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 DSpace Database Configuration
  - 3.2.1 To provide the database connection pool externally
- 3.3 DSpace Email Settings
  - 3.3.1 Wording of E-mail Messages
- 3.4 File Storage
- 3.5 Logging Configuration
- 3.6 General Plugin Configuration
- 3.7 Configuring the Search Engine
- 3.8 Handle Server Configuration
- 3.9 Delegation Administration: Authorization System Configuration
  - 3.9.1 Login as feature
- 3.10 Restricted Item Visibility Settings
- 3.11 Proxy Settings
- 3.12 Configuring Media Filters
- 3.13 Crosswalk and Packager Plugin Settings
  - 3.13.1 Configurable MODS Dissemination Crosswalk
  - 3.13.2 XSLT-based Crosswalks
    - 3.13.2.1 Testing XSLT Crosswalks
  - 3.13.3 Configurable Qualified Dublin Core (QDC) dissemination crosswalk
  - 3.13.4 Configuring Crosswalk Plugins
  - 3.13.5 Configuring Packager Plugins
- 3.14 Event System Configuration
- 3.15 Embargo
- 3.16 Checksum Checker Settings
- 3.17 Item Export and Download Settings
- 3.18 Subscription Emails
- 3.19 Hiding Metadata
- 3.20 Settings for the Submission Process
- 3.21 Configuring the Sherpa/RoMEO Publishers Policy Database Integration
- 3.22 Configuring Creative Commons License
- 3.23 WEB User Interface Configurations
- 3.24 Browse Index Configuration
  - 3.24.1 Defining the storage of the Browse Data
  - 3.24.2 Defining the Indexes
  - 3.24.3 Defining Sort Options
  - 3.24.4 Other Browse Options
  - 3.24.5 Browse Index Authority Control Configuration
  - 3.24.6 Tag cloud
- 3.25 Author (Multiple metadata value) Display
- 3.26 Links to Other Browse Contexts
- 3.27 Recent Submissions
- 3.28 Submission License Substitution Variables
- 3.29 Syndication Feed (RSS) Settings
- 3.30 OpenSearch Support
- 3.31 Content Inline Disposition Threshold

3.32 Multi-file HTML Document/Site Settings
3.33 Sitemap Settings
3.34 Authority Control Settings
3.35 Configuring Multilingual Support
3.35.1 Setting the Default Language for the Application
3.35.2 Supporting More Than One Language
3.35.2.1 Changes in dspace.cfg
3.35.2.2 Related Files
3.36 JSPUI Upload File Settings
3.37 JSP Web Interface (JSPUI) Settings
3.38 JSPUI Item Mapper
3.39 Display of Group Membership
3.40 JSPUI / XMLUI SFX Server
3.41 JSPUI Item Recommendation Setting
3.42 Controlled Vocabulary Settings
3.43 XMLUI Specific Configuration
4 Optional or Advanced Configuration Settings
4.1 The Metadata Format and Bitstream Format Registries
4.1.1 Metadata Format Registries
4.1.2 Bitstream Format Registry
4.2 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 JSPUI: Per item visual indicators for browse and search results
4.5 Recognizing Web Spiders (Bots, Crawlers, etc.)
5 Command-line Access to Configuration Properties

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)
  - `input-forms.xml` : The default deposit input forms for DSpace (see Submission User Interface documentation)
  - `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)
  - `log4j.properties` : The default logging settings for DSpace log files (usually placed in `[dspace]/log`)
  - `news-side.html` and `news-top.html` : HTML news configuration files for the JSPUI homepage (see JSPUI Configuration and Customization)
  - `news-xmlui.xml` : News configuration file for the XMLUI homepage (see XMLUI Configuration and Customization)
  - `workflow.xml` : Configuration for the Configurable Workflow feature (not used by default)
  - `xmlui.xconf` : Configuration for the XMLUI (see XMLUI Configuration and Customization)

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

> In DSpace 6.0, while the DSpace API supports dynamically reloading configurations, the user interfaces (e.g. XMLUI and JSPUI) 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). This is something we are working to correct in future versions of DSpace.

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
  - `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, 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

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.

As of DSpace 6, it is now 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.baseUrl" in dspace.cfg
dspace.baseUrl = http://dspace.myuniversity.edu

# The overrides the default "dspace.url" setting it to the same value as
my "baseUrl" above
dspace.url = ${dspace.baseUrl}

# 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

# For the example, we'll override the default oai.url in /config/modules
/oai.cfg
oai.url = ${dspace.baseUrl}/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` configuraiton file.

Main DSpace Configurations

Property:	`dspace.dir`
Example Value:	`/dspace`
Informational Note:	Root directory of DSpace installation. Omit the trailing slash '/'. Note that if you change this, there are several other parameters you will probably want to change to match, e.g. `assetstore.dir` . (*On Windows be sure to use forward slashes for the directory path!* For example: "C:/dspace" is a valid path for Windows.)
Property:	`dspace.hostname`
Example Value:	`dspace.hostname = dspace.mysu.edu`
Informational Note:	Fully qualified hostname; do not include port number.
Property:	`dspace.baseUrl`
Example Value:	http://dspacetest.myu.edu:8080
Informational Note:	Main URL at which DSpace Web UI webapp is deployed. Include any port number, but do not include the trailing '/'.
Property:	`dspace.url`
Example Value:	`dspace.url = http://dspacetest.myu.edu:8080/xmlui`
Informational note	URL that determines whether JSPUI or XMLUI will be loaded by default. Include port number etc., but NOT trailing slash. Alternatively to the example, this url can have /jspui at the end if you are using jspui instead of xmlui. You can also opt to run your UI app as your servlet engine's "ROOT" webapp. In that case, ensure that you remove /xmlui or /jspui.
Property:	`dspace.name`
Example Value:	`dspace.name = DSpace at My University`
Informational Note:	Short and sweet site name, used throughout Web UI, e-mails and elsewhere (such as OAI protocol)

DSpace Database Configuration

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

Property:	`db.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.
Property:	`db.maxconnections`
Example Value:	`db.maxconnections = 30`
Informational Note:	Maximum number of Database connections in the connection pool
Property:	`db.maxwait`
Example Value:	`db.maxwait = 5000`
Informational Note:	Maximum time to wait before giving up if all connections in pool are busy (in milliseconds).
Property:	`db.maxidle`
Example Value:	`db.maxidle = -1`
Informational Note:	Maximum number of idle connections in pool. (*-1* = unlimited)
Property:	`db.statementpool`
Example Value:	`db.statementpool = true`
Informational Note:	Determines if prepared statement should be cached. (Default is set to true)
Property:	`db.poolname`
Example Value:	`db.poolname = dspacepool`
Informational Note:	Specify a name for the connection pool. This is useful if you have multiple applications sharing Tomcat's database connection pool. If nothing is specified, it will default to 'dspacepool'

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.

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. Since DSpace 4.0, this is also the email address displayed on the contacts page.
Property:	`mail.admin`
Example Value:	`mail.admin = dspace-help@myu.edu`
Informational Note:	Email address of the general site administrator (Webmaster)
Property:	`alert.recipient`
Example Value:	`alert.recipient = john.doe@myu.edu`
Informational Note:	Enter the recipient for server errors and alerts. This property is optional.
Property:	`registration.notify`
Example Value:	`registration.notify = mike.smith@myu.edu`
Informational Note:	Enter the recipient that will be notified when a new user registers on DSpace. This property is optional.
Property:	`mail.charset`
Example Value:	`mail.charset = UTF-8`

Informational Note:	Set the default mail character set. This may be over-ridden by providing a line inside the email template '*charset: <encoding>*', otherwise this default is used.
Property:	`mail.allowed.referrers`
Example Value:	`mail.allowed.referrers = localhost`
Informational Note:	A comma separated list of hostnames that are allowed to refer browsers to email forms. Default behavior is to accept referrals only from *dspace.hostname*. This property is optional.
Property:	`mail.extraproperties`

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

Informational Note:	If you need to pass extra settings to the Java mail library. Comma separated, equals sign between the key and the value. This property is optional.
Property:	`mail.server.disabled`
Example Value:	`mail.server.disabled = false`
Informational Note:	An option is added to disable the mailserver. By default, this property is set to '`false`'. By setting value to '`true`', DSpace will not send out emails. It will instead log the subject of the email which should have been sent. This is especially useful for development and test environments where production data is used when testing functionality. This property is optional.
Property:	`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 *input-forms.xml*, the default language will be attributed to the metadata values.

Wording of E-mail Messages

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

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

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

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

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

Property:	`log.dir`
Example value:	`log.dir = ${dspace.dir}/log`
Informational Note:	This is where to put the logs. (This is used for initial configuration only)
Property:	`loglevel.dspace` (*defined in log4j.properties*)
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 log4j.properties*)
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.

Previous releases of DSpace provided an example *${dspace.dir}/config/log4j.xml* as an alternative to *log4j.properties*. This caused some confusion and has been removed. log4j continues to support both Properties and XML forms of configuration, and you may continue (or begin) to use any form that log4j supports.

General Plugin Configuration

Property:	**`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

> Since DSpace 4.0 the advanced search module named Discovery (based on Apache SOLR) is the default search provider. It provides up-to-date features, such as filtering/faceting, hit highlighting, search snippets, etc.
>
> 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 The Handle Server section of Installing DSpace.

