings
- 3.16 Configuring Media Filters
- 3.17 Crosswalk and Packager Plugin Settings
  - 3.17.1 Configurable MODS Dissemination Crosswalk
  - 3.17.2 XSLT-based Crosswalks
    - 3.17.2.1 Testing XSLT Crosswalks
  - 3.17.3 Configurable Qualified Dublin Core (QDC) dissemination crosswalk
  - 3.17.4 Configuring Crosswalk Plugins
  - 3.17.5 Configuring Packager Plugins
- 3.18 Event System Configuration
- 3.19 Embargo
- 3.20 Checksum Checker Settings
- 3.21 Item Export and Download Settings
- 3.22 Subscription Emails
- 3.23 Hiding Metadata
- 3.24 Hiding Submitter in Provenance Metadata
- 3.25 Settings for the Submission Process
- 3.26 Configuring the Sherpa/RoMEO Integration
- 3.27 Configuring Creative Commons License
- 3.28 WEB User Interface Configurations
- 3.29 Item Counts in user interface
- 3.30 Browse Index Configuration
  - 3.30.1 Defining the storage of the Browse Data
  - 3.30.2 Defining the Indexes
  - 3.30.3 Defining Sort Options
  - 3.30.4 Hierarchical Browse Indexes
  - 3.30.5 Other Browse Options
  - 3.30.6 Browse Index Authority Control Configuration
  - 3.30.7 Tag cloud
- 3.31 Links to Other Browse Contexts
- 3.32 Submission License Substitution Variables

3.33 Syndication Feed (RSS) Settings
3.34 OpenSearch Support
3.35 Content Inline Disposition Threshold / Format
3.36 Multi-file HTML Document/Site Settings
3.37 Sitemap Settings
3.38 Authority Control Settings
3.39 Configuring Multilingual Support
3.39.1 Setting the Default Language for the Application
3.39.2 Supporting More Than One Language
3.39.2.1 Changes in dspace.cfg
3.39.2.2 Related Files
3.40 Upload File Settings
3.41 SFX Server (OpenURL)
3.42 Controlled Vocabulary Settings
3.43 Request-a-copy Settings
4 Optional or Advanced Configuration Settings
4.1 The Metadata Format and Bitstream Format Registries
4.1.1 Metadata Format Registries
4.1.2 Bitstream Format Registry
4.2 Configuring Usage Instrumentation Plugins
4.2.1 The Passive Plugin
4.2.2 The Tab File Logger Plugin
4.3 Behavior of the workflow system
4.4 Recognizing Web Spiders (Bots, Crawlers, etc.)
5 Command-line Access to Configuration Properties
5.1 Captcha Verification settings

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.

DSpace provides a number of textual configuration files which may be used to configure your site based on local needs. These include:

- `[dspace]/config/dspace.cfg` : The primary configuration file, which contains the main configurations for DSpace.
- `[dspace]/config/modules/*.cfg` : Module configuration files, which are specific to various modules/features within DSpace.
- `[dspace]/config/local.cfg` : A (optional, but highly recommended) localized copy of configurations/settings specific to your DSpace (see The local.cfg Configuration Properties File below)
- Additional feature-specific configuration files also exist under `[dspace]/config/`, some of these include:
  - `default.license` : the default deposit license used by DSpace during the submission process (see Submission User Interface documentation)
  - `hibernate.cfg.xml` : The Hibernate class configuration for the DSpace database (almost never requires changing)
  - `item-submission.xml` : the default item submission process for DSpace (see Submission User Interface documentation)
  - `launcher.xml` : The configuration of the DSpace command-line "launcher" (`[dspace]/bin/dspace` , see the DSpace Command Launcher documentation)
  - `log4j2.xml` : The default logging settings for DSpace log files (usually placed in `[dspace]/log`)
  - `submission-forms.xml`: The default deposit forms for DSpace, used by `item-submission.xml` (see Submission User Interface documentation)

As most of these configurations are detailed in other areas of the DSpace documentation (see links above), this section concentrates primarily on the "*.cfg" configuration files (namely `dspace.cfg` and `local.cfg`).

Configuration File Syntax

We will use the `dspace.cfg` as our example for input conventions used throughout the system. These same input conventions apply to all DSpace *.cfg files.

All DSpace *.cfg files use the Apache Commons Configuration properties file syntax. This syntax is very similar to a standard Java properties file, with a few notable enhancements described below.

- Comments all start with a "#" symbol. These lines are ignored by DSpace.
- Other settings appear as property/value pairs of the form: `property.name = property value`
- Certain special characters (namely commas) MUST BE escaped. See the "Special Characters" section below
- Values assigned in the same *.cfg file are "additive", and result in an array of values. See "Specifying Multiple Values for Properties" below.

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.

The property value may contain references to other configuration properties, in the form `${property.name}`. A property may not refer to itself. Examples:

```
dspace.dir = /path/to/dspace
dspace.name = My DSpace

# property.name will be equal to "My DSpace is great!"
property.name = ${dspace.name} is great!

# property2.name will be equal to "/path/to/dspace/rest/of/path"
property2.name = ${dspace.dir}/rest/of/path

# However, this will result in an ERROR, as the property cannot reference itself
property3.name = ${property3.name}
```

Special Characters

Certain characters in `*.cfg` files are considered special characters, and **must** be escaped in any values. The most notable of these special characters include:

- Commas (,) : as they represent lists or arrays of values (see "Specifying Multiple Values for Properties" below)
- Backslashes (\) : as this is the escape character

This means that if a particular setting needs to use one of these special characters in its value, it must be escaped. Here's a few examples:

```
# WRONG SETTING
# This setting is INVALID. DSpace is expecting your site name to be a single value,
# But, this setting would create an array of two values: "DSpace" and "My Institution"
dspace.name = DSpace, My Institution

# CORRECT SETTING (commas is escaped)
# Instead, if the name of your DSpace includes a comma, you need to escape it with "\,"
dspace.name = DSpace\, My Institution

# WRONG SETTING
# As the backslash is the escape character, this won't work
property.name = \some\path

# CORRECT SETTING
# If you want a literal backslash, you need to escape it with "\\"
# So, the below value will be returned as "\some\path"
property.name = \\some\\path
```

Additional examples of escaping special characters are provided in the documentation of the Apache Commons Configuration properties file syntax.

Specifying Multiple Values for Properties

Because DSpace supports the Apache Commons Configuration properties file syntax, it is much easier to specify multiple values for a single setting. All you have to do is repeat the same property name multiple times in the same `*.cfg` file.

For example:

```
# The below settings define *two* AuthenticationMethods that will be enabled, LDAP and Password authentication
# Notice how the same property name is simply repeated, and passed different values.
plugin.sequence.org.dspace.authenticate.AuthenticationMethod = org.dspace.authenticate.LDAPAuthentication
plugin.sequence.org.dspace.authenticate.AuthenticationMethod = org.dspace.authenticate.PasswordAuthentication

# Alternatively, you can also define them as a comma-separated list
# (In this scenario, you would NOT escape the comma, as you want them to be considered multiple values)
# So, this single line is exactly equavalent to the settings above:
plugin.sequence.org.dspace.authenticate.AuthenticationMethod = org.dspace.authenticate.LDAPAuthentication, org.
dspace.authenticate.PasswordAuthentication
```

Please be aware that this ONLY works if you are reusing the exact same configuration in the same configuration file. This causes the values to be "additive" (i.e they are appended to the same list).

However, as you'll see below, the `local.cfg` file always *overrides* settings elsewhere. So, if the above "AuthenticationMethod" plugin was specified in both your `authentication.cfg` and your `local.cfg`, the value(s) in your `local.cfg` would *override* the defaults in your `authentication.cfg` (more on that below).

Additional examples of creating lists or arrays of values are provided in the documentation of the Apache Commons Configuration properties file syntax.

Including other Property Files

Because DSpace supports the Apache Commons Configuration properties file syntax, it also can include/embed property files within other property files by using the "`include=`" setting.

For example, the `dspace.cfg` includes/embeds all of the default `config/modules/*.cfg` files via a series of "`include=`" settings near the bottom of the `dspace.cfg`. As an example, here's a small subset of those include calls:

```
# defines our modules subdirectory
module_dir = modules

# The following lines include specific "authentication*.cfg" files inside your dspace.cfg
# This essentially "embeds" their configurations into your dspace.cfg,
# treating them as if they were a single configuration file.
include = ${module_dir}/authentication.cfg
include = ${module_dir}/authentication-ip.cfg
include = ${module_dir}/authentication-ldap.cfg
include = ${module_dir}/authentication-password.cfg
include = ${module_dir}/authentication-shibboleth.cfg
```

This ability to include properties files within others is very powerful, as it allows you to inherit settings from other files, or subdivide large configuration files. Be aware that this essentially causes DSpace to treat all included configurations *as if they were part of the parent file*. This means that, in the above example, as far as DSpace is concerned, all the settings contained within the `authentication*.cfg` files "appear" as though they are specified in the main `dspace.cfg`.

This ability to include other files is also possible with the `local.cfg` file, should you want to subdivide your localized settings into several locally specific configuration files.

Configuration Scheme for Reloading and Overriding

Known limitations to reloadable configurations

While the DSpace API supports dynamically reloading configurations, the user or machine interfaces *may* still cache some configuration settings. This means that while the API layer may reload a new value, that new value may not always affect/change the behavior of your user interface (until you restart Tomcat).

Also, please be aware that all DSpace configuration values loaded into Spring beans (for example configurations that appear in Spring XML configuration files or in @Value annotations) **are cached by Spring**. This means that they will not be reloadable within Spring beans until Tomcat is restarted.

Because DSpace supports the Apache Commons Configuration, its configurations can now be reloaded without restarting your servlet container (e.g. Tomcat). By default, DSpace checks for changes to any of its runtime configuration files every 5 seconds. If a change has been made, the configuration file is reloaded. The 5 second interval is configurable in the `config-definition.xml` (which defines the configuration scheme DSpace uses).

Additionally, DSpace provides the ability to easily override default configuration settings (in dspace.cfg or modules/*.cfg) using a local.cfg file (see The local.cfg Configuration Properties File) or using System Properties / Environment Varilables.

Both of these features are defined in DSpace's default "configuration scheme" or "configuration definition" in the `[dspace]/config/config-definition.xml` file. This file defines the Apache Commons Configuration settings that DSpace utilizes by default. It is a valid "configuration definition" file as defined by Apache Commons Configuration. See their Configuration Definition File Documentation for more details.

You are welcome to customize the `config-definition.xml` to customize your local configuration scheme as you see fit. Any customizations to this file will require restarting your servlet container (e.g. Tomcat).

By default, the DSpace `config-definition.xml` file defines the following configuration scheme:

- *Configuration File Syntax/Sources:* All DSpace configurations are loaded via Properties files (using the Configuration File Syntax detailed above)
  - Note: Apache Commons Configuration does support other configuration sources such as XML configurations or database configurations (see its Overview documentation). At this time, DSpace does not utilize these other sorts of configurations by default. However, it would be possible to customize your local config-definition.xml to load settings from other locations.
- *Configuration Files/Sources:* By default, only two configuration files are loaded into Apache Commons Configuration for DSpace:
  - `local.cfg` (see The local.cfg Configuration Properties File documentation below)
  - `dspace.cfg` (NOTE: all `modules/*.cfg` are loaded by `dspace.cfg` via "`include=`" statements at the end of that configuration file. They are essentially treated as sub-configs which are embedded/included into the `dspace.cfg`)
- *Configuration Override Scheme:* The configuration override scheme is defined as follows. Configurations specified in earlier locations will automatically override any later values:
  - System Properties (`-D[setting]=[value]`) override all other options
  - Environment Variables.
    - DSpace provides a custom environment variable syntax as follows:
      - All periods (.) in configuration names must be translated to "__P__" (two underscores, capital P, two underscores), e.g. "dspace__P__dir" environment variable will override the "dspace.dir" configuration in local.cfg (or other *.cfg files)
      - All dashes (-) in configuration names must be translated to "__D__" (two underscores, capital D, two underscores), e.g. "authentication__D__ip__P__groupname" environment variable will override the "authentication-ip.groupname" configuration in local.cfg (or other *.cfg files)

    - `local.cfg`
    - `dspace.cfg` (and all `modules/*.cfg` files) contain the default values for all settings.
- *Configuration Auto-Reload:* By default, all configuration files are automatically checked every 5 seconds for changes. If they have changed, they are automatically reloaded.

For more information on customizing our default config-definition.xml file, see the Apache Commons Configuration documentation on the configuration definition file. Internally, DSpace simply uses the `DefaultConfigurationBuilder` class provided by Apache Commons Configuration to initialize our configuration scheme (and load all configuration files).

Customizing the default configuration scheme

Because the `config-definition.xml` file is just a Configuration Definition file for Apache Commons Configuration, you can also choose to customize the above configuration scheme based on your institution's local needs. This includes, but is not limited to, changing the name of "local.cfg", adding additional configuration files/sources, or modifying the override or auto-reload schemes. For more information, see the Configuration Definition File Documentation from Apache Commons Configuration.

Why are there multiple copies of some config files?

It is important to remember that there are **multiple copies of each configuration files in an installation of DSpace**. The primary ones to be aware of are:

1. The "source" configuration file(s) are found under in `[dspace-source]/dspace/config/` or subdirectories. This also includes the `[dspace-source]/local.cfg`
2. The "runtime" configuration file(s) that are found in `[dspace]/config/`

The DSpace server (webapp) and command line programs only look at the *runtime* configuration file(s).

When you are revising/changing your configuration values, it may be tempting to *only edit the runtime file*. **DO NOT** do this. Whenever you rebuild DSpace, it will "reset" your runtime configuration to whatever is in your source directories (the previous runtime configuration is copied to a date suffixed file, should you ever need to restore it).

Instead, we recommend to **always make the same changes to the source version** of the configuration file in addition to the runtime file. In other words, the source and runtime files should always be identical / kept in sync.

One way to keep the two files in synchronization is to edit your files in `[dspace-source]/dspace/config/` and then run the following commands to rebuild DSpace and install the updated configs:

```
cd [dspace-source]/dspace/
mvn package
cd [dspace-source]/dspace/target/dspace-installer
ant update_configs
```

This will copy the source configuration files into the runtime ([dspace]/config) directory. Another option to manually sync the files by copying them to each directory.

Please note that there are additional "ant" commands to help with configuration management:

- "ant update_configs" ==> Moves existing configs in `[dspace]/config/` to *.old files and replaces them with what is in `[dspace-source]/dspace/config/`
- "ant -Doverwrite=false update_configs" ==> Leaves existing configs in `[dspace]/config/` intact. Just copies new configs from `[dspace-source]/dspace/config/` over to *.new files.

The `local.cfg` Configuration Properties File

build.properties has been replaced by local.cfg

As of DSpace 6 and above, the old "`build.properties`" configuration file has been replaced by this new "`local.cfg`" configuration file. For individuals who are familiar with the old `build.properties` file, this new `local.cfg` differs in a few key ways:

- Unlike `build.properties`, the `local.cfg` file can be used to override ANY setting in any other configuration file (`dspace.cfg` or `modules/*.cfg`). To override a default setting, simply copy the configuration into your `local.cfg` and change its value(s).
- Unlike `build.properties`, the `local.cfg` file is not utilized during the compilation process (e.g. mvn package). But, it is automatically copied alongside the final `dspace.cfg` into your installation location (`[dspace]/config/`), where it overrides default DSpace settings with your locally specific settings *at runtime*.
- Like `build.properties`, the `local.cfg` file is expected to be specified in the source directory by default (`[dspace-source]`). There is an example (`[dspace-source]/dspace/config/local.cfg.EXAMPLE`) provided which you can use to create a `[dspace-source]/dspace/config/local.cfg`.

Many configurations have changed names between DSpace 5 (and below) and DSpace 6 (and above)

If you are upgrading from an earlier version of DSpace, you will need to be aware that *many* configuration names/keys have changed. Because Apache Commons Configuration allows for auto-overriding of configurations, all configuration names/keys in different `*.cfg` files MUST be uniquely named (otherwise accidental, unintended overriding may occur).

In order to create this powerful ability to override configurations in your `local.cfg`, all `modules/*.cfg` files had their configurations **renamed** to be prepended with the module name. As a basic example, all the configuration settings within the `modules/oai.cfg` configuration now start with "`oai.`".

Additionally, while the `local.cfg` may look *similar* to the old `build.properties`, many of its configurations have slightly different names. *So, simply copying your build.properties into a local.cfg will NOT work*.

This means that DSpace 5.x (or below) configurations are NOT guaranteed compatible with DSpace 6. While you obviously can use your old configurations as a reference, you will need to start with fresh copy of all configuration files, and reapply any necessary configuration changes (this has always been the recommended procedure). However, as you'll see below, you'll likely want to do that anyways in order to take full advantage of the new `local.cfg` file.

It is possible to easily override default DSpace configurations (from `dspace.cfg` or `modules/*.cfg` files) in your own `local.cfg` configuration file.

A example `[dspace-source]/dspace/config/local.cfg.EXAMPLE` is provided with DSpace. The example only provides a few key configurations which most DSpace sites are likely to need to customize. However, you may add (or remove) any other configuration to your `local.cfg` to customize it as you see fit.

To get started, simply create your own `[dspace-source]/dspace/config/local.cfg` based on the example, e.g.

```
cd [dspace-source]/dspace/config/
cp local.cfg.EXAMPLE local.cfg
```

You can then begin to edit your `local.cfg` with your local settings for DSpace. There are a few key things to note about the `local.cfg` file:

- **Override any default configurations:** Any setting in your `local.cfg` will automatically OVERRIDE a setting of the same name in the `dspace.cfg` or any `modules/*.cfg` file. This also means that you can copy ANY configuration (from `dspace.cfg` or any `modules/*.cfg` file) into your `local.cfg` to specify a new value.
  - For example, specifying `dspace.url` in `local.cfg` will override the default value of `dspace.url` in `dspace.cfg`.
  - Also, specifying `oai.solr.url` in `local.cfg` will override the default value of `oai.solr.url` in `config/modules/oai.cfg`
- **Configuration Syntax:** The `local.cfg` file uses the Apache Commons Configuration Property file syntax (like all *.cfg files) . For more information see the section on Configuration File Syntax above.
  - This means the `local.cfg` also supports enhanced features like the ability to include other config files (via "`include=`" statements).
- **Override local.cfg via System Properties:** As needed, you also are able to OVERRIDE settings in your `local.cfg` by specifying them as System Properties or Environment Variables.
  - For example, if you wanted to change your `dspace.dir` in development/staging environment, you could specify it as a System Property (e.g. `-Ddspace.dir=[new-location]`). This new value will override any value in *both* `local.cfg` and `dspace.cfg`.

When you build DSpace (e.g. mvn package), this `local.cfg` file will be automatically copied to `[dspace]/config/local.cfg`. Similar to the `dspace.cfg`, the "runtime" configuration (used by DSpace) is the one in `[dspace]/config/local.cfg`. See the Why are there multiple copies of some config files? question above for more details on the runtime vs source configuration.

Here's a very basic example of settings you could place into your `local.cfg` file (with inline comments):

```
# This is a simple example local.cfg file which shows off options
# for creating your own local.cfg

# This overrides the default value of "dspace.dir" in dspace.cfg
dspace.dir = C:/dspace/

# This overrides the default value of "dspace.server.url" in dspace.cfg
dspace.server.url = https://dspace.myuniversity.edu/server

# The overrides the default "dspace.ui.url" setting it to the same value as my "baseUrl" above
dspace.ui.url = https://dspace.myuniversity.edu

# If our database settings are the same as the default ones in dspace.cfg,
# then, we may be able to simply customize the db.username and db.password
db.username = myuser
db.password = mypassword

# For DSpace, we want the LDAP and Password authentication plugins enabled
# This overrides the default AuthenticationMethod in /config/modules/authentication.cfg
# Since we specified the same key twice, these two values are appended (see Configuration File Syntax above)
plugin.sequence.org.dspace.authenticate.AuthenticationMethod = org.dspace.authenticate.LDAPAuthentication
plugin.sequence.org.dspace.authenticate.AuthenticationMethod = org.dspace.authenticate.PasswordAuthentication

# We also can reference other configurations in values.
# For instance, we can set the "mail.admin" and the "feedback.recipient" to be the same email
mail.admin = myemail@myuniversity.edu
feedback.recipient = ${mail.admin}

# For the example, we'll override the default oai.path in /config/modules/oai.cfg
# This puts our OAI-PMH interface at ${dspace.server.url}/oaipmh
oai.path = oaipmh

# We'll also override the default oai.solr.url in /config/modules/oai.cfg
# Notice here we're referencing a configuration (solr.server) that only exists in dspace.cfg
# This is allowed. Your local.cfg can reference configs from other *.cfg files.
oai.solr.url = ${solr.server}/oaipmh

# Finally, this local.cfg also supports adding "include=" statements, to include
# additional local configuration files.
# In this example, a local-rest.cfg and local-curate.cfg (in the same directory)
# will automatically be included as part of this local.cfg.
# This allows you to subdivide you local configs in as many (or few) config files
# as you desire.
include = local-rest.cfg
include = local-curate.cfg
```

The `dspace.cfg` Configuration Properties File

Any dspace.cfg setting can be overridden in your local.cfg

Remember, *any* of the below `dspace.cfg` settings can be copied into your local.cfg configuration file and overridden. So, rather than editing the `dspace.cfg` (or any of the `modules/*.cfg`), it's recommended to simply override the default values in your `local.cfg`. That way, your `local.cfg` can serve as the record of which configurations you have actually tweaked in your DSpace, which may help to simplify future upgrades.

The `dspace.cfg` contains basic information about a DSpace installation, including system path information, network host information, and other like items. It is the default configuration file for DSpace, used by DSpace when it is actively running. However, as noted above, any of these default configurations may be overridden in your own `local.cfg` configuration file.

Main DSpace Configurations

Property:	`dspace.dir`

Example Value:	`/dspace`
Informational Note:	Root directory of DSpace installation. Omit the trailing slash '/'. Note that this setting is used by default in other settings, e.g. `assetstore.dir` . (*On Windows be sure to use forward slashes for the directory path!* For example: "C:/dspace" is a valid path for Window.)
Property:	`dspace.server.url`
Example Value:	`https://dspace.myu.edu/server`
Informational Note:	Main URL at which DSpace backend ("server" webapp) is publicly available. If using port 80 (HTTP) or 443 (HTTPS), you may omit the port number. Otherwise the port number must be included. **Do not include a trailing slash ('/').** In Production, you must use HTTPS if you wish to access the REST API from a different server/domain. This configuration should match the User Interface's "rest" settings in the config.yml (specifically the "ssl", "host", "port" and "nameSpace" settings in that file). See User Interface Configuration
Property:	`dspace.server.ssr.url`

Example Value:	`http://localhost:8080/server`
Informational Note:	*(8.1 and later)* Optional, separate Server-Side Rendering URL for the REST API. When specified, this URL will be used by the DSpace User Interface during server-side execution. This may be a private or localhost URL, but it must be accessible to the User Interface's server-side code. For example, the "dspace.server.url" (above) should always reference a public URL like "https://mydspace.edu/server". But, you could set "dspace.server.ssr.url" to a localhost URL (e.g. "http://localhost:8080/server") if your REST API is running on the same machine as the User Interface. This configuration would result in all client-side code (running in the user's browser) accessing the REST API via the "dspace.server.url" (a public, HTTPS URL), while the server-side code (triggered by SSR) would access the REST API via "dspace.server.ssr.url" (a localhost HTTP URL). When this configuration is specified you **must** set the same URL in the User Interface's "rest.ssrBaseUrl" settings in its config.yml. See User Interface Configuration
Property:	`dspace.ui.url`
Example Value:	`dspace.ui.url = http://dspacetest.myu.edu:4000`
Informational note	Main URL at which the DSpace frontend (Angular User Interface) is publicly available. If using port 80 (HTTP) or 443 (HTTPS), you may omit the port number. Otherwise the port number must be included. **Do not include a trailing slash ('/').** In Production, you should be using HTTPS for security purposes. This URL should match the URL you type in the browser to access your User Interface. In the backend, this URL is primarily used to build UI-based URLs in sitemaps, email messages, etc. Therefore, it need not be set on initial installation, but it should be configured as soon as your user interface is installed. If you are not using the DSpace UI (and running the backend "headless"), this may be set to the URL of whatever you consider your primary "user interface".
Property:	`dspace.name`

Example Value:	`dspace.name = DSpace at My University`
Informational Note:	Short and sweet site name, used in e-mails, exports and machine interfaces (e.g. OAI-PMH). It is not currently used by the Angular UI.

General Solr Configuration

DSpace uses Solr for various indexing purposes, and uses a pool of open connections to manage communication with Solr. These properties configure the connections between DSpace and Solr.

See also the additional Solr configuration properties for specific indexes such as search, statistics, authority and OAI PMH.

Property:	`solr.server`
Example Value:	`solr.server = http://localhost:8983/solr`
Informational Note:	Base URL to the Solr server. Specific indexes append to this value.
Property:	`solr.client.maxTotalConnections`
Example Value:	`solr.client.maxTotalConnections = 20`
Informational Note:	The maximum number of connections that will be opened between DSpace and Solr.
Property:	`solr.client.maxPerRoute`
Example Value:	`solr.client.maxPerRoute = 15`
Informational Note:	The maximum number of connections that will be opened between DSpace and a specific Solr instance (if you have more than one).
Property:	`solr.client.keepAlive`
Example Value:	`solr.client.keepAlive = 5000`
Informational Note:	The default amount of time that a connection in use will be held open, in milliseconds. Solr may specify a different keep-alive interval and it will be obeyed.
Property:	`solr.client.timeToLive`
Example Value:	`solr.client.timeToLive = 600`
Informational Note:	The maximum amount of time before an open connection will be closed when idle, in seconds. New connections will be opened as needed, subject to the above limits.