Property:	`handle.canonical.prefix`
Example Value	handle.canonical.prefix = http://hdl.handle.net/ handle.canonical.prefix = ${dspace.url}/handle/
Informational Note:	Canonical Handle URL prefix. By default, DSpace is configured to use http://hdl.handle.net/ as the canonical URL prefix when generating `dc.identifier.uri` during submission, and in the 'identifier' displayed in item record pages. If you do not subscribe to CNRI's handle service, you can change this to match the persistent URL service you use, or you can force DSpace to use your site's URL, e.g. `handle.canonical.prefix = ${dspace.url}/handle/`. Note that this will not alter `dc.identifer.uri` metadata for existing items (only for subsequent submissions).
Property:	`handle.prefix`
Example Value	`handle.prefix = 1234.56789`
Informational Note:	The default installed by DSpace is `123456789` but you will replace this upon receiving a handle from CNRI.
Property:	`handle.dir`

Example Value:	`handle.dir = ${dspace.dir}/handle-server`
Informational Note:	The default files, as shown in the Example Value is where DSpace will install the files used for the Handle Server.
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.

Community Administration:
The properties for collection administrators work similar to those of community administrators,
with respect to collection administration.
	`core.authorization.collection-admin.policies`
	`core.authorization.collection-admin.template-item`
	`core.authorization.collection-admin.submitters`
	`core.authorization.collection-admin.workflows`
	`core.authorization.collection-admin.admin-group`

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

Collection Administration:
Bundles of bitstreams, related to items owned by collections in the above Community. The properties for collection administrators work similar to those of community administrators, with respect to administration of bitstreams related to items in underlying collections.
	`core.authorization.collection-admin.item.create-bitstream`
	`core.authorization.collection-admin.item.delete-bitstream`
	`core.authorization.collection-admin.item-admin.cc-license`

Item Administration. The properties for item administrators work similar to those of community and collection administrators, with respect to administration of items in underlying collections.	`core.authorization.item-admin.policies`

Item Administration:
Bundles of bitstreams, related to items owned by collections in the above Community. The properties for item administrators work similar to those of community and collection administrators,
with respect to administration of bitstreams
related to items in underlying collections.
	`core.authorization.item-admin.create-bitstream`
	`core.authorization.item-admin.delete-bitstream`
	`core.authorization.item-admin.cc-license`

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. **Please note that this configuration parameter has changed name in DSpace 4.0 from xmlui.user.assumelogin to webui.user.assumelogin as it is now supported also in the JSP UI**

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.

Proxy Settings

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

Property:	`http.proxy.host`
Example Value	`http.proxy.host = proxy.myu.edu`
Informational Note	Enter the host name without the port number.
Property:	`http.proxy.port`
Example Value	`http.proxy.port = 2048`
Informational Note	Enter the port number for the proxy server.
Property	`useProxies`
Example Value:	`useProxies = true`
Informational Note:	If your DSpace instance is protected by a proxy server, in order for log4j to log the correct IP address of the user rather than of the proxy, it must be configured to look for the X-Forwarded-For header. This feature can be enabled by ensuring this setting is set to *true (default is false).* This also affects IPAuthentication, and should be enabled for that to work properly if your installation uses 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, Html Text Extractor, \`
	
	`Word Text Extractor, JPEG Thumbnail`

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

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

Example Value:	`plugin.named.org.dspace.app.mediafilter.FormatFilter = \`
	`org.dspace.app.mediafilter.PDFFilter = PDF Text Extractor, \`
	`org.dspace.app.mediafilter.HTMLFilter = HTML Text Extractor, \`
	`org.dspace.app.mediafilter.WordFilter = Word Text Extractor, \`
	`org.dspace.app.mediafilter.JPEGFilter = JPEG Thumbnail, \`
	`org.dspace.app.mediafilter.BrandedPreviewJPEGFilter = Branded Preview JPEG`

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

Property:	`filter.org.dspace.app.mediafilter.PDFFilter.inputFormats`
	`filter.org.dspace.app.mediafilter.HTMLFilter.inputFormats`
	`filter.org.dspace.app.mediafilter.WordFilter.inputFormats`
	`filter.org.dspace.app.mediafilter.JPEGFilter.inputFormats`
	`filter.org.dspace.app.mediafilter.BrandedPreviewJPEGFilter.inputFormats`

Example Value:	`filter.org.dspace.app.mediafilter.PDFFilter.inputFormats = Adobe PDF`
	`filter.org.dspace.app.mediafilter.HTMLFilter.inputFormats = HTML, Text`
	`filter.org.dspace.app.mediafilter.WordFilter.inputFormats = Microsoft Word`
	`filter.org.dspace.app.mediafilter.JPEGFilter.inputFormats = BMP, GIF, JPEG, \`
	`image/png`
	`filter.org.dspace.app.mediafilter.BrandedPreviewJPEGFilter.inputFormats = BMP, \`
	`GIF, JPEG, image/png`

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

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

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

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

> **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 "`crosswalk.mods.properties.`"; the final word of the property name becomes the plugin's name. For example, a property name `crosswalk.mods.properties.MODS` defines a crosswalk plugin named "`MODS`".