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 only supports PostgreSQL.

Property:	`db.url`
Example Value:	`db.url = jdbc:postgresql://localhost:5432/dspace`
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:	`db.password`
Example Value:	`db.password = dspacepassword`
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 = public`
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. For PostgreSQL databases, this is often best set to "public" (default schema). For Oracle databases, the schema is usually equivalent to the username of your database account. So, for Oracle, this may be set to `${db.username}` in most scenarios.
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.cleanDisabled`
Example Value:	`db.cleanDisabled = true`
Informational Note:	This is a developer-based setting which determines whether you are allowed to run "./dspace database clean" to completely delete all content and tables in your database. This should always be set to "true" in Production to protect against accidentally deleting all your content by running that command. (Default is set to true)

To provide the database connection pool externally

Alternately, you may supply a configured connection pool out of JNDI. The object must be named `jdbc/dspace` (the full path is `java:comp/env/jdbc/dspace`). DSpace will always look up this name and, if found, will use the returned object as its database connection pool. If not found, the above `db.*` properties will be used to create the pool.

If you are using Tomcat, then the object might be defined using a `<Resource>` element, or connected to a `<Resource>` child of `<GlobalNamingResources>` using a `<ResourceLink>` element. See your Servlet container's documentation for details of configuring the JNDI initial context. For example, Tomcat provides a useful JNDI Datasource How-to

Earlier releases of DSpace provided a configuration property `db.jndi` to specify the name to be looked up, but that has been removed. The name is specified in `config/spring/api/core-hibernate.xml` if you really need to change it.

DSpace Email Settings

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

DSpace will look up a javax.mail.Session object in JNDI and, if found, will use that to send email. Otherwise it will create a Session using some of the properties detailed below.

Property:	`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 sent to the email address of choice. This configuration is currently limited to only one recipient. This is also the email address displayed on the contacts page.
Property:	`mail.admin`
Example Value:	`mail.admin = dspace-help@myu.edu`
Example Value:	Email address of the general site administrator (Webmaster). System notifications/reports and other sysadmin emails are sent to this email address.
Property:	`mail.admin.name`
Example Value:	`mail.admin.name = DSpace Administrator`
Example Value:	Name associated with the `mail.admin` email address.
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 and defaults to the `${mail.admin}` setting
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 & defaults to no value.
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 '#set($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. This property is optional. **UNSUPPORTED in DSpace 7.0**
Property:	`mail.extraproperties`

Example Value:	`# Example which can fix "Could not convert socket to TLS" errors (i.e. SMTP over TLS)`
	`mail.extraproperties = mail.smtp.socketFactory.port=587, \`
	`mail.smtp.starttls.enable=true, \`
	`mail.smtp.starttls.required=true, \`
	`mail.smtp.ssl.protocols=TLSv1.2`
	
	`# Different example of using SSL for your Mail libary`
	`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:	`mail.session.name`
Example Value:	`mail.session.name = myDSpace`
Informational Note:	Specifies the name of a javax.mail.Session object stored in JNDI under `java:comp/env/mail`. The default value is "Session".
Property:	`default.language`
Example Value:	`default.language = en_US`
Informational Note:	If no other language is explicitly stated in the *submission-forms.xml*, the default language will be attributed to the metadata values. See also Multilingual Support

Property:	`mail.message.headers`

Example Value:	`mail.message.headers = subject`
	
	`mail.message.headers = charset`

Informational Note:	When processing a message template, setting a Velocity variable whose name is one of the values of this configuration property will add or replace a message header of the same name, using the value of the variable as the header's value. See "Templates can set message headers".
Property:	`mail.welcome.enabled`
Example Value:	`mail.welcome.enabled = true`
Informational Note:	Enable a "welcome letter" to the newly-registered user. By default this is `false`. See the `welcome` email template.

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 e-mails 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. *${params[2]}*) for the template's positional parameters.

Each file is a Velocity template. You can use the full Velocity Template Language to help you customize messages. There are two Velocity variables pre-defined by DSpace when processing an e-mail template:

- `params` is the array of message parameters provided by the DSpace code which is sending the message. These are indexed by number, starting at zero.
- `config` is the table of DSpace configuration properties (such as `dspace.name`). These are looked up using `config.get(property name)`.

Sample message template

```
## This is a comment.  It will not be part of the message.
This is the body of the message.  The code which sends it supplied two parameters:  ${params[0]} and ${params
[1]}.
The name of this DSpace instance is ${config.get('dspace.name')} and you can browse it at ${config.get('dspace.
url')}.
```

Also see the template `config/emails/register` for an example of each.

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`

Configure set of property keys accessible by Velocity Templates

For security reasons, the list of configuration properties available to use in Email or LDN templates should be as restrictive as possible, to avoid any unwanted information disclosure.

By default, an attempt to access a disallowed or missing configuration property will result in a null or blank string. You can also run Velocity in "strict mode" which will throw a fatal exception if a message is constructed with an invalid configuration property.

If you have custom email templates accessing configuration properties outside of these defaults, make sure to configure them here.

```
# Allowed configuration properties, to pass in a "config" map to email and LDN templates.
# This allows templates to easily access dynamic configuration properties, without
# exposing sensitive information to the templating engine
message.templates.allowed-config = dspace.name
message.templates.allowed-config = dspace.shortname
message.templates.allowed-config = dspace.ui.url
message.templates.allowed-config = mail.helpdesk
message.templates.allowed-config = mail.message.helpdesk.telephone
message.templates.allowed-config = mail.admin
message.templates.allowed-config = mail.admin.name

# Whether to run Velocity in strict mode (null parameter values in templates for LDN or Email will result
# in an Exception instead of a blank string)
# Default: false (this can introduce unwanted side-effects if e.g. a submitter eperson is deleted for a
workflow task)
#message.templates.strict_mode = false
```

Templates can set message headers

A template can set arbitrary email message headers by defining Velocity variables.

This can be leveraged to control how emails are processed or displayed, e.g. when using DSpace alongside a ticket queue, issue tracker or email program.

Pre-requesites

Headers need to be enabled before they can be set. To enable a specific header, you need to name it as a value of the DSpace configuration property `mail.message.headers` in your local.cfg or dspace.cfg.

⚠️ Beware: You always need to enable the special headers "subject" and "charset"!

To enable setting the header "Reply-To", put this in you local.cfg:

Enabling email headers in local.cfg

```
# subject and charset must be named explicitely
mail.message.headers = subject
mail.message.headers = charset

# Enable setting Reply-To field from within VTL template for email.
mail.message.headers = Reply-To
```

💡 To check which headers are currently enabled use the `dsprop` command line utility:

Check currently enabled email headers

```
$ dspace dsprop -p mail.message.headers
> subject
> charset
> Reply-To
```

Enable headers containing hyphens

⚠️ If the message header you are setting contains hyphen(s), you need to configure Velocity engine to allow hyphens in variable names. This can be configured in /dspace-api/src/main/java/org/dspace/core/Email.java by adding another call to `VELOCITY_PROPERTIES.put()`:

Allow hyphens in header names in Email.java

```
VELOCITY_PROPERTIES.put("parser.allow_hyphen_in_identifiers", "true");
```

Setting the header

To set an enabled header to a specific value, use Velocity's set-Directive, e.g.:

Setting message header in template

```
#set(Reply-To = "submitter@myu.edu")
#set(My-Header = "Hello World!")
```

⚠ When setting email headers, you do **not** need to prefix the header name with a dollar sign ($), contrary to "ordinary" Velocity variables.

⚠ Some characters have a special meaning in VTL. Using them in a string value may lead to unexpected behavior or break the template, and sending the email may fail completely or partially. In that case, check DSpace's logfile for errors related to `org.apache.velocity.velocity-engine-core`.

Headers subject and charset are special

⚠ Two Velocity variable names are special in DSpace email templates:

name	meaning
subject	supplies the Subject: header's value.
charset	sets the `charset` parameter of the `Content-Type:` header of the bodypart, when there is a single bodypart. It also causes the subject value to be treated as being encoded in this charset. If not set, the charset defaults to US-ASCII as specified in RFC 2046. If there are multiple bodyparts, all are assumed to be encoded in US-ASCII and `charset` has no effect on them.

Sample message template

```
## This is a comment.  It will not be part of the sent message.
#set($subject = "This will be the Subject: of the message")
This is the body of the message.
```

File Storage

Beginning with DSpace 6, your file storage location (aka bitstore) is now defined in the `[dspace]/config/spring/api/bitstore.xml` Spring configuration file. By default it is defined as the `[dspace]/assetstore/`. More information on modifying your file storage location can be found at Configuring the Bitstream Store in the Storage Layer documentation.

DSpace supports multiple options for storing your repository bitstreams (uploaded files). The files are not stored in the database, instead they are provided via a configured "assetstore" or "bitstore".

By default, the assetstore is simply a directory on your server (`[dspace]/assetstore/`) under which bitstreams (files) are stored by DSpace.

At this time, DSpace supports two primary locations for storing your files:

1. Your local filesystem (used by default), specifically under the `[dspace]/assetstore/` directory
2. OR, Amazon S3 (requires your own Amazon S3 account)

More information on configuring or customizing the storage location of your files can be found in the Storage Layer documentation.

Logging Configuration

Logging configuration has now moved to `${dspace.dir}/config/log4j2.xml`

Property:	`log.init.config`
Example Value:	`log.init.config = ${dspace.dir}/config/log4j2.xml`

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/log4j2.xml`
	`log.init.config = ${dspace.dir}/config/log4j2-console.xml`

Property:	`log.dir  (defined in log4j2.xml)`
Example value:	`log.dir = ${log4j:configParentLocation}/../log`
Informational Note:	This is where to put the DSpace logs. The default setting (shown above) writes all DSpace logs to the ${dspace.dir}/log/ directory.
Property:	`loglevel.dspace (defined in log4j2.xml)`
Example value:	`loglevel.dspace = INFO`
Informational Note:	Log level for all DSpace-specific code (org.dspace.* packages). By default, DSpace only provides general INFO logs (in order to keep log sizes reasonable). As necessary, you can temporarily change this setting to any of the following (ordered for most information to least): DEBUG, INFO, WARN, ERROR, FATAL Please be aware we do not recommend running at the DEBUG level in Production for significant periods of time, as it will cause the logs to be extremely large in size.
Property:	`loglevel.other (defined in log4j2.xml)`
Example value:	`loglevel.other = INFO`
Informational Note:	Log level for other third-party tools/APIs used by DSpace (non-DSpace specific code). By default, DSpace only provides general INFO logs (in order to keep log sizes reasonable). As necessary, you can temporarily change this setting to any of the following (ordered for most information to least): DEBUG, INFO, WARN, ERROR, FATAL Please be aware we do not recommend running at the DEBUG level in Production for significant periods of time, as it will cause the logs to be extremely large in size.

General Plugin Configuration

Property:	**`plugin.classpath`**
Example Value:	/opt/dspace/plugins/aPlugin.jar:/opt/dspace/moreplugins
Informational Note:	Search path for third-party plugin classes. This is a colon-separated list of directories and JAR files, each of which will be searched for plugin classes after looking in all the places where DSpace classes are found. In this way you can designate one or more locations for plugin files which will not be affected by DSpace upgrades.

Configuring the Search Engine

DSpace's search module is also known as "Discovery" and utilizes Apache Solr for indexing. It provides up-to-date features, such as filtering/faceting, hit highlighting, search snippets, etc.

Detailed documentation is available for customization, see Discovery.

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 Handle.Net Registry Support.

Property:	`handle.canonical.prefix`

Example Value	`handle.canonical.prefix = http://hdl.handle.net/` `handle.canonical.prefix = ${dspace.ui.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.ui.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.
Property	`handle.additional.prefixes`
Example Value	`handle.additional.prefixes = 1234.56789.0, 1234.56789.1, 987`
Informational Note:	List any additional prefixes that need to be managed by this handle server. For example, any handle prefixes that came from an external repository whose items have now been added to this DSpace. Multiple additional prefixes may be added in a comma separated list.

Delegation Administration: Authorization System Configuration

It is possible to delegate the administration of Communities and Collections. This functionality eliminates the need for an Administrator Superuser account for these purposes. An EPerson that will be attributed Delegate Admin rights for a certain community or collection will also "inherit" the rights for underlying collections and items. As a result, a community admin will also be collection admin for all underlying collections. Likewise, a collection admin will also gain admin rights for all the items owned by the collection.

Authorization to execute the functions that are allowed to user with WRITE permission on an object will be attributed to be the ADMIN of the object (e.g. community/collection/admin will be always allowed to edit metadata of the object). The default will be "*true*" for all the configurations.

Community Administration: Subcommunities and Collections	
Property:	`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.

Collection 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 Collection. The 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`

Inheritance of collection default policy (since 7.1)

Property:	core.authorization.installitem.inheritance-read.append-mode
Example Value:	core.authorization.installitem.inheritance-read.append-mode = false
Informational Note:	Determine if the DEFAULT READ policies of the collection should be always appended to the policies of the new item (property set to **true**) or used only when no other READ policy has been defined in the submission process (property set to **false**). Please note that also in append mode an open access default policy will be NOT inherited if other policies have been defined in the submission (i.e. if the item was restricted)

Login as feature

Property:	`webui.user.assumelogin`
Example Value:	`webui.user.assumelogin = 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.