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

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

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

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

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

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

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

XSLT-based Crosswalks

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

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

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

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

```
crosswalk.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.

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 EventSystemPrototype.

Property:	`event.dispatcher.default.class`
Example Value:	`event.dispatcher.default.class = org.dspace.event.BasicDispatcher`
Informational Note:	This is the default synchronous dispatcher (Same behavior as traditional DSpace).
Property:	`event.dispatcher.default.consumers`
Example Value:	`event.dispatcher.default.consumers = search, browse, eperson`
Informational Note:	This is the default synchronous dispatcher (Same behavior as traditional DSpace).
Property:	`event.dispatcher.noindex.class`
Example Value:	`event.dispatcher.noindex.class = org.dspace.event.BasicDispatcher`
Informational Note:	The noindex dispatcher will not create search or browse indexes (useful for batch item imports).
Property:	`event.dispatcher.noindex.consumers`
Example Value:	`event.dispatcher.noindex.consumers = eperson`
Informational Note:	The noindex dispatcher will not create search or browse indexes (useful for batch item imports).
Property:	`event.consumer.search.class`
Example Value:	`event.consumer.search.class = org.dspace.search.SearchConsumer`
Informational Note:	Consumer to maintain the search index.
Property:	`event.consumer.search.filters`
Example Value:	{{event.consumer.search.filters = }} `Community | Collection | Item | Bundle+Add | Create | Modify | Modify_Metadata | Delete | Remove`
Informational Note:	Consumer to maintain the search index.
Property:	`event.consumer.browse.class`
Example Value:	`event.consumer.browse.class = org.dspace.browse.BrowseConsumer`
Informational Note:	Consumer to maintain the browse index.
Property:	`event.consumer.browse.filters`
Example Value:	`event.consumer.browse.filters = Community | Collection | Item | Bundle+Add | Create | Modify | Modify_Metadata | Delete | Remove`
Informational Note:	Consumer to maintain the browse index.
Property:	`event.consumer.eperson.class`
Example Value:	`event.consumer.eperson.class = org.dspace.eperson.EPersonConsumer`
Informational Note:	Consumer related to EPerson changes

Property:	`event.consumer.eperson.filters`
Example Value:	`event.consumer.eperson.filters = EPerson+Create`
Informational Note:	Consumer related to EPerson changes
Property:	`event.consumer.test.class`
Example Value:	`event.consumer.test.class = org.dspace.event.TestConsumer`
Informational Note:	Test consumer for debugging and monitoring. **Commented out by default.**
Property:	`event.consumer.test.filters`
Example Value:	`event.consumer.test.filters = All+All`
Informational Note:	Test consumer for debugging and monitoring. **Commented out by default.**
Property:	`testConsumer.verbose`
Example Value:	`testConsumer.verbose = true`
Informational Note:	Set this to true to enable testConsumer messages to standard output. **Commented out by default.**

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 now comes with a Checksum Checker script (`[dspace]/bin/dspace checker`) which can be scheduled to verify the checksum of every item within DSpace. Since DSpace calculates and records the checksum of every file submitted to it, this script is able to determine whether or not a file has been changed (either manually or by some sort of corruption or virus). The idea being that the earlier you can identify a file has changed, the more likely you'd be able to recover it (assuming it was not a wanted change).

Property:	`plugin.single.org.dspace.checker. BitstreamDispatcher`
Example Value:	`plugin.single.org.dspace.checker. BitstreamDispatcher = org.dspace.checker. SimpleDispatcher`
Informational Note:	The Default dispatcher is case non is specified.
Property:	`checker.retention.default`
Example Value:	`checker.retention.default = 10y`
Informational Note:	This option specifies the default time frame after which all checksum checks are removed from the database (defaults to 10 years). This means that after 10 years, all successful or unsuccessful matches are removed from the database.
Property:	`checker.retention.CHECKSUM_MATCH`
Example Value:	`checker.retention.CHECKSUM_MATCH = 8w`
Informational Note:	This option specifies the time frame after which a successful match will be removed from your DSpace database (defaults to 8 weeks). This means that after 8 weeks, all successful matches are automatically deleted from your database (in order to keep that database table from growing too large).

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 now possible to hide metadata from public consumption that is only available to the Administrator.

Property:	`metadata.hide.dc.description.provenance`
Example Value:	`metadata.hide.dc.description.provenance = true`
Informational Note:	Hides the metadata in the property key above except to the administrator. Fields named here are hidden in the following places UNLESS the logged-in user is an Administrator: 1. XMLUI metadata XML view, and Item splash pages (long and short views). 2. JSPUI Item splash pages 3. OAI-PMH server. 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.

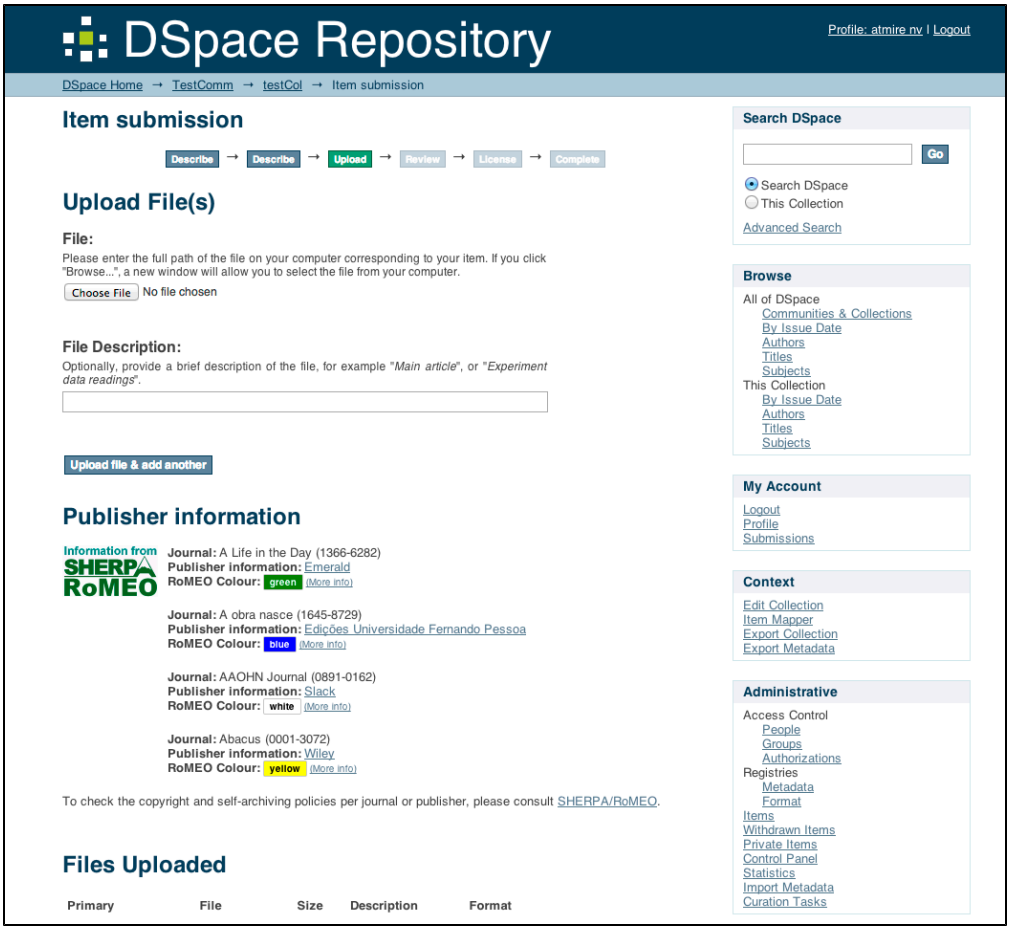
Settings for the Submission Process

These settings control three aspects of the submission process: thesis submission permission, whether or not a bitstream file is required when submitting to a collection and whether to show a progress bar during the file upload.

Property:	`webui.submit.blocktheses`
Example Value:	`webui.submit.blocktheses = false`
Informational Note:	Controls whether or not the UI blocks a submission which is marked as a thesis.
Property:	`webui.submit.upload.required`
Example Value:	`webui.submit.upload.required = true`
Informational Note:	Whether or not a file is **required** to be uploaded during the "Upload" step in the submission process. The default is true. If set to "false", then the submitter (human being) has the option to skip the uploading of a file.
Property:	`webui.submit.upload.html5`
Example Value:	`webui.submit.upload.html5 = true`
Informational Note:	If the browser supports it, JSPUI uses html5 File API to enhance file upload. If this property is set to false the enhanced file upload is not used even if the browser would support it.
Property:	`webui.submit.upload.progressbar`
Example Value:	`webui.submit.upload.progressbar = true`

Informational Note:	Whether to show a progress bar during file upload. Please note that to work this feature requires a JSON endpoint (json /uploadProgress) that is enabled by default. See the named plugin for the interface org.dspace.app.webui.json.JSONRequest
	
	`org.dspace.app.webui.json.`
	`UploadProgressJSON = uploadProgress`
	
	This property is actually supported only by the JSPUI. The XMLUI doesn't yet provide a progress bar indicator for file upload.

Configuring the Sherpa/RoMEO Publishers Policy Database Integration



DSpace 4.0 introduced integration with the Sherpa/RoMEO Publishers Policy Database in order to allow displaying the publisher policy in the submission upload step. The submission step interface is available in JSPUI (since DSpace 4.0) and in XMLUI (since DSpace 5.0) and enabled by default, however to use it in production (over 500 requests per day), you must register for a free API key (see below for details).

Property:	`webui.submission.sherparomeo-policy-enabled`
Example Value:	`webui.submission.sherparomeo-policy-enabled = true`
Informational Note:	Controls whether or not the UI submission should try to use the Sherpa/RoMEO Publishers Policy Database Integration (default **true**)
Property:	sherpa.romeo.url
Example Value:	sherpa.romeo.url = http://www.sherpa.ac.uk/romeo/api29.php
Informational Note:	The Sherpa/RoMEO endpoint. Shared with the authority control feauture for Journal Title autocomplete see AuthorityControlSettings
Property:	sherpa.romeo.apikey
Example Value:	`sherpa.romeo.apikey = YOUR-API-KEY`
Informational Note:	Allow to use a specific API key to raise the usage limit (500 calls /day for unregistred user). You can register for a free api access key at http://www.sherpa.ac.uk/news/romeoapikeys.htm

The functionality rely on understanding to which Journal (ISSN) is related the submitting item. This is done out of box looking to some item metadata but a different strategy can be used as for example look to a metadata authority in the case that the Sherpa/RoMEO autocomplete for Journal is used (see AuthorityControlSettings)

The strategy used to discover the Journal related to the submission item is defined in the spring file **/config/spring/api/sherpa.xml**