Restricted Item Visibility Settings

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

Property:	`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.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.

Restricted Bundle Settings

Some bundles are intended only to be accessed and processed internally, such as TEXT, LICENSE, and SWORD. This configuration section allows any bundle to be restricted to only site administrators or the system bypass.

Property:	`core.authorization.restricted-bundle`
Example Value:	`core.authorization.restricted-bundle` = TEXT

Informational Note:	All bundles of the given name will have polciies stripped on item install or creation, or when inheriting from a new collection.

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. *Only currently used for Creative Commons licensing feature (to contact their API), and Sitemap generation (to ping search server regarding updates)*
Property:	`http.proxy.port`
Example Value	`http.proxy.port = 2048`
Informational Note	Enter the port number for the proxy server. *Only currently used for Creative Commons licensing feature (to contact their API), and Sitemap generation (to ping search server regarding updates)*
Property:	http.proxy.hosts-to-ignore
Example Value	http.proxy.hosts-to-ignore = 127.0.0.1, localhost
Informational Note	The configured proxy will be ignored for the configured hosts (the connection id made with no proxy in the middle)
Property	`useProxies`
Example Value:	`useProxies = true`
Informational Note:	As of DSpace 7 (and above), this setting defaults to **true**. If "useProxies" is enabled, the authentication and statistics logging code will read the X-Forwarded-For header in order to determine the correct client IP address. As the User Interface uses Angular Universal (for SEO support), the proxy server that comes with Angular Universal is always enabled. By default, only your local server (127.0.0.1) and the public IP address of `dspace.ui.url` are "trusted" as a proxy. If your DSpace instance is protected by external proxy server, you may need to update the "proxies.trusted.ipranges" property below. This also affects IPAuthentication, and should be enabled for that to work properly if your installation uses a proxy server.
Property	proxies.trusted.ipranges
Example Value:	proxies.trusted.ipranges `= 127.0.0.1`
Informational Note:	By default, only proxies running on localhost (127.0.0.1) and the `dspace.ui.url` (public IP address) are "trusted". This allows our Angular User Interface to communicate with the REST API via a trusted proxy, which is required for Angular Universal (for SEO support). You can specify a range by only listing the first three ip-address blocks, e.g. 128.177.243 You can list multiple IP addresses or ranges by comma-separating them.
Property	proxies.trusted.include_ui_ip

Example Value:	proxies.trusted.include_ui_ip = true
Informational Note:	This setting specifies whether to automatically trust IP address of the `dspace.ui.url` as a proxy. By default, this is always set to true to ensure the UI is fully trusted by the backend. However, if you are not using the Angular UI, you may choose to set this to "false" in order to only trust proxies running on localhost (127.0.0.1) by default.
Property	server.forward-headers-strategy
Example Value:	server.forward-headers-strategy = FRAMEWORK
Informational Note:	This is a Spring Boot setting which may be overridden/specified in your local.cfg. By default, Spring Boot does not automatically use X-Forwarded-* Headers when generating links (and similar) in the REST API. When using a proxy in front of the REST API, you may need to modify this setting: • NATIVE = allows your web server to natively support standard Forwarded headers • FRAMEWORK = enables Spring Framework's built in filter to manage these headers in Spring Boot. (This value may be useful to set for DSpace if you find that X-Forwarded headers are not working) • NONE = default value. Forwarded headers are ignored For more information see the Spring Boot docs at https://docs.spring.io/spring-boot/docs/current/reference/html/howto.html#howto-use-behind-a-proxy-server

Configuring Media Filters

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

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

Property:	`filter.plugins`

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

Informational Note:	This setting lists the names of all enabled MediaFilter or FormatFilter plugins. To enable multiple plugins, list them on separate lines (as shown above) or provide a comma separated list.
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`
	`plugin.named.org.dspace.app.mediafilter.FormatFilter = org.dspace.app.mediafilter.HTMLFilter = HTML Text Extractor`
	`plugin.named.org.dspace.app.mediafilter.FormatFilter = org.dspace.app.mediafilter.WordFilter = Word Text Extractor`
	`plugin.named.org.dspace.app.mediafilter.FormatFilter = org.dspace.app.mediafilter.JPEGFilter = JPEG Thumbnail`
	`plugin.named.org.dspace.app.mediafilter.FormatFilter = org.dspace.app.mediafilter.BrandedPreviewJPEGFilter = Branded Preview JPEG`

Informational Note:	Assign "human-understandable" names to each filter. These names are used to enable/disable plugins using "filter.plugins" setting above. As with the previous setting, multiple plugins can be listed here on separate lines (or comma separated)

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). These must match format names in the DSpace file format registry.
Property:	filter.org.dspace.app.mediafilter.publicPermission
Example Value:	filter.org.dspace.app.mediafilter.publicPermission = JPEGFilter
	Optionally, configure filter(s) which should **always** create publicly accessible bitstreams (e.g. useful if you want thumbnails to always be publicly accessible). By default, any bitstreams created by a filter will inherit the same permissions as the original file (e.g. if original image is access restricted, then thumbnail will also be access restricted by default).
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` .

More Information on MediaFilters

For more information on Media/Format Filters, see the section on Mediafilters for Transforming DSpace Content.

Crosswalk and Packager Plugin Settings

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

More Information on Packagers & Crosswalks

For more information on using Packagers and Crosswalks, see the section on Importing and Exporting Content via Packages.

Configurable MODS Dissemination Crosswalk

The MODS crosswalk is a self-named plugin. To configure an instance of the MODS crosswalk, add a property to the DSpace configuration starting with "c`rosswalk.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.submission.PluginName.stylesheet =
          1        2        3               4
```

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

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

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

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

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

For example:

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

If you remove all XSLTDisseminationCrosswalks please disable the XSLTDisseminationCrosswalk in the list of selfnamed plugins. If no XSLTDisseminationCrosswalks are configured but the plugin is loaded the PluginManager will log an error message ("Self-named plugin class "org.dspace.content.crosswalk.XSLTDisseminationCrosswalk" returned null or empty name list!").

Testing XSLT Crosswalks

The XSLT crosswalks will automatically reload an XSL stylesheet that has been modified, so you can edit and test stylesheets without restarting DSpace. You can test a crosswalk by using a command-line utitlity. To test a dissemination crosswalk you have to run:

```
[dspace]/bin/dspace dsrun org.dspace.content.crosswalk.XSLTDisseminationCrosswalk <plugin name> <handle>
[output-file]
```

For example, you can test the marc plugin on the handle 123456789/3 with:

```
[dspace]/bin/dspace dsrun org.dspace.content.crosswalk.XSLTDisseminationCrosswalk marc 123456789/3
```

Informations from the script will be printed to stderr while the XML output of the dissemination crosswalk will be printed to stdout. You can give a third parameter containing a filename to write the output into a file, but be careful: the file will be overwritten if it exists.

When you are working on XSLTCrosswalks it is very helpful to get the original XML on which the XSLT processor works. Use the crosswalk dim to get the original XML:

```
[dspace]/bin/dspace dsrun org.dspace.content.crosswalk.XSLTDisseminationCrosswalk dim 123456789/3
```

Testing a submission crosswalk works quite the same way. Use the following command-line utility, it calls the crosswalk plugin to translate an XML document you submit, and displays the resulting intermediate XML (DIM). Invoke it with:

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

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

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

Configurable Qualified Dublin Core (QDC) dissemination crosswalk

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

The following is from *dspace.cfg* file:

Properties:	`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.

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 Event Mechanism.

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.discovery.class`
Example Value:	`event.consumer.discovery.class = org.dspace.discovery.IndexEventConsumer`
Informational Note:	Consumer to maintain the search/browse (Discovery) index.
Property:	`event.consumer.discovery.filters`
Example Value:	`event.consumer.discovery.filters = Community | Collection | Item | Bundle | Site+Add | Create | Modify | Modify_Metadata | Delete | Remove`
Informational Note:	Consumer to maintain the search/browse (Discovery) 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.**

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.

More Embargo Details

More details on Embargo configuration, including specific examples can be found in the Embargo section of the documentation.

Checksum Checker Settings

DSpace 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).

More Checksum Checking Details

For more information on using DSpace's built-in Checksum verification system, see the section on Validating CheckSums of Bitstreams.

Item Export and Download Settings

It is possible for an authorized user to request a complete export and download of a DSpace item in a compressed zip file. This zip file may contain the following:
dublin_core.xml
license.txt
contents *(listing of the contents)*
handle *file itself and the extract file if available*

The configuration settings control several aspects of this feature:

Property:	`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.

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.

Hiding Metadata

It is possible to hide metadata from public consumption, so that it's only available to users with WRITE permissions on the Item. (NOTE: Prior to 7.6.1, Administrator privileges were required for hidden metadata. This was modified to allow users to submit hidden metadata fields, as well as allow Community /Collection Admins to see hidden metadata.)

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 has WRITE permissions on the Item: 1. REST API (and therefore the User Interface) 2. RDF (everywhere as there is currently no possibility to authenticate) 3. OAI-PMH server (everywhere as there is currently no possibility to authenticate) 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.

Hiding Submitter in Provenance Metadata

As of DSpace 8, it's possible to configure DSpace to no longer include Submitter details (name and email) in the "dc.description.provenance" field.

Property:	`metadata.privacy.dc.description.provenance`
Example Value:	`metadata.privacy.dc.description.provenance = true`
Informational Note:	If set to "true", all Submitter details (name and email) are excluded from the "dc.description.provenance" metadata field during Submission and Workflow. Default value is "false", which means the Submitter name and email will be included in "dc.description.provenance" metadata fields.

Settings for the Submission Process

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.
Property	submitter.allow.external.entities
Example Value:	submitter.allow.external.entities = false
Informational Note:	If set to "true", all non-admin users will be allowed to import external entities directly into the DSpace repository during the submission process. CAUTION! THIS BYPASSES THE SUBMISSION WORKFLOW OF THE SPECIFIC COLLECTION, BE CAREFUL WHEN ENABLING THIS! The default value is "false", which means only admin users are allowed to import external entities directly into your DSpace repository.

Configuring the Sherpa/RoMEO Integration

DSpace has integration with the Sherpa/RoMEO API in order to allow importing data from Sherpa/RoMEO during the submission You must register for a free API key (see below for details).

Property:	sherpa.romeo.url
Example Value:	sherpa.romeo.url = https://v2.sherpa.ac.uk/cgi/retrieve
Informational Note:	The Sherpa/RoMEO endpoint.
Property:	sherpa.romeo.apikey
Example Value:	`sherpa.romeo.apikey = YOUR-API-KEY`
Informational Note:	Allow to use a specific API key to raise the usage limit (500 calls/day for unregistred user). You MUST register for a free api access key at https://v2.sherpa.ac.uk/api/

The functionality rely on understanding to which Journal (ISSN) is related the submitting item. This is done out of box looking to some item metadata but a different strategy can be used as for example look to a metadata authority in the case that the Sherpa/RoMEO autocomplete for Journal is used (see AuthorityControlSettings)

The strategy used to discover the Journal related to the submission item is defined in the spring file **/config/spring/api/sherpa.xml**