```
<bean class="org.dspace.app.sherpa.submit.SHERPASubmitConfigurationService"
                id="org.dspace.app.sherpa.submit.
SHERPASubmitConfigurationService">
                <property name="issnItemExtractors">
                        <list>
                                <bean class="org.dspace.app.sherpa.submit.
MetadataValueISSNExtractor">
                                        <property name="metadataList">
                                                <list>
                                                        <value>dc.
identifier.issn</value>
                                                </list>
                                        </property>
                                </bean>
                                <!-- Use the follow if you have the
SHERPARoMEOJournalTitle enabled
                                <bean class="org.dspace.app.sherpa.submit.
MetadataAuthorityISSNExtractor">
                                        <property name="metadataList">
                                                <list>
                                                        <value>dc.title.
alternative</value>
                                                </list>
                                        </property>
                                </bean>  -->
                        </list>
                </property>
        </bean>
```

Configuring Creative Commons License

The following configurations are for the Creative Commons license step in the submission process. Submitters are given an opportunity to select a Creative 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 (step 5), but it is commented out, so enabling Creative Commons licensing is typically just a matter of uncommenting the CC License step.

Since DSpace 5.6 Creative Commons licensing is captured in exactly the same way in each UI. The Creative Commons REST 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)

Behaviour change

Since DSpace 5.6 Creative Commons licensing is captured in exactly the same way in each UI and some fix has been introduced.

For JSPUI users this mean:

- The full (HTML) text of the CC License is not longer stored in a bitstream named "license_txt" in the CC-LICENSE bundle
- Previous existent license_txt remain untouched but new item will not receive such bitstream

For XMLUI users:

- the RDF version of the CC License is now stored properly without the Creative Commons API XML envelop (**DS-3326** - Getting issue details... STATUS)
- previous RDF license, i.e. the one associated with item created with version less than 5.6 remain untouched

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. If you change from the default value (dc.rights.uri), you will have to reconfigure the XMLUI for proper display of license data`
Property:	`cc.license.name`
Example Value:	`cc.license.name = dc.rights`
Informational Note:	`The field that holds the Creative Commons license Name. If you change from the default value (dc.rights), you will have to reconfigure the XMLUI for proper display of license data`
Property:	`cc.submit.setname`
Example Value:	`cc.submit.setname = true`
Informational Note:	`If true, the license assignment will add the field configured with the "cc.license.name" with the name of the CC license; if false, only "cc.license.uri" field is added.`
Property:	`cc.submit.addbitstream`
Example Value:	`cc.submit.addbitstream = true`
Informational Note:	`If true, the license assignment will add a bitstream with the CC license RDF; if false, only metadata field(s) are added.`
Property:	`cc.license.classfilter`
Example Value:	`cc.license.classfilter = recombo,mark`
Informational Note:	`This list defines the values that will be excluded from the license (class) selection list, as defined by the web service at the URL: http://api.creativecommons.org/rest/1.5/classes`
Property:	`cc.license.jurisdiction`
Example Value:	`cc.license.jurisdiction = nz`

Informational Note:	Should a jurisdiction be used? If so, which one? See http://creativecommons.org/international/ for a list of possible codes (e.g. nz = New Zealand, uk = England and Wales, jp = Japan) 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.

WEB User Interface Configurations

General Web User Interface Configurations
In this section of Configuration, we address the agnostic WEB User Interface that is used for JSPUI and XMLUI. Some of the configurations will give information towards customization or refer you to the appropriate documentation.

Property:	`webui.licence_bundle.show`
Example Value:	`webui.licence_bundle.show = false`
Informational Note:	Sets whether to display the contents of the license bundle (often just the deposit license in the standard DSpace installation).
Property:	`webui.browse.thumbnail.show`
Example Value:	`webui.browse.thumbnail.show = true`
Informational Note:	Controls whether to display thumbnails on browse and search result pages. If you have customized the Browse columnlist, then you must also include a "thumbnail" column in your configuration. *(This configuration property key is not used by XMLUI. To show thumbnails using XMLUI, you need to create a theme which displays them).*
Property:	`webui.browse.thumbnail.maxheight`
Example Value:	`webui.browse.thumbnail.maxheight = 80`
Informational Note:	This property determines the maximum height of the browse/search thumbnails in pixels (px). This only needs to be set if the thumbnails are required to be smaller than the dimensions of thumbnails generated by MediaFilter.
Property:	`webui.browse.thumbnail.maxwidth`
Example Value:	`webui.browse.thumbnail.maxwidth = 80`
Informational Note:	This determines the maximum width of the browse/search thumbnails in pixels (px). This only needs to be set if the thumbnails are required to be smaller than the dimensions of thumbnails generated by MediaFilter.
Property:	`webui.item.thumbnail.show`
Example Value:	`webui.item.thumbnail.show = true`
Informational Note:	This determines whether or not to display the thumbnail against each bitstream. *(This configuration property key is not used by XMLUI. To show thumbnails using XMLUI, you need to create a theme which displays them).*
Property:	`webui.browse.thumbnail.linkbehavior`
Example Value:	`webui.browse.thumbnail.linkbehavior = item`
Informational Note:	This determines where clicks on the thumbnail in browse and search screens should lead. The only values currently supported are "item" or "bitstream", which will either take the user to the item page, or directly download the bitstream.
Property:	`thumbnail.maxwidth`

Example Value:	`thumbnail.maxwidth = 80`
Informational Note:	This property sets the maximum width of generated thumbnails that are being displayed on item pages.
Property:	`thumbnail.maxheight`
Example Value:	`thumbnail.maxheight = 80`
Informational Note:	This property sets the maximum height of generated thumbnails that are being displayed on item pages.
Property:	`webui.preview.enabled`
Example Value:	`webui.preview.enabled = false`
Informational Note:	Whether or not the user can "preview" the image.
Property:	`webui.preview.maxwidth`
Example Value:	`webui.preview.maxwidth = 600`
Informational Note:	This property sets the maximum width for the preview image.
Property:	`webui.preview.maxheight`
Example Value:	`webui.preview.maxheight = 600`
Informational Note:	This property sets the maximum height for the preview image.
Property:	`webui.preview.brand`
Example Value:	`webui.preview.brand = My Institution Name`
Informational Note:	This is the brand text that will appear with the image.
Property:	`webui.preview.brand.abbrev`
Example Value:	`webui.preview.brand.abbrev = MyOrg`
Informational Note:	An abbreviated form of the full Branded Name. This will be used when the preview image cannot fit the normal text.
Property:	`webui.preview.brand.height`
Example Value:	`webui.preview.brand.height = 20`
Informational Note:	The height (in px) of the brand.
Property:	`webui.preview.brand.font`
Example Value:	`webui.preview.brand.font = SansSerif`
Informational Note:	This property sets the font for your Brand text that appears with the image.
Property:	`webui.preview.brand.fontpoint`
Example Value:	`webui.preview.brand.fontpoint = 12`
Informational Note:	This property sets the font point (size) for your Brand text that appears with the image.
Property:	`webui.preview.dc`
Example Value:	`webui.preview.dc = rights`
Informational Note:	The Dublin Core field that will display along with the preview. This field is optional.
Property:	`webui.strengths.show`
Example Value:	`webui.strengths.show = false`

Informational Note:	Determines if communities and collections should display item counts when listed. The default behavior if omitted, is false.
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 either your Messages.properties (JSPUI) or messages.xml (XMLUI) with new message keys referencing this <index-name>. JSPUI Example (Messages.properties): • `browse.type.metadata.<index-name> = My New Field` XMLUI Example (messages.xml): • `<message key="xmlui.ArtifactBrowser.Navigation.browse_<index-name>">My New Fields</message>` • `<message key="xmlui.ArtifactBrowser.ConfigurableBrowse.title.metadata.<index-name>">Browsing {0} by My New Field {1}</message>` • `<message key="xmlui.ArtifactBrowser.ConfigurableBrowse.trail.metadata.<index-name>">Browsing {0} by My New Field</message>` • `<message key="xmlui.ArtifactBrowser.ConfigurableBrowse.<index-name>.column_heading">My New Field</message>`

`(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. The default is dc (for Dublin Core).
`<element>`	(Only for "metadata" indexes) 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>`	(Only for "metadata" indexes) This is the qualifier to the <element> component. The user has two choices: an asterisk "**" or a proper qualifier of the element. The asterisk is a wildcard and causes DSpace to index all types of the schema element. For example, if you have the element "contributor" and the qualifier "**" then you would index all contributor data regardless of the qualifier. Another example, you have the element "subject" and the qualifier "lcsh" would cause the indexing of only those fields that have the qualifier "lcsh". (This means you would only index Library of Congress Subject Headings and not all data elements that are subjects.
`<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

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

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)

Author (Multiple metadata value) Display

This section actually applies to any field with multiple values, but authors are the define case and example here.

Property:	`webui.browse.author-field`
Example Value:	`webui.browse.author-field = dc.contributor.*`
Informational Note:	This defines which field is the author/editor, etc. listing.

Replace `dc.contributor.*` with another field if appropriate. The field should be listed in the configuration for `webui.itemlist.columns`, otherwise you will not see its effect. It must also be defined in `webui.itemlist.columns` as being of the datatype *text* otherwise the functionality will be overridden by the specific data type feature. (This setting is not used by the XMLUI as it is controlled by your theme).

Now that we know which field is our author or other multiple metadata value field we can provide the option to truncate the number of values displayed by default. We replace the remaining list of values with "et al" or the language pack specific alternative. Note that this is just for the default, and users will have the option of changing the number displayed when they browse the results. See the following table:

Property:	`webui.browse.author-limit`
Example Value:	`webui.browse.author-limit = <` *n* `>`
Informational Note:	Where < *n* > is an integer number of values to be displayed. Use `-1` for unlimited (the default value).

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.

Recent Submissions

> Since DSpace 4.0 this applies only to JSPUI. XMLUI uses Discovery to configure the recent submissions.

This allows us to define which index to base Recent Submission display on, and how many we should show at any one time. This uses the PluginManager to automatically load the relevant plugin for the Community and Collection home pages. Values given in examples are the defaults supplied in *dspace.cfg*

Property:	`recent.submission.sort-option`
Example Value:	`recent.submission.sort-option = dateaccessioned`
Informational Note:	Define the sort name (from *webui.browse.sort-options*) to use for displaying recent submissions. (Only used by JSPUI)
Property:	`recent.submissions.count`
Example Value:	`recent.submissions.count = 5`
Informational Note:	Defines how many recent submissions should be displayed at any one time. (Only used by JSPUI)

There will be the need to set up the processors that the PluginManager will load to actually perform the recent submissions query on the relevant pages. This is already configured by default *dspace.cfg* so there should be no need for the administrator/programmer to worry about this.

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

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

Submission License Substitution Variables

Property:	`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

This will enable syndication feeds, links display on community and collection home pages. This setting is not used by the XMLUI, as you enable feeds in your theme.