```
<bean class="org.dspace.app.sherpa.submit.SHERPASubmitConfigurationService"
                id="org.dspace.app.sherpa.submit.SHERPASubmitConfigurationService">
                <property name="issnItemExtractors">
                        <list>
                                <bean class="org.dspace.app.sherpa.submit.MetadataValueISSNExtractor">
                                        <property name="metadataList">
                                                <list>
                                                        <value>dc.identifier.issn</value>
                                                </list>
                                        </property>
                                </bean>
                                <!-- Use the follow if you have the SHERPARoMEOJournalTitle enabled
                                <bean class="org.dspace.app.sherpa.submit.MetadataAuthorityISSNExtractor">
                                        <property name="metadataList">
                                                <list>
                                                        <value>dc.title.alternative</value>
                                                </list>
                                        </property>
                                </bean>  -->
                        </list>
                </property>
        </bean>
```

Configuring Creative Commons License

The following configurations are for the Creative Commons license step in the submission process. Submitters are given an opportunity to select a Creative 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 optionally available and may be configured for any given collection that has a defined submission sequence, or be part of the "default" submission process. This process is described in the Submission User Interface section of this manual. There is a Creative Commons step already defined, but it is commented out, so enabling Creative Commons licensing is typically just a matter of uncommenting that step.

When enabled, the Creative Commons public API is utilized. This allows DSpace to store metadata references to the selected CC license, while also storing the CC License as a bitstream. The following CC License information are captured:

- The URL of the CC License is stored in the "dc.rights.uri" metadata field (or whatever field is configured in the "cc.license.uri" setting below)
- The name of the CC License is stored in the "dc.rights" metadata field (or whatever field is configured in the "cc.license.name" setting below). This only occurs if "cc.submit.setname=true" (default value)
- The RDF version of the CC License is stored in a bitstream named "license_rdf" in the CC-LICENSE bundle (as long as "cc.submit.addbitstream=true", which is the default value)

The following configurations (in dspace.cfg) relate to the Creative Commons license process:

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.
Property:	`cc.license.name`
Example Value:	`cc.license.name = dc.rights`
Informational Note:	The field that holds the Creative Commons license Name.
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:	`# Configure specific jurisdiction via language code:`
	`cc.license.jurisdiction = nz`
	`# Set property to 'empty' to overwrite default value from dspace.cfg:`
	`cc.license.jurisdiction =`

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) Commenting out this field will cause DSpace to select the latest, unported CC license (currently version 4.0). However, as Creative Commons 4.0 does not provide jurisdiction specific licenses, if you specify this setting, your DSpace will continue to use older, Creative Commons 3.0 jurisdiction licenses. If commenting out this field in your local.cfg doesn't seem to work, double check your dspace.cfg. If cc.license.jurisdiction is set to a specific value in dspace.cfg, commenting it out in local.cfg will result in your DSpace using the value configured in dspace.cfg. Possible workaround: Set the property to an 'empty value' like in the example above.
Property	`cc.license.locale`
Example Value:	`cc.license.locale` = en
Informational Note:	Locale to be used (in the form: language or language_country), e.g. "en" or "en_US" If no default locale is defined the Creative Commons default locale will be used.

WEB User Interface Configurations

General Web User Interface Configurations

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). **UNSUPPORTED in DSpace 7.0**
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.maxwidth`
Example Value:	`webui.preview.maxwidth = 600`
Informational Note:	This property sets the maximum width for the preview image. Only used for BrandedPreviewJPEGFilter
Property:	`webui.preview.maxheight`
Example Value:	`webui.preview.maxheight = 600`
Informational Note:	This property sets the maximum height for the preview image. Only used for BrandedPreviewJPEGFilter
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. Only used for BrandedPreviewJPEGFilter
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. Only used for BrandedPreviewJPEGFilter
Property:	`webui.preview.brand.height`
Example Value:	`webui.preview.brand.height = 20`

Informational Note:	The height (in px) of the brand. Only used for BrandedPreviewJPEGFilter
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. Only used for BrandedPreviewJPEGFilter
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. Only used for BrandedPreviewJPEGFilter
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. Only used for BrandedPreviewJPEGFilter

Item Counts in user interface

Available in 7.6 or later

Optionally, you can enable item counts to be displayed in the user interface for every Community and Collection. This uses the same configuration that was in place for DSpace 6 and earlier.

Property:	`webui.strengths.show`
Example Value:	`webui.strengths.show = false`
Informational Note:	When "true" this will display the count of archived items (in the User Interface's browse screens). By default this is "false" (disabled). When enabled, the counts may be counted in real time, or fetched from the cache (see next option).
Property:	`webui.strengths.cache`
Example Value:	`webui.strengths.cache = false`
Informational Note:	When showing the strengths (i.e. item counts), should they be counted in real time, or fetched from the cache. Counts fetched in real time will perform an actual count of the index contents every time a page with this feature is requested, which may not scale. If you set the property key is set to cache ("true"), the counts will be cached on first load.

Browse Index Configuration

The browse indexes for DSpace can be extensively configured. These configurations are used by Discovery. 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:item:dateissued` `webui.browse.index.2 = author:metadata:dc.contributor.*\,dc.creator:text`
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 storage of the Browse Data

Optionally, you may configure a custom implementation use for the Browse DAOs both for read operations (create/update operations are handled by Event Consumers). However, as of DSpace 6, DSpace only includes one out-of-the-box option:

- SOLR Browse Engine (SOLR DAOs), default since DSpace 4.0 - This enables Apache Solr to be utilized as a backend for all browsing of DSpace. This option requires that you have Discovery (Solr search/browse engine) enabled in your DSpace.

Property:	`browseDAO.class`
Example Value:	browseDAO.class = org.dspace.browse.SolrBrowseDAO
Informational Note:	This property configures the Java class that is used for READ operations by the Browse System. **You need to have Discovery enabled (this is the default since DSpace 4.0) to use the Solr Browse DAOs**

Defining the Indexes

If you make changes in this section be sure to update your SOLR indexes running the Discovery Maintenance Script, see Discovery

DSpace comes with four default indexes pre-defined: author, title, date issued, and subjects. Users may also define additional indexes or re-configure the current indexes for different levels of specificity. For example, the default entries that appear in the dspace.cfg as default installation:

```
webui.browse.index.1 = dateissued:item:dateissued
webui.browse.index.2 = author:metadata:dc.contributor.*\,dc.creator:text
webui.browse.index.3 = title:item:title
webui.browse.index.4 = subject:metadata:dc.subject.*:text
#webui.browse.index.5 = dateaccessioned:item:dateaccessioned
```

There are two types of indexes which are provided in this default integration:

- "`item`" indexes which have a format of `webui.browse.index.<n> = <index-name> : item : <sort-type> : (asc | desc)`
- "`metadata`" indexes which have a format of `webui.browse.index.<n> = <index-name> : metadata : <comma-separated-list-of-metadata-fields> : (date | text) : (asc | dec) : <sort-type>`

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. In order for the DSpace UI to display human-friendly description for this index, you'll need to update the UI's language packs (e.g. `src/assets/i18n/en.json5`) to include a key using this index name, for example: • `browse.metadata.<index-name> = "MyField",` • `browse.metadata.<index-name>.breadcrumbs = "Browse by MyField",`
`(metadata|item)`	Only two options are available: "`metadata`" or "`item`" • "`metadata`" indexes allow you to index all items based on one or more metadata fields. The list of fields should be provided as part of the "metadata" configuration. Only items which have values for these fields will appear in this index (e.g. if you have a "metadata" index for "`dc.subject.*`", an item will not appear in that browse/search if it doesn't have a "`dc.subject.*`" value). The browse index will have to parts: first it lists all values of the specified metadata fields. If the user select one of these values the index lists all items in which the specified metadata field is assigned with the selected value. ◦ Note: If you set a <sort-type> to be used, this sort type is not used on the values of the metadata fields but on the order of the items when listing all items that have a specific value of the metadata field. • "`item`" indexes provide you with a browseable list of ALL items in the site, sorted by a particular metadata field. The field this index is sorted by is referenced by `<sort-option-name>` (which should refer to a corresponding "`webui.itemlist.sort-option.<n>`" setting... see Defining Sort Options below for more information)
`<schema-prefix>`	(Only for "metadata" indexes) The schema used for the field to be index. First part of a metadata field name. The default is dc (for Dublin Core).
`<element>`	(Only for "metadata" indexes) The schema element. Second part of a metadata field name. 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>`	(Only for "metadata" indexes) This is the qualifier to the <element> component. Third part of a metadata field name. 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.)
`<sort-type>`	(Optional, should be set for "item" indexes) This refers to the sort type / data type of the field: • `date` the index type will be treated as a date object and sorted as such • `text` the index type will be treated as plain text and sorted as such • (any other value refers to a custom <sort-type> which should be defined in a corresponding `webui.itemlist.sort-option.<n>` setting. See Defining Sort Options below for more information.)
`<sort-order>`	(Optional) The default sort order. Choose `asc` (ascending) or `desc` (descending). Ascending is the default value, but descending may be useful for date-based indexes (e.g. to display most recent submissions first)

Defining Sort Options

If you make changes in this section be sure to update your SOLR indexes running the Discovery Maintenance Script, see Discovery

Sort options/types will be available when browsing a list of items (either on "`item`" index type above or after selecting a specific value for "metadata" indexes). You can define an arbitrary number of fields to sort on. 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> = <sort-type-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.itemlist.sort-option.<n>`	*n* is an arbitrary number you choose.
`<sort-type-name>`	The name by which the sort option will be identified. This is the name by which it is referred in the "webui.browse.index" settings (see Defining the Indexes).
`<schema-prefix>`	The schema used for the field to be sorted on in the 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>`	This refers to the datatype of the field: `date` the sort field will be treated as a date object `text` the sort field will be treated as plain text. `title` the sort field will be treated like a title, which will include a link to the item page

Hierarchical Browse Indexes

No configuration is necessary for hierarchical browse indexes (Browse by Subject Category). These are automatically generated based on the used controlled vocabularies in your submission forms. Default DSpace has one hierarchical browse index (Browse by Subject Category), since "srsc" is the only vocabulary used in the default submission-forms.xml.

Please note that when using another vocabulary, the UI's language packs (e.g. src/assets/i18n/en.json5) will need to be updated as well, e.g.:

- `"menu.section.browse_global_by_srsc": "By Subject Category"`
- `"browse.metadata.srsc.breadcrumbs": "Browse by Subject Category"`
- `"browse.comcol.by.srsc": "By Subject Category"`

Starting with DSpace 7.6.1, these Hierarchical "Browse By" options can be **disabled** via the below configuration:

Property:	`webui.browse.vocabularies.disabled`
Example Value:	`webui.browse.vocabularies.disabled = srsc`

Informational Note:	By default, all controlled vocabularies used within your submission forms (submission-forms.xml) will be enabled in the Browse By menu of the User Interface. If you wish to disable any from display in the UI, you can list them in this configuration. Multiple values can be comma separated (or this config can be repeated). Changes to this configuration will not take effect until your servlet engine (e.g. Tomcat) is restarted.

Additional settings are detailed in Authority Control of Metadata Values

Other Browse Options

We set other browse values in the following section.

Property:	`webui.browse.metadata.show-freq.< n >`
Example Value:	`webui.browse.metadata.show-freq.1 = false`
Informational Note:	This enable/disable the show of frequencies (count) in metadata browse <n> refers to the browse configuration. As default frequencies are shown for all metadata browse
Property:	`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:	

Tag cloud

Apart from the single (type=metadata) and full (type=item) browse pages, tag cloud is a new way to display the unique values of a metadata field.

To enable “tag cloud” browsing for a specific index you need to declare it in the dspace.cfg configuration file using the following option:

Property:	webui.browse.index.tagcloud.<*n*>
Example Value:	webui.browse.index.tagcloud.1 = true

Informational Note:	Enable/Disable tag cloud in browsing for a specific index. ‘n’ is the index number of the specific index which needs to be of type ‘metadata’. Possible values: true, false Default value is false. If no option exists for a specific index, it is assumed to be false. You do not have to re-index discovery when you change this configuration **UNSUPPORTED in DSpace 7.0**

Tag cloud configuration

The appearance configuration for the tag cloud is located in the Discovery xml configuration file (*dspace/config/spring/api/discovery.xml*). Without configuring the appearance, the default one will be applied to the tag cloud

In this file, there must be a bean named “*browseTagCloudConfiguration*” of class “*org.dspace.discovery.configuration.TagCloudConfiguration*”. This bean can have any of the following properties. If some is missing, the default value will be applied.

displayScore	Should display the score of each tag next to it? Default: false
shouldCenter	Should display the tag as center aligned in the page or left aligned? Possible values: true | false. Default: true
totalTags	How many tags will be shown. Value -1 means all of them. Default: -1
cloudCase	The letter case of the tags. Possible values: Case.LOWER | Case.UPPER | Case.CAPITALIZATION | Case.PRESERVE_CASE | Case.CASE_SENSITIVE Default: Case.PRESERVE_CASE
randomColors	If the 3 css classes of the tag cloud should be independent of score (random=yes) or based on the score. Possible values: true | false . Default: true
fontFrom	The font size (in em) for the tag with the lowest score. Possible values: any decimal. Default: 1.1
fontTo	The font size (in em) for the tag with the lowest score. Possible values: any decimal. Default: 3.2
cuttingLevel	The score that tags with lower than that will not appear in the rag cloud. Possible values: any integer from 1 to infinity. Default: 0
ordering	The ordering of the tags (based either on the name or the score of the tag) Possible values: Tag.NameComparatorAsc | Tag.NameComparatorDesc | Tag.ScoreComparatorAsc | Tag.ScoreComparatorDesc Default: Tag.GreekNameComparatorAsc

When tagCloud is rendered there are some CSS classes that you can change in order to change the tagcloud appearance.

Class	Note
tagcloud	General class for the whole tagcloud
tagcloud_1	Specific tag class for tag of type 1 (based on score)
tagcloud_2	Specific tag class for tag of type 2 (based on score)
tagcloud_3	Specific tag class for tag of type 3 (based on score)

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.

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.

Syndication Feed (RSS) Settings

Supported as of 7.3 and above.

Please note that Syndication (RSS/Atom) feeds **require** that OpenSearch is enabled to function. When enabled, a syndication feed will be available on the DSpace homepage (for entire site), and on each community/collection homepage (specific to that community/collection). Because Syndication Feeds use OpenSearch, all OpenSearch settings also apply to Syndication Feeds.

Property:	`websvc.opensearch.enable`
Example Value:	`webui.opensearch.enable = true`
Informational Note:	By default, OpenSearch & Syndication feeds are set to true (on) . Change key to "false" to disable. NOTE this setting affects OpenSearch Support as well

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

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 RSS/Atom feeds **require** that OpenSearch is enabled to function.

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 = true/false`
Informational Note:	Whether or not OpenSearch is enabled. By default, the feature is enabled to support RSS/Atom feeds. Change to "false" to disable.
Property:	`websvc.opensearch.svccontext`
Example Value:	`websvc.opensearch.svccontext = opensearch`
Informational Note:	The URL path where OpenSearch is made available on the backend. For example, "opensearch" means it is available at `${dspace.server.url}/opensearch`
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.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.

Content Inline Disposition Threshold / Format

The following configurations may be used to change the disposition behavior of the browser. This allows you to specify when a file (bitstream) in DSpace should be downloaded, or attempt to be opened in a user's browser.

Property:	`webui.content_disposition_threshold`
Example value:	`webui.content_disposition_threshold = 8388608`
Informational Note:	The default filesize is set to 8MB. When a file/bitstream being viewed is larger than 8MB, the user's browser will download the file to their local machine and the user will have to open it manually. All files smaller than this threshold will be sent "inline" to the user's browser, allowing the browser to decide whether to open it within the browser or download it. The value provided is always in bytes. For example: 4 MB = 4194304, 8 MB = 8388608, 16 MB = 16777216 NOTE: This threshold is only applied if the file/bitstream does NOT match the below "webui.content_disposition_format" list.
Property:	`webui.content_disposition_inline`
Example value:	`webui.content_disposition_inline = application/pdf, image/gif, image/jpeg, image/png, audio/mpeg, video/mpeg, video/mp4`
Informational Note:	Set which file mimetypes or file extensions are allowed to be opened inline in a user's browser. By default, ALL files will be downloaded, regardless of the threshold above, unless specified in this configuration. NOTE: For security reasons, some file formats (e.g. HTML, XML, RDF, JS) will always be downloaded regardless of the settings here. This blocks these formats from executing embedded JavaScript when opened inline, protecting the site from potential XSS attacks. For example, setting this to "application/pdf, image/gif, image/jpeg, image/png" will ensure that all files/bitstreams matching those MIME Types will be opened inline in the user's browser. But, all other files will be downloaded. File extensions may also be used to reference formats. For example, setting "pdf, jpg" will ensure all files ending in ".pdf" or ".jpg" will be opened in a user's browser. (NOTE: This setting replaces "webui.content_disposition_format", which is no longer supported. That prior setting listed formats **not allowed** to be opened inline, while the new setting lists formats that **are allowed** to be opended inline.)

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 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. **UNSUPPORTED IN DSpace 7.0**

Sitemap Settings

To aid web crawlers index the content within your repository, you can make use of sitemaps. For best SEO, Sitemaps are enabled by default and update automatically (see cron setting).

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. Include everything except the Sitemap UL itself (which will be URL-encoded and appended to form the actual URL "pinged").Add the following to the above parameter if you have an application ID with Yahoo: *http://search.yahooapis.com/SiteExplorererService/V1/updateNotification?appid=REPLACE_ME?url=_* . *(Replace the component _REPLACE_ME* with your application ID). There is no known "ping" URL for MSN/Live search.
Property:	`sitemap.cron`
Example Value:	`sitemap.cron = 0 15 1 * * ?`
Informational Note:	The DSpace sitemaps are regenerated on a regular basis based on the Cron syntax provided in this configuration. By default, sitemaps are updated daily at 1:15am local time. Cron syntax is defined at https://www.quartz-scheduler.org/api/2.3.0/org/quartz/CronTrigger.html Remove (comment out) this config to disable the sitemap scheduler. Sitemap scheduler can also be disabled by setting to "-" (single dash) in local.cfg.

Authority Control Settings

Two features fall under the header of Authority Control: Choice Management and Authority Control of Item ("DC") metadata values. Authority control is a fully optional feature in DSpace 1.6. Implemented out of the box are the Library of Congress Names service, and the Sherpa Romeo authority plugin.

For an in-depth description of this feature, please consult: Authority Control of Metadata Values

Property:	`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.SHERPARoMEOPublisher = SRPublisher, \`
	`org.dspace.content.authority.SHERPARoMEOJournalTitle = SRJournalTitle, \`
	`org.dspace.content.authority.SolrAuthority = SolrAuthorAuthority`

Informational Note:	List of all enabled authority control plugins
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 / sherpa.romeo.apikey`
Informational Note:	Please refers to the Sherpa/RoMEO Publishers Policy Database Integration section for details about such properties. See Configuring the Sherpa/RoMEO Publishers Policy Database Integration
Property:	`orcid.api.url`
Example Value:	`orcid.api.url = https://pub.orcid.org/v3.0`
Informational Note:	Location (URL) of the ORCID v3 Public API
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.

Configuring Multilingual Support

See Multilingual Support for more details/examples.

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.locales`
Example Value:	`webui.supported.locales = en, de`
`or perhaps`	`webui.supported.locales = en, en_ca, de`
Informational Note:	All the locales that are supported by this instance of DSpace. Comma separated list. **UNSUPPORTED IN DSpace 7.0.** *However, the DSpace 7 UI has a similar "languages" setting in environment.*.ts*

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/server/src/main/resources/Messages.properties`
- `[dspace-source]/dspace/modules/server/src/main/resources/Messages_en.properties`
- `[dspace-source]/dspace/modules/server/src/main/resources/Messages_de.properties`
- `[dspace-source]/dspace/modules/server/src/main/resources/Messages_fr.properties`

Files to be localized:

- `[dspace-source]/dspace/modules/server/src/main/resources/Messages_LOCALE.properties`
- `[dspace-source]/dspace/config/submission-forms_LOCALE.xml`
- `[dspace-source]/dspace/config/default_LOCALE.license - should be pure ASCII`
- `[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`

Upload File Settings

Property:	`upload.temp.dir`
Example Value:	`upload.temp.dir = ${dspace.dir}/upload`
Informational Note:	This property sets where DSpace temporarily stores uploaded files.

SFX Server (OpenURL)

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>
```