Property:	`webui.feed.enable`
Example Value:	`webui.feed.enable = true`
Informational Note:	By default, RSS feeds are set to true (on) . Change key to "false" to disable.
Property:	`webui.feed.items`

Example Value:	`webui.feed.items = 4`
Informational Note:	Defines the number of DSpace items per feed (the most recent submissions)
Property:	`webui.feed.cache.size`
Example Value:	`webui.feed.cache.size = 100`
Informational Note:	Defines the maximum number of feeds in memory cache. Value of "`0`" will disable caching.
Property:	`webui.feed.cache.age`
Example Value:	`webui.feed.cache.age = 48`
Informational Note:	Defines the number of hours to keep cached feeds before checking currency. The value of "*0*" will force a check with each request.
Property:	`webui.feed.formats`
Example Value:	`webui.feed.formats = rss_1.0,rss_2.0,atom_1.0`
Informational Note:	Defines which syndication formats to offer. You can use more than one; use a comma-separated list. The following list are the available values: rss_0.90, rss_0.91, rss_0.92, rss_0.93, rss_0.94, rss_1.0, rss_2.0, atom_1.0.
Property:	`webui.feed.localresolve`
Example Value:	`webui.feed.localresolve = false`
Informational Note:	By default, (set to false), URLs returned by the feed will point at the global handle resolver (e.g. http://hdl.handle.net/123456789/1). If set to ***true*** the local server URLs are used (e.g. http://myserver.myorg/handle/123456789/1).
Property:	`webui.feed.item.title`
Example Value:	`webui.feed.item.title = dc.title`
Informational Note:	This property customizes each single-value field displayed in the feed information for each item. Each of the fields takes a **single** metadata field. The form of the key is <scheme prefix>.<element>.<qualifier> In place of the qualifier, one may leave it blank to exclude any qualifiers or use the wildcard "*" to include all qualifiers for a particular element.
Property:	`webui.feed.item.date`
Example Value:	`webui.feed.item.date = dc.date.issued`
Informational Note:	This property customizes each single-value field displayed in the feed information for each item. Each of the fields takes a **single** metadata field. The form of the key is <scheme prefix>.<element>.<qualifier> In place of the qualifier, one may leave it blank to exclude any qualifiers or use the wildcard "*" to include all qualifiers for a particular element.
Property:	`webui.feed.item.description`
Example Value:	`webui.feed.item.description = dc.title, dc.contributor.author, \ dc.contributor.editor, dc.description.abstract, \ dc.description`

Informational Note:	One can customize the metadata fields to show in the feed for each item's description. Elements are displayed in the order they are specified in *dspace.cfg*.Like other property keys, the format of this property key is: *webui.feed.item.description = <scheme prefix>.<element>.<qualifier>*. In place of the qualifier, one may leave it blank to exclude any qualifiers or use the wildcard "*" to include all qualifiers for a particular element.
Property:	`webui.feed.item.author`
Example Value:	`webui.feed.item.author = dc.contributor.author`
Informational Note:	The name of field to use for authors (Atom only); repeatable.
Property:	`webui.feed.logo.url`
Example Value:	`webui.feed.logo.url = ${dspace.url}/themes/mysite/images/mysite-logo.png`
Informational Note:	Customize the image icon included with the site-wide feeds. This must be an absolute URL.
Property:	`webui.feed.item.dc.creator`
Example Value:	`webui.feed.item.dc.creator = dc.contributor.author`
Informational Note:	This optional property adds *structured* DC elements as XML elements to the feed description. They are not the same thing as, for example, *webui.feed.item.description*. Useful when a program or stylesheet will be transforming a feed and wants separate author, description, date, etc.
Property:	`webui.feed.item.dc.date`
Example Value:	`webui.feed.item.dc.date = dc.date.issued`
Informational Note:	This optional property adds *structured* DC elements as XML elements to the feed description. They are not the same thing as, for example, *webui.feed.item.description*. Useful when a program or stylesheet will be transforming a feed and wants separate author, description, date, etc.
Property:	`webui.feed.item.dc.description`
Example Value:	`webui.feed.item.dc.description = dc.description.abstract`
Informational Note:	This optional property adds *structured* DC elements as XML elements to the feed description. They are not the same thing as, for example, *webui.feed.item.description*. Useful when a program or stylesheet will be transforming a feed and wants separate author, description, date, etc.
Property:	`webui.feed.podcast.collections`
Example Value:	`webui.feed.podcast.collections = 1811/45183,1811/47223`
Informational Note:	This optional property enables Podcast Support on the RSS feed for the specified collection handles. The podcast is iTunes compatible and will expose the bitstreams in the items for viewing and download by the podcast reader. Multiple values are separated by commas. For more on using/enabling Media RSS Feeds to share content via iTunesU, see: Enable Media RSS Feeds
Property:	`webui.feed.podcast.communities`
Example Value:	`webui.feed.podcast.communities = 1811/47223`
Informational Note:	This optional property enables Podcast Support on the RSS feed for the specified community handles. The podcast is iTunes compatible and will expose the bitstreams in the items for viewing and download by the podcast reader. Multiple values are separated by commas. For more on using/enabling Media RSS Feeds to share content via iTunesU, see: Enable Media RSS Feeds

Property:	`webui.feed.podcast.mimetypes`
Example Value:	`webui.feed.podcast.mimetypes = audio/x-mpeg, application/pdf`
Informational Note:	This optional property for Podcast Support, allows you to choose which MIME types of bitstreams are to be enclosed in the podcast feed. Multiple values are separated by commas. For more on using /enabling Media RSS Feeds to share content via iTunesU, see: Enable Media RSS Feeds
Property:	`webui.feed.podcast.sourceuri`
Example Value:	`webui.feed.podcast.sourceuri = dc.source.uri`
Informational Note:	This optional property for the Podcast Support will allow you to use a value for a metadata field as a replacement for actual bitstreams to be enclosed in the RSS feed. A use case for specifying the external sourceuri would be if you have a non-DSpace media streaming server that has a copy of your media file that you would prefer to have the media streamed from. For more on using /enabling Media RSS Feeds to share content via iTunesU, see: Enable Media RSS Feeds

OpenSearch Support

OpenSearch is a small set of conventions and documents for describing and using "search engines", meaning any service that returns a set of results for a query. See extensive description in the *Business Layer section* of the documentation.

Please note that for result data formatting, OpenSearch uses Syndication Feed Settings (RSS). So, even if Syndication Feeds **are not** enable, they **must** be configured to enable OpenSearch. OpenSearch uses all the configuration properties for DSpace RSS to determine the mapping of metadata fields to feed fields. Note that a new field for authors has been added (used in Atom format only).

Property:	`websvc.opensearch.enable`
Example Value:	`websvc.opensearch.enable = false`
Informational Note:	Whether or not OpenSearch is enabled. By default, the feature is disabled. Change the property key to "true" to enable.
Property:	`websvc.opensearch.uicontext`
Example Value:	`websvc.opensearch.uicontext = simple-search`
Informational Note:	Context for HTML request URLs. Change only for non-standard servlet mapping. **IMPORTANT**: If you are using XMLUI and have Discovery enabled, this property's value should be changed to *discover*.
Property:	`websvc.opensearch.svccontext`
Example Value:	`websvc.opensearch.svccontext = open-search/`
Informational Note:	Context for RSS/Atom request URLs. Change only for non-standard servlet mapping. **IMPORTANT**: If you are using XMLUI and have Discovery enabled, this property's value should be changed to open-search/*discover*.
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

The following configuration is used to change the disposition behavior of the browser. That is, when the browser will attempt to open the file or download it to the user-specified location. For example, the default size is 8MB. When an item being viewed is larger than 8MB, the browser will download the file to the desktop (or wherever you have it set to download) and the user will have to open it manually.

Property:	`webui.content_disposition_threshold`
Example value:	`webui.content_disposition_threshold = 8388608`
Informational Note:	The default value is set to 8MB. This property key applies to the JSPUI interface.
Property:	`xmlui.content_disposition_threshold`
Example Value:	`xmlui.content_disposition_threshold = 8388608`
Informational Note:	The default value is set to 8MB. This property key applies to the XMLUI (Manakin) interface.

Other values are possible:
4 MB = 41943048 MB = 838860816 MB = 16777216

Multi-file HTML Document/Site Settings

The setting is used to configure the "depth" of request for html documents bearing the same name.

Property:	`webui.html.max-depth-guess`
Example Value:	`webui.html.max-depth-guess = 3`
Informational Note:	When serving up composite HTML items in the JSP UI, how deep can the request be for us to serve up a file with the same name? For example, if one receives a request for "*foo/bar/index.html*" and one has a bitstream called just "*index.html*", DSpace will serve up the former bitstream (*foo/bar/index.html*) for the request if *webui.html.max-depth-guess* is 2 or greater. If *webui.html.max-depth-guess* is 1 or less, then DSpace would not serve that bitstream, as the depth of the file is greater. If webui.html.max-depth-guess is zero, the request filename and path must always exactly match the bitstream name. The default is set to 3.
Property:	`xmlui.html.max-depth-guess`
Example Value:	`xmlui.html.max-depth-guess = 3`
Informational Note:	When serving up composite HTML items in the XMLUI, how deep can the request be for us to serve up a file with the same name? For example, if one receives a request for "*foo/bar/index.html*" and one has a bitstream called just "*index.html*", DSpace will serve up the former bitstream (*foo/bar/index.html*) for the request if *webui.html.max-depth-guess* is 2 or greater. If *xmlui.html.max-depth-guess* is 1 or less, then DSpace would not serve that bitstream, as the depth of the file is greater. If _webui.html.max-depth-guess _is zero, the request filename and path must always exactly match the bitstream name. The default is set to 3.

Sitemap Settings

To aid web crawlers index the content within your repository, you can make use of sitemaps.

Property:	`sitemap.dir`
Example Value:	`sitemap.dir = ${dspace.dir}/sitemaps`
Informational Note:	The directory where the generate sitemaps are stored.
Property:	`sitemap.engineurls`
Example Value:	sitemap.engineurls = http://www.google.com/webmasters/sitemaps/ping?sitemap=
Informational Note:	Comma-separated list of search engine URLs to "ping" when a new Sitemap has been created. 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.

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.LCNameAuthority = LCNameAuthority, \ org.dspace.content.authority.SHERPARoMEOPublisher = SRPublisher, \ org.dspace.content.authority.SHERPARoMEOJournalTitle = SRJournalTitle`
Informational Note:	--
Property:	`plugin.selfnamed.org.dspace.content.authority.ChoiceAuthority`
Example Value:	`plugin.selfnamed.org.dspace.content.authority.ChoiceAuthority = \ org.dspace.content.authority.DCInputAuthority`
Property:	`lcname.url`
Example Value:	`lcname.url = http://alcme.oclc.org/srw/search/lcnaf_`
Informational Note:	Location (URL) of the Library of Congress Name Service
Property:	`sherpa.romeo.url / 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/v2.1`
Informational Note:	Location (URL) of the ORCID v2 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.

Property:	`xmlui.lookup.select.size`
Example Value:	`xmlui.lookup.select.size = 12`
Informational Note:	This property sets the number of selectable choices in the Choices lookup popup

Configuring Multilingual Support

[i18n – Locales]

Setting the Default Language for the Application

Property:	`default.locale`
Example Value:	`default.locale = en`
Informational Note:	The default language for the application is set with this property key. This is a locale according to i18n and might consist of country, country_language or country_language_variant. If no default locale is defined, then the server default locale will be used. The format of a local specifier is described here: http://java.sun.com/j2se/1.4.2/docs/api/java/util/Locale.html

Supporting More Than One Language

Changes in dspace.cfg

Property:	`webui.supported.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.

The table above, if needed and is used will result in:

- a language switch in the default header
- the user will be enabled to choose his/her preferred language, this will be part of his/her profile
- wording of emails
  - mails to registered users, e.g. alerting service will use the preferred language of the user
  - mails to unregistered users, e.g. suggest an item will use the language of the session
- according to the language selected for the session, using dspace-admin Edit News will edit the news file of the language according to session

Related Files

If you set webui.supported.locales make sure that all the related additional files for each language are available. *LOCALE* should correspond to the locale set in *webui.supported.locales*, e. g.: for webui.supported.locales = en, de, fr, there should be:

- `[dspace-source]/dspace/modules/jspui/src/main/resources/Messages.properties`
- `[dspace-source]/dspace/modules/jspui/src/main/resources/Messages_en.properties`
- `[dspace-source]/dspace/modules/jspui/src/main/resources/Messages_de.properties`
- `[dspace-source]/dspace/modules/jspui/src/main/resources/Messages_fr.properties`

Files to be localized:

- `[dspace-source]/dspace/modules/jspui/src/main/resources/Messages_LOCALE.properties`
- `[dspace-source]/dspace/config/input-forms_LOCALE.xml`
- `[dspace-source]/dspace/config/default_LOCALE.license - should be pure ASCII`
- `[dspace-source]/dspace/config/news-top_LOCALE.html`
- `[dspace-source]/dspace/config/news-side_LOCALE.html`
- `[dspace-source]/dspace/config/emails/change_password_LOCALE`
- `[dspace-source]/dspace/config/emails/feedback_LOCALE`
- `[dspace-source]/dspace/config/emails/internal_error_LOCALE`
- `[dspace-source]/dspace/config/emails/register_LOCALE`
- `[dspace-source]/dspace/config/emails/submit_archive_LOCALE`
- `[dspace-source]/dspace/config/emails/submit_reject_LOCALE`

- [dspace-source]/dspace/config/emails/submit_task_LOCALE
- [dspace-source]/dspace/config/emails/subscription_LOCALE
- [dspace-source]/dspace/config/emails/suggest_LOCALE
- [dspace]/webapps/jspui/help/collection-admin_LOCALE.html - in html keep the jump link as original; must be copied to [dspace-source]/dspace/modules/jspui/src/main/webapp/help
- [dspace]/webapps/jspui/help/index_LOCALE.html - must be copied to [dspace-source]/dspace/modules/jspui/src/main/webapp/help
- [dspace]/webapps/jspui/help/site-admin_LOCALE.html - must be copied to [dspace-source]/dspace/modules/jspui/src/main/webapp/help

JSPUI Upload File Settings

To alter these properties for the XMLUI, please consult the Cocoon specific configuration at /WEB-INF/cocoon/properties/core.properties.

Property:	`upload.temp.dir`
Example Value:	`upload.temp.dir = ${dspace.dir}/upload`
Informational Note:	This property sets where DSpace temporarily stores uploaded files.
Property:	`upload.max`
Example Value:	`upload.max = 536870912`
Informational Note:	Maximum size of uploaded files in bytes. A negative setting will result in no limit being set. The default is set for 512Mb.

JSP Web Interface (JSPUI) Settings

The following section is limited to JSPUI. If the user wishes to use XMLUI settings, please refer to Chapter 7: XMLUI Configuration and Customization.

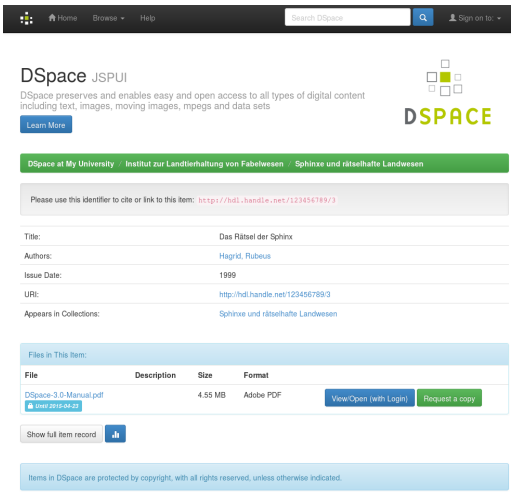
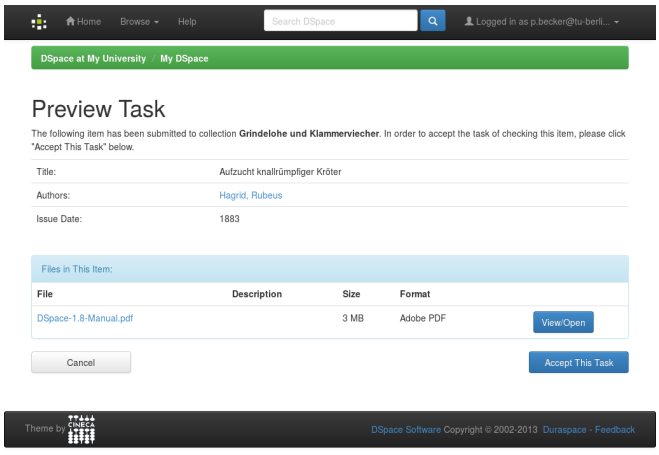
Property:	`webui.itemdisplay.default`
Example Value:	`webui.itemdisplay.default = dc.title, dc.title.alternative, \ dc.contributor.*, dc.subject, dc.data.issued (date), \ dc.publisher, dc.identifier.citation, \ dc.relation.ispartofseries, dc.description.abstract, \ dc.description, dc.identifier.govdoc, \ dc.identifier.uri (link), dc.identifier.isbn, \ dc.identifier.issn, dc.identifier.ismn, dc.identifier`
Informational Note:	This is used to customize the DC metadata fields that display in the item display (the brief display) when pulling up a record. The format is: `<schema>.<element>.<_optional_qualifier>` . In place of the qualifier, one can use the wildcard "*" to include all fields of the same element, or, leave it blank for unqualified elements. Additionally, two additional options are available for behavior /rendering: (date) and (link). See the following examples:

	`dc.title` = Dublin Core element "title" (unqualified)
	`dc.title.alternative` = DC element "title", qualifier "alternative"
	`dc.title.*` = All fields with Dublin Core element 'title' (any or no qualifier)
	`dc.identifier.uri(link)` = DC identifier.uri, rendered as a link
	`dc.date.issued(date)` = DC date.issued, rendered as a date
	The `Messages.properties` file controls how the fields defined above will display to the user. If the field is missing from the *Messages.properties* file, it will not be displayed. Look in `Messages.properties` under the `metadata.dc.<field>`. Example:
	`metadata.dc.contributor.other = Authors`
	`metadata.dc.contributor.author = Authors`
	`metadata.dc.title.* = Title`
	Please note: The order in which you place the values to the property key control the order in which they will display to the user on the outside world. (See the Example Value above).

Property:	`webui.resolver.1.urn`
	`webui.resolver.1.baseurl`
	`webui.resolver.2.urn`
	`webui.resolver.2.baseurl`

Example Value:	`webui.resolver.1.urn = doi`
	`webui.resolver.1.baseurl = http://dx.doi.org/`
	`webui.resolver.2.urn = hdl`
	`webui.resolver.2.baseurl = http://hdl.handle.net/`

Informational Note:	When using "resolver" in *webui.itemdisplay* to render identifiers as resolvable links, the base URL is taken from <code>webui.resolver.<n>.baseurl<code> where <code>webui.resolver.<n>.baseurl<code> matches the urn specified in the metadata value. The value is appended to the "baseurl" as is, so the baseurl needs to end with the forward slash almost in any case. If no urn is specified in the value it will be displayed as simple text. For the doi and hdl urn defaults values are provided, respectively http://dc.doi.org and http://hdl.handle.net are used. If a metadata value with style "doi", "handle" or "resolver" matches a URL already, it is simply rendered as a link with no other manipulation.
Property:	webui.preferred.identifier
Example Value:	webui.preferred.identifier = handle
Informational Note:	At the top of the item view a persistent identifier is shown to be used to refer to this item. If you use Item Level Versioning and DSpace is configured to, it shows a version history. Per default DSpace uses handle as preferred identifier. If you've configured DSpace to register DOIs you can decide to use DOIs instead of handles at the top of the item view and within the version history. Set the property webui.preferred.identifier = doi to do so.
Property:	webui.identifier.strip-prefixes
Example Value:	webui.identifier.strip-prefixes = true
Informational Note:	In the version history Persistent Identifiers can be shown with or without their prefixes, e.g. a handle can be shown as handle:10673

	/6 or just as 10673/6. A DOI can be can be shown as 10.5072 /example-doi-123 or as doi:105072/example-doi-123. This property controlls whether the handles are stripped (default) or not.
Property:	`plugin.single.org.dspace.app.webui.util. StyleSelection`
Example Value:	`plugin.single.org.dspace.app. webui.util.StyleSelection = \ org.dspace.app.web.util. CollectionStyleSelection #org.dspace.app.web.util. MetadataStyleSelection`
Informational Note:	Specify which strategy to use for select the style for an item.
Property:	`webui.itemdisplay.thesis.collections`
Example Value:	`webui.itemdisplay.thesis.collections = 123456789 /24, 123456789/35`
Informational Note:	Specify which collections use which views by Handle.
Property:	`webui.itemdisplay.label.restricted.bitstreams`
Example Value:	`webui.itemdisplay.label.restricted.bitstreams = true`
Informational Note:	  If set to all, all users will get a warning if access restrictions are in place fc

	Should access restricted bitstreams be labeled as such? If set true, all bitstreams which cannot currently not be read by an anonymous user are labeled as being access restricted. If a resource policy to allow read access for anonymous users with an unreached start date exists, this date is shown as well.
Property:	`webui.itemdisplay.metadata-style` `webui.itemdisplay.metadata-style`
Example Value:	`webui.itemdisplay.metadata-style = schema.element[.qualifier|.*]` `webui.itemdisplay.metadata-style = dc.type`
Informational Note:	Specify which metadata to use as name of the style
Property:	`webui.itemlist.columns`
Example Value:	`webui.itemlist.columns = thumbnail, dc.date.issued (date), dc.title, \` `dc.contributor.*`
Informational Note:	Customize the DC fields to use in the item listing page. Elements will be displayed left to right in the order they are specified here. The form is <schema prefix>.<element>[.<qualifier> | .*][(date)], ... Although not a requirement, it would make sense to include among the listed fields at least the date and title fields as specified by the we `bui.browse.index` configuration options in the next section mentioned. (cf.) If you have enabled thumbnails (webui.browse.thumbnail.show), you must also include a 'thumbnail' entry in your columns, this is where the thumbnail will be displayed.
Property:	`webui.itemlist.width`
Example Value:	`webui.itemlist.width = *, 130, 60%, 40%`
Informational Note:	You can customize the width of each column with the following line--you can have numbers (pixels) or percentages. For the 'thumbnail' column, a setting of '*' will use the max width specified for browse thumbnails (cf. `webui.browse.thumbnail.maxwidth, thumbnail.maxwidth`)

Property:	`webui.itemlist.browse.<index name>.sort.<sort name>.columns`
	`webui.itemlist.sort.<sort name>.columns`
	`webui.itemlist.browse.<browse name>.columns`

	`webui.itemlist.<sort or index name>.columns`
Example Value:	
Informational Note:	You can override the DC fields used on the listing page for a given browse index and/or sort option. As a sort option or index may be defined on a field that isn't normally included in the list, this allows you to display the fields that have been indexed/sorted on. There are a number of forms the configuration can take, and the order in which they are listed below is the priority in which they will be used (so a combination of an index name and sort name will take precedence over just the browse name).In the last case, a sort option name will always take precedence over a browse index name. Note also, that for any additional columns you list, you will need to ensure there is an *itemlist.<field name>* entry in the messages file.
Property:	`webui.itemlist.dateaccessioned.columns`
Example Value:	`webui.itemlist.dateaccessioned.columns = thumbnail, dc.date.accessioned(date), dc.title, dc.contributor.*`
Informational Note:	This would display the date of the accession in place of the issue date whenever the dateaccessioned browsed index or sort option is selected. Just like *webui.itemlist.columns*, you will need to include a 'thumbnail' entry to display the thumbnails in the item list.
Property:	`webui.itemlist.dateaccessioned.widths`
Example Value:	`webui.itemlist.dateaccessioned.widths = *, 130, 60%, 40%`
Informational Note:	As in the aforementioned property key, you can customize the width of the columns for each configured column list, substituting "*.widths"* for "*.columns"* in the property name. See the setting for *webui.itemlist.widths* for more information.
Property:	`webui.itemlist.tablewidth`
Example Value:	`webui.itemlist.tablewidth = 100%`
Informational Note:	You can also set the overall size of the item list table with the following setting. It can lead to faster table rendering when used with the column widths above, but not generally recommended.
Property:	`webui.session.invalidate`
Example Value:	`webui.session.invalidate = true`
Informational Note:	Enable or disable session invalidation upon login or logout. This feature is enabled by default to help prevent session hijacking but may cause problems for shibboleth, etc. If omitted, the default value is "*true*". [Only used for JSPUI authentication].
Property:	`jspui.google.analytics.key`
Example Value:	`jspui.google.analytics.key = UA-XXXXXX-X`
Informational Note:	If you would like to use Google Analytics to track general website statistics then use the following parameter to provide your Analytics key.

JSPUI Item Mapper

Because the item mapper requires a primitive implementation of the browse system to be present, we simply need to tell that system which of our indexes defines the author browse (or equivalent) so that the mapper can list authors' items for mapping

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

Property:	`itemmap.author.index`
Example Value:	`itemmap.author.index = author`
Informational Note:	If you change the name of your author browse field, you will also need to update this property key.

Display of Group Membership

Property:	`webui.mydspace.showgroupmembership`
Example Value:	`webui.mydspace.showgroupmembership = false`
Informational Note:	To display group membership set to "true". If omitted, the default behavior is false.

JSPUI / XMLUI SFX Server

SFX Server is an OpenURL Resolver.

Property:	`sfx.server.url`
Example Value:	`sfx.server.url` = `http://sfx.myu.edu:8888/sfx?`
	`sfx.server.url` = `http://worldcatlibraries.org/registry/gateway?`
Informational Note:	SFX query is appended to this URL. If this property is commented out or omitted, SFX support is switched off.

All the parameters mapping are defined in `[dspace]/config/sfx.xml` file. The program will check the parameters in `sfx.xml` and retrieve the correct metadata of the item. It will then parse the string to your resolver.

For the following example, the program will search the first query-pair which is DOI of the item. If there is a DOI for that item, your retrieval results will be, for example:
`http://researchspace.auckland.ac.nz/handle/2292/5763`

Example. For setting DOI in `sfx.xml`