Controlled Vocabulary Settings

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

The need for a limited set of keywords is important since it eliminates the ambiguity of a free description system, consequently simplifying the task of finding specific items of information.

The controlled vocabulary add-on allows the user to choose from a defined set of keywords 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.

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

Request-a-copy Settings

Note: As of Dspace 9.0, these properties have been moved from **dspace.cfg** to their own module in **[dspace]/config/modules/requestitem.cfg**

Property:	`request.item.type`
Example Value	`request.item.type = all`
Informational Note	This parameter manages who can file a request for an item. The parameter is optional. When it is empty or commented out, request a copy is disabled across the entire repository. When set to **all**, any user can file a request for a copy. When set to **logged**, only registered users can file a request for copy.
Property:	`mail.helpdesk`

Example Value	`mail.helpdesk = foo@bar.com`
Informational Note	The email address assigned to this parameter will receive the emails both for granting or rejecting request a copy requests, as well as requests to change item policies. This parameter is optional. If it is empty or commented out, it will default to `mail.admin`. *WARNING:* This setting is only utilized if the `RequestItemHelpdeskStrategy` bean is enabled in `[dspace]/config/spring/api/requestitem.xml` (see below)
Property:	request.item.helpdesk.override
Example Value	`request.item.helpdesk.override = true`
Informational Note	Should all Request Copy emails go to the `mail.helpdesk` instead of the item submitter? Default is `false`, which sends Item Requests to the item submitter. *WARNING:* This setting is only utilized if the `RequestItemHelpdeskStrategy` bean is enabled in `[dspace]/config/spring/api/requestitem.xml` (see below)
Property:	request.item.reject.mail
Example Value	`request.item.reject.mail = true`
Informational Note	Should a rejection of a copy request send an email back to the requester? Defaults to "true", which means a rejection email is sent back. Setting it to "false" results in a silent rejection.
Property:	request.item.grant.bundles
Example Value	`request.item.grant.bundles = ORIGINAL`
Informational Note	Bundles to use when granting access and inspecting file size, if "all files" is indicated. This supports multiple values. To specify multiple bundles, separate them with a commao r (preferred) repeat the configuration property with additional values. Default: ORIGINAL
Property:	request.item.grant.link
Example Value	`request.item.grant.link = true`
Informational Note	Prior to DSpace 9.0, all granted requests would send the file as an attachment. If a secure web link to the bitstream(s) should be sent when files are over a certain file size threshold, enable the following property Default: true
Property:	request.item.grant.link.filesize
Example Value	`request.item.grant.link.filesize = 20`

Informational Note	If request.item.grant.link is enabled, you can specify a minimum file size (in megabytes) to use as a threshold, or else the default attachment functionality will be used instead. If 'all files' was indicated in the request, this threshold will be activated if any of the files meet the configured size minimum. To send links instead of attachments for all files, set this property to 0. Default: 20
Property:	request.item.grant.link.period
Example Values	`request.item.grant.link.period = FOREVER` `request.item.grant.link.period = +1DAY`
Informational Note	Valid access periods, in seconds, to allow the approver to select when using links. These are presented to users with friendly labels via i18n keys in the angular frontend. These should be in a format like +<n><unit>, where <n> is the number of units and <unit> is one of "DAYS", MONTHS", "YEARS". (or singular if n=1). The default is FOREVER, which will set a very distant expiry date for permanent access. The first access period in the list will be the default option in the grant form dropdown. To configure displayed labels, edit the angular .json5 i18n assets to include keys beginning with "grant-request-copy.access-period." and ending in the period, e.g.: "grant-request-copy.access-period.+1DAY": "1 day", "grant-request-copy.access-period.+7DAYS": "1 week", "grant-request-copy.access-period.+1MONTH": "1 month", "grant-request-copy.access-period.+3MONTHS": "3 months", "grant-request-copy.access-period.FOREVER": "Forever",
Property:	request.item.grant.link.dateformat
Example Value	`request.item.grant.link.dateformat = yyyy-MM-dd`
Informational Note	Date format to use for the access expiry date in the grant email
Property:	request.item.create.captcha
Example Value	`request.item.create.captcha = false`
Informational Note	Require a captcha for item request creation. Note, this requires altcha.cfg to be configured with a valid HMAC key Default: false

For more administation documentation and helpdesk strategy examples, see Request a Copy

Optional or Advanced Configuration Settings

The following section explains how to configure either optional features or advanced features that are not necessary to make DSpace "out-of-the-box"

The Metadata Format and Bitstream Format Registries

The *[dspace]/config/registries* directory contains three XML files. These are used to load the *initial* contents of the Dublin Core Metadata registry and Bitstream Format registry and SWORD metadata registry. After the initial loading (performed by *ant fresh_install* above), the registries reside in the database; the XML files are not updated.

In order to change the registries, you may adjust the XML files before the first installation of DSpace. On an already running instance it is recommended to change bitstream registries via DSpace admin UI, but the metadata registries can be loaded again at any time from the XML files without difficulty. The changes made via admin UI are not reflected in the XML files.

Metadata Format Registries

The default metadata schema is Dublin Core, so DSpace is distributed with a default Dublin Core Metadata Registry. Currently, the system requires that every item have a Dublin Core record.

There is a set of Dublin Core Elements, which is used by the system and should not be removed or moved to another schema. See Appendix: Default Dublin Core Metadata registry.

Note: altering a Metadata Registry has no effect on corresponding parts, e.g. item submission interface, item display, item import and vice versa. Every metadata element used in submission interface or item import must be registered before using it.

Note also that deleting a metadata element will delete all its corresponding values.

If you wish to add more metadata elements, you can do this in one of two ways. Via the DSpace admin UI you may define new metadata elements in the different available schemas. But you may also modify the XML file (or provide an additional one), and re-import the data as follows:

```
[dspace]/bin/dspace registry-loader -metadata [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>
```

The set of metadata registry files which is read by the MetadataImporter tool is configured by the `metadata.registry.load` property in `dspace.cfg` or `local.cfg`. If you wish to use the importer to load a new metadata namespace from a new file, you will need to add the path to your new registry file as an additional value of this property before running the tool.

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.

Configuring Usage Instrumentation Plugins

A usage instrumentation plugin is configured as a Spring bean in the applicationContext.xml for each of the various user interface web applications. It will require the injection of an instance of EventService, which it will use to register itself on the UsageEvent bus. See the configuration file for examples.

More than one such plugin may be configured – each will receive all usage events.

If you wish to write your own, it must extend the abstract class org.dspace.usage.AbstractUsageEventListener.

The Passive Plugin

The Passive plugin is provided as the class org.dspace.usage.PassiveUsageEventListener. It absorbs events without effect, and serves as a simple example of how to write a UsageEvent listener.

The Tab File Logger Plugin

The Tab File Logger plugin is provided as the class org.dspace.usage.TabFileUsageEventListener. It writes event records to a file in tab-separated column format. If left unconfigured, it will write to `[DSpace]/log/usage-events.tsv`. To specify the file path, provide an absolute path, or a path relative to log.dir, as the value for usageEvent.tabFileLogger.file in dspace.cfg.

Behavior of the workflow system

DSpace contains a workflow system to review submissions as described in detail as part of the architecture of the business logic layer and in Configurable Workflow. The file **[dspace]/config/modules/workflow.cfg** contains additional properties to configure details of the workflow system.

The property `workflow.reviewer.file-edit` controls whether files may be added/edited/removed during review (set to true) or whether files can be downloaded during review only.

[dspace]/config/modules/workflow.cfg

```
#Allow the reviewers to add/edit/remove files from the submission
#When changing this property you might want to alert submitters in the license that reviewers can alter their
files
workflow.reviewer.file-edit=false
```

The workflow system will send notifications on new Items waiting to be reviewed to all EPersons that may resolve those. Tasks can be taken to avoid that two EPersons work on the same task at the same time without knowing from each other. When a EPerson returns a task to the pool without resolving it (by accepting or rejecting the submission), another E-Mail is sent. In case you only want to be notified of completely new tasks entering a step of the workflow system, you may switch off notifications on tasks returned to the pool by setting workflow.notify.returend.tasks to false in config/modules/workflow.cfg as shown below:

[dspace]/config/modules/workflow.cfg

```
# Notify reviewers about tasks returned to the pool
workflow.notify.returned.tasks = false
```

By default notifications are sent for tasks returned to the pool.

Recognizing Web Spiders (Bots, Crawlers, etc.)

DSpace can often recognize that a given access request comes from a web spider that is indexing your repository. These accesses can be flagged for separate treatment (perhaps exclusion) in usage statistics. This requires patterns to match against incoming requests. These patterns exist in files that you will find in `config/spiders`.

In the `spiders` directory itself, you will find a number of files provided by iplists.com. These files contain network address patterns which have been discovered to identify a number of known indexing services and other spiders. You can add your own files here if you wish to exclude more addresses that you know of. You will need to include your files' names in the list configured in `config/modules/solr-statistics.cfg`. The `iplists.com-*.txt` files can be updated using a tool provided by DSpace. See SOLR Statistics for details.

In the `spiders` directory you will also find two subdirectories. `agents` contains files filled with regular expressions, one per line. An incoming request's `User-Agent` header is tested with each expression found in any of these files until an expression matches. If there is a match, the request is marked as being from a spider, otherwise not. `domains` similarly contains files filled with regular expressions which are used to test the domain name from which the request comes. You may add your own files of regular expressions to either directory if you wish to test requests with patterns of your own devising.

Command-line Access to Configuration Properties

You can resolve a configuration property name to its value using the command `dspace dsprop -p some.property.name`. The output is undecorated and may be suitable for use in scripts.

The `dsprop command has these options:`

name	argument	meaning
--property -p	name	the name of the desired configuration property. This option is required.
--module -m	name	the name of the module in which the property is found. If omitted, the value of --property is the entire name. If used, the name will be composed as module.property. For example, "`-m dspace -p url`" will look up the value of `dspace.url`.
--raw -r		if used, this prevents the substitution of other property values into the value of the requested property. It is also useful to see all of the propery values when a specific property has an **array of values** (i.e. the configuration supports specifying multiple values). Otherwise, by default , dsprop may only return the first value in the array.
--help -h -?		Display help similar to this table.

Captcha Verification settings

See: CAPTCHA Verification

Directories and Files

1 Overview
2 Source Directory Layout
3 Installed Directory Layout
4 Contents of Server Web Application
5 Log Files
    5.1 log4j2.xml File.

Overview

A complete DSpace installation consists of three separate directory trees:

- **The source directory:**: This is where (surprise!) the source code lives. Note that the config files here are used only during the initial install process. After the install, config files should be changed in the install directory. It is referred to in this document as *[dspace-source]*.
- **The install directory:**: This directory is populated during the install process and also by DSpace as it runs. It contains config files, command-line tools (and the libraries necessary to run them), and usually -- although not necessarily -- the contents of the DSpace archive (depending on how DSpace is configured). After the initial build and install, changes to config files should be made in this directory. It is referred to in this document as *[dspace]*.
- **The web deployment directory:**: This directory is generated by the web server the first time it finds a dspace.war file in its webapps directory. It contains the unpacked contents of dspace.war, i.e. the JSPs and java classes and libraries necessary to run DSpace. Files in this directory should never be edited directly; if you wish to modify your DSpace installation, you should edit files in the source directory and then rebuild. The contents of this directory aren't listed here since its creation is completely automatic. It is usually referred to in this document as *[tomcat]/webapps /dspace*.

Source Directory Layout

- *[dspace-source]*
    - *LICENSE* - DSpace source code license.
    - *README* - Obligatory basic information file.
    - *dspace/* - Directory which contains all build and configuration information for DSpace
        - *bin/* - Some shell and Perl scripts for running DSpace command-line tasks. Primary among them is the 'dspace' commandline utility
        - *config/* - Configuration files:
            - local.cfg.EXAMPLE - an example "local.cfg" file, which can be used to store all your local configuration overrides. See Configuration Reference.
            - *controlled-vocabularies/* - Fixed, limited vocabularies used in metadata entry
            - *crosswalks/* - Metadata crosswalks - property files or XSL stylesheets
            - *emails/* - Text and layout templates for emails sent out by the system.
            - entities/ - Configuration files for Configurable Entities
            - modules/ - Configurations for modules / individual features within DSpace
            - registries/ - **Initial** contents of the bitstream format registry and Dublin Core element/qualifier registry. These are only used on initial system setup, after which they are maintained in the database, but may also be reloaded later using the `[dspace]/bin/dspace registry-loader` command-line tool
            - spring/ - Spring XML configurations used by DSpace for various features.
            - *dspace.cfg* - The Main DSpace configuration file
            - *dc2mods.cfg* - Mappings from Dublin Core metadata to MODS for the METS export.
            - *default.license* - The default license that users must grant when submitting items.
            - *dstat.cfg* , *dstat.map* - Configuration for statistical reports.
            - *submission-forms.xml* , *item-submission.xml* - Submission UI configuration files
        - *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. See also Advanced Customisation
        - *solr/* - Solr configuration files for all Solr indexes used by DSpace.
        - *src/* - Maven configurations for DSpace System. This directory contains the Maven and Ant build files for DSpace.
        - *target/* - (Only exists after building DSpace) This is the location Maven uses to build your DSpace installation package.
            - *dspace-installer*- 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 (to build the dspace-api.jar)
    - *dspace-oai* - OAI-PMH source module (to build to dspace-oai.jar)
    - *dspace-rdf* - RDF source module (to build to dspace-rdf.jar)
    - dspace-server-webapp - Primary backend webapp which hosts the REST API, along with any other enabled modules (OAI, RDF, SWORD, etc).
    - *dspace-services* - Common Services module
    - *dspace-sword* - SWORD (Simple Web-service Offering Repository Deposit) deposit service source module
    - *dspace-swordv2* - SWORDv2 source module
    - *pom.xml* - DSpace Parent Project definition

Installed Directory Layout

Below is the basic layout of a DSpace installation using the default configuration. These paths can be configured if necessary.

- *[dspace]*

- *assetstore/* - assetstore files. This is where all the files uploaded into DSpace are stored by default. See Storage Layer
- *bin/* - shell scripts for DSpace command-line tasks. Primary among them is the 'dspace' commandline utility
- *config/* - configuration, with sub-directories as above
- *etc/* - administrative and database management files
- *exports/* - temporary storage for any export packages
- *handle-server/* - CNRI Handle Server files and configuration. See Handle.Net Registry Support
- *imports/* - temporary storage for any import packages
- *lib/* - JARs, including dspace-api.jar, containing the DSpace classes
- *log/* - log files
- *reports/* - reports generated by statistical report generator
- *solr/* - Solr search/browse indexes
- *triplestore/* - RDF triple store index files (when enabled)
- *upload/* - temporary directory used during file uploads etc.
- *var/* - variable runtime data. Currently used for OAI-PMH harvesting state and similar runtime artifacts
- *webapps/* - location where DSpace installs all Web Applications

DSpace upgrade and maintenance workflows (notably `ant update`) may preserve previous versions of directories and files inside the installed directory rather than overwriting them. These typically appear as:

```
bin.bak-YYYYMMDD-HHMMSS/
lib.bak-YYYYMMDD-HHMMSS/
webapps.bak-YYYYMMDD-HHMMSS/
config/[filename].cfg-YYYYMMDD-HHMMSS.old
```

These backup directories and files are not used by DSpace at runtime and may be removed once an upgrade is validated. You may wish to review them periodically, as they can grow in size over time.

Contents of Server Web Application

DSpace's Ant build file creates a webapps/*server/* directory with the following structure:

- server/
  - index.html - Root page of the third party HAL Browser (used to browse/search REST API)
  - login.html - (Custom) Login page for HAL Browser (supporting DSpace authentication plugins)
  - js/ - Javascript overrides for HAL Browser (main HAL Browser code is brought in via Spring REST dependencies)

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]/config/log4j2.xml* file.
[dspace]/log/dspace-cli.log.yyyy-mm-dd	DSpace log file for "command-line" tasks only. Any task that is run from the command-line (e.g. [dspace]/bin/dspace script) will log to this file **instead** of the main log file. You can control the verbosity of this by editing the *[dspace]/config/log4j2-cli.xml* file.
[dspace]/log/handle-plugin.log	The Handle server runs as a separate process from the DSpace Web UI (which runs under Tomcat's JVM). Due to a limitation of log4j's 'rolling file appenders', the DSpace code running in the Handle server's JVM must use a separate log file. The DSpace code that is run as part of a Handle resolution request writes log information to this file. You can control the verbosity of this by editing *[dspace]/config/log4j2-handle-plugin.xml*
[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.
[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.
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*.

log4j2.xml File.

the file *[dspace]/config/log4j2.xml* controls how and where log files are created. There are three sets of configurations in that file, called A1, and A2. These are used to control the logs for DSpace (as a whole), and the checksum checker respectively. As implied by the name, this configuration use Log4j v2. For more information on syntax, see https://logging.apache.org/log4j/2.x/manual/configuration.html

DSpace Item State Definitions

Workspace item

An item that is under submission and active edit by an authorized user. The workspace item is visible only to the submitter and the system administrators. (Currently there is no simple way to find/browse such items other than with the direct item ID or to use the supervisor functionality). Using the supervisor functionality, a system admin can allow other authorized user to see/edit the item in the workspace state.

Expected use cases:

- Self deposit
- Collaboration over an in-progress submission for a small group of researchers. (This use case is implemented only with major limitations, using the supervision feature – concurrency, lack of delegation: supervision must be defined by the system administrators, etc.)

Workflow Item

An item that is under review for quality control and policy compliance. The workflow item is visible to the original submitter (currently only basic metadata are visible out-of-box in the mydspace summary list), users assigned to the specific workflow step where the item resides, and system administrators. (Currently there is no simple way to find/browse such items other than with the direct item ID or to use the abort workflow functionality).

Expected use cases:

- Quality control
- Improvements to the bibliographic record (metadata available in workflow can be different than those asked of the submitter)
- Check of policy / copyright

Withdrawn item

It is the removal of an Item from the archive. However, a withdrawn item is still available to Administrative users (and may optionally be restored to the archive at a later date). A withdrawn item disappears from DSpace (except from Administrative screens) and the item appears to be deleted.

Expected use cases:

- Staging area for item to be removed when copyright issues arise with publisher. If the copyright issue is confirmed, the item will be permanently deleted or kept in the withdrawn state for future reference.
- Logical deletion delegated to community/collection admin, where permanent deletion is reserved to system administrators
- Logical deletion, where permanent deletion is not an option for an organization
- Removal of an old version of an item, forcing redirect to a new up-to-date version of the item (this use case is not currently implemented out-of-box in DSpace)

By design, withdrawing an item is reversible. As an administrator, you can reverse the withdrawing of an item, through the action "reinstate". As a mechanism to support this, a resource policy state "WITHDRAWN_READ" was introduced.

When as item is withdrawn, all READ policies associated with the item and its underlying bundles and bitstreams, are changed into WITHDRAWN_READ policies. This achieves 2 things:

1. The READ policy information in itself is still preserved, and can get switched back to normal READ policies if the item gets reinstated.
2. As long as the item is withdrawn, those WITHDRAWN_READ policies should not give any users or groups read rights.

WITHDRAWN_READ was introduced in DS-3097 after it was observed that even though an item was withdrawn, the related bitstreams were still accessible.

Non-Discoverable item (also known as Private item)

A non-discoverable (or "private") item is one that is simply **hidden** from all search/browse/OAI results, and is therefore only accessible via direct link (or bookmark). By default, all Items are *discoverable*, meaning they will appear in search/browse/OAI results.

It's important to clarify that non-discoverable items *may or may not be access restricted.* It is possible for an Item to be anonymously visible, but non-discoverable, so that you can only access the item if you are given a link to it.

This state should only refer to the discoverable nature of the item. A non-discoverable (or "private") item will not be included in any system that aims to help users to find items. So it will not appear in:

- Browse
- Recent submission
- Search result
- OAI-PMH (at least for the ListRecords and ListIdentifiers verb; though the OAI-PMH specification is not clear about inconsistent implementation of the ListRecords and GetRecord verb)
- REST list and search methods

It should be accessible under the actual Authorization Policies of DSpace using direct URL or query method such as:

- Splash page access (i.e. /handle/<xxxxx>/<yyyyy>)

- OAI-PMH GetRecord verb
- REST direct access /rest/item/<item-id> or equivalent

Expected use cases:

- Provide a light rights awareness feature where discovery is not enabled for search and/or browse
- Hide “special items” such as repository presentations, guides or support materials
- Hide an old version of an Item in cases where real versioning is not appropriate or liked
- Hide specific types of item such as “Item used to record Journal record: Journal Title, ISSN, Publisher etc.” used as authority file for metadata (dc.relation.ispartof) of “normal item”

Archived/Published item

An item that is in a stable state, available in the repository under the defined Authorization Policies. Changes to these items are possible only for a restricted group of users (administrators) and should produce versioning according to the Institution's policy.

Embargoed Item

Are a special case of Archived/Published Item. The item has some time based access policy attached to it and/or the underlying bitstreams. Specifically, read permission for someone (EPerson Group) starting from a defined date. Typically embargo is applied to the bitstreams so that "fulltext" has initially very limited access (normally administrators or other "repository staff" groups) and only after a defined date will the fulltext become visible to all users (Anonymous group). This scenario is used to implement typical "embargo requirements" from publishers -- see Delayed Open Access.

If the metadata of the item should be visible only to a specific group of users, it is possible to define an embargo policy also for the ITEM itself. A READ policy for a specific group will mean that only the users in that group will be able to access the item splash page. Note that the DSpace REST API & UI is fully rights aware (see Discovery documentation for more information, especially the section on "Access Rights Awareness"), meaning that an embargoed item is hidden automatically until the embargo expires.

Metadata and Bitstream Format Registries

1 Default Dublin Core Metadata Registry (DC)
2 Dublin Core Terms Registry (DCTERMS)
3 Local Metadata Registry (local)
4 Default Bitstream Format Registry
5 Registry Loader Input Formats
    5.1 Bitstream format
    5.2 Metadata format

Initial definitions for these registries may be found in `[DSpace]/config/registries`.

Default Dublin Core Metadata Registry (DC)

The default DSpace Dublin Core Metadata Registry was originally derived from the 15 Dublin Core elements. This registry initializes the default schema, where **dc** is used to identify the namespace. As this registry is meant to track the Dublin Core standard, it's recommended that the local DSpace administrator not add/remove metadata fields from this namespace; the "local" namespace should be used instead (see below).

element	qualifier	scope note
contributor		A person, organization, or service responsible for the content of the resource. Catch-all for unspecified contributors.
contributor	advisor	Use primarily for thesis advisor.
contributor	author[1]	Author(s) of the work (used by default)
contributor	editor	
contributor	illustrator	
contributor	other	
coverage	spatial	Spatial characteristics of content.
coverage	temporal	Temporal characteristics of content.
creator		May be used as an alternative to "contributor.author"
date		Use qualified form if possible.
date	accessioned[1]	Date DSpace takes possession of item.
date	available[1]	Date or date range item became available to the public.
date	copyright	Date of copyright.
date	created	Date of creation or manufacture of intellectual content if different from date.issued.
date	issued[1]	Date of publication or distribution.
date	submitted	Recommend for theses/dissertations.
identifier		Catch-all for unambiguous identifiers not defined by qualified form; use identifier.other for a known identifier common to a local collection instead of unqualified form.
identifier	citation[2]	Human-readable, standard bibliographic citation of non-DSpace format of this item
identifier	govdoc[2]	A government document number
identifier	isbn[2]	International Standard Book Number
identifier	issn[2]	International Standard Serial Number
identifier	sici	Serial Item and Contribution Identifier
identifier	ismn[2]	International Standard Music Number
identifier	other[2]	A known identifier type common to a local collection.
identifier	uri[1]	Uniform Resource Identifier
description[1]		Catch-all for any description not defined by qualifiers.
description	abstract[1]	Abstract or summary.

description	provenance[1]	The history of custody of the item since its creation, including any changes successive custodians made to it.
description	sponsorship[2]	Information about sponsoring agencies, individuals, or contractual arrangements for the item.
description	statementofresponsibility	To preserve statement of responsibility from MARC records.
description	tableofcontents	A table of contents for a given item.
description	uri	Uniform Resource Identifier pointing to description of this item.
format[2]		Catch-all for any format information not defined by qualifiers.
format	extent[2]	Size or duration.
format	medium[2]	Physical medium.
format	mimetype[2]	Registered MIME type identifiers.
language		Catch-all for non-ISO forms of the language of the item, accommodating harvested values.
language	iso[2]	Current ISO standard for language of intellectual content, including country codes (e.g. "en_US").
publisher[2]		Entity responsible for publication, distribution, or imprint.
relation		Catch-all for references to other related items.
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[2]		Uncontrolled index term.
subject	classification	Catch-all for value from local classification system. Global classification systems will receive specific qualifier
subject	ddc	Dewey Decimal Classification Number
subject	lcc	Library of Congress Classification Number
subject	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	alternative[2]	Varying (or substitute) form of title proper appearing in item, e.g. abbreviation or translation
type[1]		Nature or genre of content.

[1] Used by several functional areas of DSpace. **DO NOT REMOVE WITHOUT INVESTIGATING THE CONSEQUENCES**

[2] This field is included in the default DSpace Submission User Interface. Removing this field from your registry will break the default DSpace submission form.

Dublin Core Terms Registry (DCTERMS)

The Dublin Core Terms (DCTERMS) registry was introduced in DSpace 4. This registry initializes an optional metadata schema, where **dcterms** is used to identify the namespace. In DSpace 4, none of these fields are used by any of the system functionality out of the box. The registry and schema were added as a first step to facilitate the future migration of the DSpace specific DC schema, to this schema that complies to current Dublin Core standards.

The main advantage of the DCTERMS schema is that no field name details gets lost during harvesting, as opposed to harvesting of so called "simple" dublin core, where the qualifiers from the above schema are omitted during harvesting.

As this registry is meant to track the Dublin Core Terms standard, it's recommended that the local DSpace administrator not add/remove metadata fields from this namespace; the "local" namespace should be used instead (see below).

term	scope note
abstract	A summary of the resource.
accessRights	Information about who can access the resource or an indication of its security status. May include information regarding access or restrictions based on privacy, security, or other policies.
accrualMethod	The method by which items are added to a collection.
accrualPeriodicity	The frequency with which items are added to a collection.
accrualPolicy	The policy governing the addition of items to a collection.
alternative	An alternative name for the resource.
audience	A class of entity for whom the resource is intended or useful.
available	Date (often a range) that the resource became or will become available.
bibliographicCitation	Recommended practice is to include sufficient bibliographic detail to identify the resource as unambiguously as possible.
conformsTo	An established standard to which the described resource conforms.
contributor	An entity responsible for making contributions to the resource. Examples of a Contributor include a person, an organization, or a service.
coverage	The spatial or temporal topic of the resource, the spatial applicability of the resource, or the jurisdiction under which the resource is relevant.
created	Date of creation of the resource.
creator	An entity primarily responsible for making the resource.
date	A point or period of time associated with an event in the lifecycle of the resource.
dateAccepted	Date of acceptance of the resource.
dateCopyrighted	Date of copyright.
dateSubmitted	Date of submission of the resource.
description	An account of the resource.
educationLevel	A class of entity, defined in terms of progression through an educational or training context, for which the described resource is intended.
extent	The size or duration of the resource.
format	The file format, physical medium, or dimensions of the resource.
hasFormat	A related resource that is substantially the same as the pre-existing described resource, but in another format.
hasPart	A related resource that is included either physically or logically in the described resource.
hasVersion	A related resource that is a version, edition, or adaptation of the described resource.
identifier	An unambiguous reference to the resource within a given context.
instructionalMethod	A process, used to engender knowledge, attitudes and skills, that the described resource is designed to support.

isFormatOf	A related resource that is substantially the same as the described resource, but in another format.
isPartOf	A related resource in which the described resource is physically or logically included.
isReferencedBy	A related resource that references, cites, or otherwise points to the described resource.
isReplacedBy	A related resource that supplants, displaces, or supersedes the described resource.
isRequiredBy	A related resource that requires the described resource to support its function, delivery, or coherence.
issued	Date of formal issuance (e.g., publication) of the resource.
isVersionOf	A related resource of which the described resource is a version, edition, or adaptation.
language	A language of the resource.
license	A legal document giving official permission to do something with the resource.
mediator	An entity that mediates access to the resource and for whom the resource is intended or useful.
medium	The material or physical carrier of the resource.
modified	Date on which the resource was changed.
provenance	A statement of any changes in ownership and custody of the resource since its creation that are significant for its authenticity, integrity, and interpretation.
publisher	An entity responsible for making the resource available.
references	A related resource that is referenced, cited, or otherwise pointed to by the described resource.
relation	A related resource.
replaces	A related resource that is supplanted, displaced, or superseded by the described resource.
requires	A related resource that is required by the described resource to support its function, delivery, or coherence.
rights	Information about rights held in and over the resource.
rightsHolder	A person or organization owning or managing rights over the resource.
source	A related resource from which the described resource is derived.
spatial	Spatial characteristics of the resource.
subject	The topic of the resource.
tableOfContents	A list of subunits of the resource.
temporal	Temporal characteristics of the resource.
title	A name given to the resource.
type	The nature or genre of the resource.
valid	Date (often a range) of validity of a resource.

Local Metadata Registry (local)

Editing the DC and DCTERMS schemas is recommended against because it may complicate the upgrade path in case a newer version of DSpace needs to make changes or migrations in these standard metadata fields. Therefore, an empty metadata schema called "local" is provided (since DSpace 6), which can be used by the DSpace administrator as a namespace for custom local metadata fields. Such custom fields would be anything that does not fit into DC or DCTERMS. Future DSpace migrations will not touch fields in the "local" namespace.

element	qualifier	scope note
<empty by default>		<fields to be populated by DSpace administrator if needed>

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 several functional areas of DSpace. **DO NOT REMOVE WITHOUT INVESTIGATING THE CONSEQUENCES**

Registry Loader Input Formats

Registries are installed using the `bin/dspace registry-loader` command. Metadata registries may also be built and managed through the Web interface using the Metadata Registry Management feature.

The registry loader understands two separate XML schemae representing metadata and bitstream format registries.

Bitstream format

```
<?xml version='1.0'?>
<dspace-bitstream-types>
  <bitstream-type>
    <mimetype>mime/type</mimetype>
    <short_description>Thing v1</short_description>
    <description>Thing Version 1 Document Format</description>
    <support_level>1</support_level>
    <internal>false</internal>
    <extension>thingdoc</extension>
    <extension>thd</extension>
  </bitstream-type>
     ...
</dspace-bitstream-types>
```

element	meaning
mimetype	Media Type of the format
short_description	a brief description of the format, such as "PDF" or "plain text"
description	a fuller description, such as "Portable Document Format"
support_level	a number: 1 = unknown, 2 = known, 3 = supported
internal	`true` if the format is used by DSpace and should not be altered or removed
extension	a filename extension which implies this format, such as `.pdf` May be repeated to register multiple extensions.

Metadata format

```
<?xml version="1.0"?>
<dspace-dc-types>
  <dspace-header>
    <title></title>
    <contributor.author>Alice</contributor.author>
        <contributor.editor>Bob</contributor.editor>
        <date-created>01-Jan-1990</date-created>
        <description>The Example Nonexistent metadata schema</description>
    <description.version>1.0</description.version>
  </dspace-header>

  <dc-schema>
    <name>ne</name>
    <namespace>https://example.com/schemas/nonexistent/1.0</namespace>
  </dc-schema>

  <dc-type>
    <schema>ne</schema>
    <element>type</element>
    <qualifier>subtype</qualifier>
    <scope_note>unused</scope_note>
  </dc-type>
    ...
</dspace-dc-types>
```

element	meaning
dspace-header	Documentation of the format document itself. Not currently used by DSpace.
dc-schema	Identifies a metadata schema.
name	A short label for the schema: a local name.
namespace	The unique identifier for the schema, usually a URI: the global name.

dc-type	Defines a single metadata field within the schema.
schema	The local name for the schema which defines this field, such as "dc". This should match one of the known <dc-schema><name> names
element	The major identifier for the field, such as "contributor"
qualifier	The qualifier or minor identifier for the field, which names a sub-type of the major identifier, such as "author". May be omitted.
scope_note	Brief explanation of the use(s) of the field. May be omitted.

History

- Changes in 10.x
- Changes in Older Releases

Changes in 10.x

- Changes in DSpace 10.0

Changes in DSpace 10.0

- UI Changes: https://github.com/DSpace/dspace-angular/milestone/30?closed=1
- REST API Changes: https://github.com/DSpace/DSpace/milestone/78?closed=1
- REST Contract Changes: https://github.com/DSpace/RestContract/milestone/21?closed=1

Changes in Older Releases

All historical changes from older releases of DSpace may be found in the online documentation for those older releases.