```
<query-pairs>
    <field>
        <querystring>rft_id=info:doi/</querystring>
        <dc-schema>dc</dc-schema>
        <dc-element>identifier</dc-element>
        <dc-qualifier>doi</dc-qualifier>
    </field>
</query-pairs>
```

If there is no DOI for that item, it will search next query-pair based on the `[dspace]/config/sfx.xml` and then so on.

Example of using ISSN, volume, issue for item without DOI
`[http://researchspace.auckland.ac.nz/handle/2292/4947]`

For parameter passing to the `<querystring>`

```
<querystring>rft_id=info:doi/</querystring>
```

Please refer to these:
[http://ocoins.info/cobgbook.html]
[http://ocoins.info/cobg.html]

Program assume won't get empty string for the item, as there will at least author, title for the item to pass to the resolver.

For contributor author, program maintains original DSpace SFX function of extracting author's first and last name.

```
<field>
    <querystring>rft.aulast=</querystring>
    <dc-schema>dc</dc-schema>
    <dc-element>contributor</dc-element>
    <dc-qualifier>author</dc-qualifier>
</field>
<field>
    <querystring>rft.aufirst=</querystring>
    <dc-schema>dc</dc-schema>
    <dc-element>contributor</dc-element>
    <dc-qualifier>author</dc-qualifier>
</field>
```

JSPUI Item Recommendation Setting

Property:	`webui.suggest.enable`
Example Value:	webui.suggest.enable = true
Informational Note:	Show a link to the item recommendation page from item display page.
Property:	`webui.suggest.loggedinusers.only`
Example Value:	`webui.suggest.loggedinusers.only = true`
Informational Note:	Enable only if the user is logged in. If this key commented out, the default value is false.

Controlled Vocabulary Settings

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

Property:	`webui.controlledvocabulary.enable`
Example Value:	`webui.controlledvocabulary.enable = true`
Informational Note:	Enable or disable the controlled vocabulary add-on. WARNING: This feature is not compatible with WAI (it requires JavaScript to function).

The need for a limited set of keywords is important since it eliminates the ambiguity of a free description system, consequently simplifying the task of finding specific items of information.

The controlled vocabulary add-on allows the user to choose from a defined set of keywords organized in an tree (taxonomy) and then use these keywords to describe items while they are being submitted.

We have also developed a small search engine that displays the classification tree (or taxonomy) allowing the user to select the branches that best describe the information that he/she seeks.

The taxonomies are described in XML following this (very simple) structure:

```
<node id="acmccs98" label="ACMCCS98">
    <isComposedBy>
        <node id="A." label="General Literature">
            <isComposedBy>
                <node id="A.0" label="GENERAL"/>
                <node id="A.1" label="INTRODUCTORY AND SURVEY"/>
            </isComposedBy>
        </node>
    </isComposedBy>
</node>
```

You are free to use any application you want to create your controlled vocabularies. A simple text editor should be enough for small projects. Bigger projects will require more complex tools. You may use Protegé to create your taxonomies, save them as OWL and then use a XML Stylesheet (XSLT) to transform your documents to the appropriate format. Future enhancements to this add-on should make it compatible with standard schemas such as OWL or RDF.

In order to make DSpace compatible with WAI 2.0, the add-on is **turned off** by default (the add-on relies strongly on JavaScript to function). It can be activated by setting the following property in `dspace.cfg`:

```
webui.controlledvocabulary.enable = true
```

New vocabularies should be placed in `[dspace]/config/controlled-vocabularies/` and must be according to the structure described. A validation XML Schema (named `controlledvocabulary.xsd`) is also available in that directory.

Vocabularies need to be associated with the correspondent DC metadata fields. Edit the file `[dspace]/config/input-forms.xml` and place a *"vocabulary"* tag under the *"field"* element that you want to control. Set value of the *"vocabulary"* element to the name of the file that contains the vocabulary, leaving out the extension (the add-on will only load files with extension "*.xml"). For example:

```
<field>
    <dc-schema>dc</dc-schema>
    <dc-element>subject</dc-element>
    <dc-qualifier></dc-qualifier>
    <!-- An input-type of twobox MUST be marked as repeatable -->
    <repeatable>true</repeatable>
    <label>Subject Keywords</label>
    <input-type>twobox</input-type>
    <hint> Enter appropriate subject keywords or phrases below. </hint>
    <required></required>
    <vocabulary [closed="false"]>nsi</vocabulary>
</field>
```

The vocabulary element has an optional boolean attribute **closed** that can be used to force input only with the JavaScript of controlled-vocabulary add-on. The default behavior (i.e. without this attribute) is as set **closed="false"**. This allow the user also to enter the value in free way.

The following vocabularies are currently available by default:

- **nsi** - *nsi.xml* - The Norwegian Science Index
- **srsc** - *srsc.xml* - Swedish Research Subject Categories

3. JSPUI Session Invalidation

Property:	`webui.session.invalidate`
Example Value:	`webui.session.invalidate = true`
Informational Note:	Enable or disable session invalidation upon login or logout. This feature is enabled by default to help prevent session hijacking but may cause problems for shibboleth, etc. If omitted, the default value is 'true'.

XMLUI Specific Configuration

The DSpace digital repository supports two user interfaces: one based upon JSP technologies and the other based upon the Apache Cocoon framework. This section describes those configurations settings which are specific to the XMLUI interface based upon the Cocoon framework. *(Prior to DSpace Release 1.5.1 XMLUI was referred to Manakin. You may still see references to "Manakin")*

Property:	`xmlui.force.ssl`
Example Value:	`xmlui.force.ssl = true`
Informational Note:	Force all authenticated connections to use SSL, only non-authenticated connections are allowed over plain http. If set to true, then you need to ensure that the "*dspace.hostname*" parameter is set correctly.
Property:	`xmlui.user.registration`
Example Value:	`xmlui.user.registration = true`
Informational Note:	Determine if new users should be allowed to register. This parameter is useful in conjunction with Shibboleth where you want to disallow registration because Shibboleth will automatically register the user. Default value is true.
Property:	`xmlui.user.editmetadata`
Example Value:	`xmlui.user.editmetadata = true`
Informational Note:	Determines if users should be able to edit their own metadata. This parameter is useful in conjunction with Shibboleth where you want to disable the user's ability to edit their metadata because it came from Shibboleth. Default value is true.
Property:	xmlui.session.ipcheck
Example Value:	xmlui.session.ipcheck = true
Informational Note:	Check if the user has a consistent ip address from the start of the login process to the end of the login process. Disabling this check is not recommended unless absolutely necessary as the ip check can be helpful for preventing session hijacking. Possible reasons to set this to false: many-to-many wireless networks that prevent consistent ip addresses or complex proxying of requests. The default value is true.
Property:	`xmlui.user.loginredirect`
Example Value:	`xmlui.user.loginredirect = /profile`
Informational Note:	After a user has logged into the system, which url should they be directed? Leave this parameter blank or undefined to direct users to the homepage, or */profile* for the user's profile, or another reasonable choice is */submissions* to see if the user has any tasks awaiting their attention. The default is the repository home page.
Property:	`xmlui.theme.allowoverrides`
Example Value:	`xmlui.theme.allowoverrides = false`
Informational Note:	Allow the user to override which theme is used to display a particular page. When submitting a request add the HTTP parameter "themepath" which corresponds to a particular theme, that specified theme will be used instead of the any other configured theme. Note that this is a potential security hole allowing execution of unintended code on the server, this option is only for development and debugging it should be turned off for any production repository. The default value unless otherwise specified is "false".
Property:	xmlui.theme.enableConcatenation
Example Value:	xmlui.theme.enableConcatenation = false
Informational Note:	

	Enabling this property will concatenate CSS, JS and JSON files where possible. CSS files can be concatenated if multiple CSS files with the same media attribute are used in the same page. Links to the CSS files are automatically referring to the concatenated resulting CSS file. The theme sitemap should be updated to use the ConcatenationReader for all js, css and json files before enabling this property.
Property:	xmlui.theme.enableMinification
Example Value:	xmlui.theme.enableMinification = false
Informational Note:	Enabling this property will minify CSS, JS and JSON files where possible. The theme sitemap should be updated to use the ConcatenationReader for all js, css and json files before enabling this property.
Property:	xmlui.theme.mirage.item-list.emphasis
Example Value:	xmlui.theme.mirage.item-list.emphasis = file
Informational Note:	When set to "file" the item listings in your repository will include the generated thumbnails of uploaded files. Alternatively, you can set this parameter to metadata to put more emphasis on the metadata and effectively hide the thumbnails. The default value is "metadata".
Property:	mirage2.item-view.bitstream.href.label.1 mirage2.item-view.bitstream.href.label.2
Example Value:	mirage2.item-view.bitstream.href.label.1 = label mirage2.item-view.bitstream.href.label.2 = title
Informational Note:	**Mirage 2 theme ONLY** Determines if the bitstream filename (title) or description (label) is being used as the display label on the hyperlinks to download the actual files. By default, the file description (label) will be shown. If this value is empty, the filename (title) will be used as a fallback. More information and screenshots.
Property:	`xmlui.bundle.upload`
Example Value:	`xmlui.bundle.upload = ORIGINAL, METADATA, THUMBNAIL, LICENSE, CC_LICENSE`
Informational Note:	Determine which bundles administrators and collection administrators may upload into an existing item through the administrative interface. If the user does not have the appropriate privileges (add and write) on the bundle then that bundle will not be shown to the user as an option.
Property:	`xmlui.community-list.render.full`
Example Value:	`xmlui.community-list.render.full = true`
Informational Note:	On the community-list page should all the metadata about a community/collection be available to the theme. This parameter defaults to true, but if you are experiencing performance problems on the community-list page you should experiment with turning this option off.
Property:	`xmlui.community-list.cache`
Example Value:	`xmlui.community-list.cache = 12 hours`
Informational Note:	Normally, the XMLUI will fully verify any cache pages before using a cache copy. This means that when the community-list page is viewed the database is queried for each community/collection to see if their metadata has been modified. This can be expensive for repositories with a large community tree. To help solve this problem you can set the cache to be assumed valued for a specific set of time. The downside of this is that new or editing communities /collections may not show up the website for a period of time.
Property:	`xmlui.bitstream.mods`

Example Value:	`xmlui.bitstream.mods = true`
Informational Note:	Optionally, you may configure XMLUI to take advantage of metadata stored as a bitstream. The MODS metadata file must be inside the "METADATA" bundle and named MODS.xml. If this option is set to 'true' and the bitstream is present then it is made available to the theme for display.
Property:	`xmlui.bitstream.mets`
Example Value:	`xmlui.bitstream.mets = true`
Informational Note:	Optionally, you may configure Manakin to take advantage of metadata stored as a bitstream. The METS metadata file must be inside the "METADATA" bundle and named METS.xml. If this option is set to "true" and the bitstream is present then it is made available to the theme for display.
Property:	`xmlui.google.analytics.key`
Example Value:	`xmlui.google.analytics.key = UA-XXXXXX-X`
Informational Note:	If you would like to use Google Analytics to track general website statistics then use the following parameter to provide your analytics key. First sign up for an account at http://analytics.google.com, then create an entry for your repositories website. Google Analytics will give you a snippet of javascript code to place on your site, inside that snip it is your Google Analytics key usually found in the line: _uacct = "UA-XXXXXXX-X" Take this key (just the UA-XXXXXX-X part) and place it here in this parameter.
Property:	`xmlui.controlpanel.activity.max`
Example Value:	`xmlui.controlpanel.activity.max = 250`
Informational Note:	Assign how many page views will be recorded and displayed in the control panel's activity viewer. The activity tab allows an administrator to debug problems in a running DSpace by understanding who and how their dspace is currently being used. The default value is 250.
Property:	`xmlui.controlpanel.activity.ipheader`
Example Value:	`xmlui.controlpanel.activity.ipheader = X-Forward-For`
Informational Note:	Determine where the control panel's activity viewer receives an events IP address from. If your DSpace is in a load balanced environment or otherwise behind a context-switch then you will need to set the parameter to the HTTP parameter that records the original IP address.
Property:	`xmlui.search.metadata_export`
Example Value:	`xmlui.search.metadata_export = admin`
Informational Note:	Determine the access rights necessary to export DSpace metadata from search results in a CSV format (compatible with Batch Metadata Editing tool). By default, only Administrators can export metadata from search results. Other options include: • `admin` = Administrative users only • `user` = Any logged in user • `anonymous` = Anyone in the world

Optional or Advanced Configuration Settings

The following section explains how to configure either optional features or advanced features that are not necessary to make DSpace "out-of-the-box"

The Metadata Format and Bitstream Format Registries

The *[dspace]/config/registries* directory contains three XML files. These are used to load the *initial* contents of the Dublin Core Metadata registry and Bitstream Format registry and SWORD metadata registry. After the initial loading (performed by *ant fresh_install* above), the registries reside in the database; the XML files are not updated.

In order to change the registries, you may adjust the XML files before the first installation of DSpace. On an already running instance it is recommended to change bitstream registries via DSpace admin UI, but the metadata registries can be loaded again at any time from the XML files without difficult. The changes made via admin UI are not reflected in the XML files.

Metadata Format Registries

The default metadata schema is Dublin Core, so DSpace is distributed with a default Dublin Core Metadata Registry. Currently, the system requires that every item have a Dublin Core record.

There is a set of Dublin Core Elements, which is used by the system and should not be removed or moved to another schema, see Appendix: Default Dublin Core Metadata registry.

Note: altering a Metadata Registry has no effect on corresponding parts, e.g. item submission interface, item display, item import and vice versa. Every metadata element used in submission interface or item import must be registered before using it.

Note also that deleting a metadata element will delete all its corresponding values.

If you wish to add more metadata elements, you can do this in one of two ways. Via the DSpace admin UI you may define new metadata elements in the different available schemas. But you may also modify the XML file (or provide an additional one), and re-import the data as follows:

```
[dspace]/bin/dspace dsrun org.dspace.administer.MetadataImporter -f [xml
file]
```

The XML file should be structured as follows:

```
<dspace-dc-types>
    <dc-type>
        <schema>dc</schema>
        <element>contributor</element>
        <qualifier>advisor</qualifier>
        <scope_note>Use primarily for thesis advisor.</scope_note>
    </dc-type>
</dspace-dc-types>
```

Bitstream Format Registry

The bitstream formats recognized by the system and levels of support are similarly stored in the bitstream format registry. This can also be edited at install-time via *[dspace]/config/registries/bitstream-formats.xml* or by the administration Web UI. The contents of the bitstream format registry are entirely up to you, though the system requires that the following two formats are present:

- *Unknown*
- *License*
  Deleting a format will cause any existing bitstreams of this format to be reverted to the unknown bitstream format.

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 workflow systems to review submissions as described in detail as part of the architecture of the business logic layer. There is the original workflow systems and a configurable one. The file **[dspace]/config/modules/workflow.cfg** contains properties to configure details of the workflow systems.

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
```

Both workflow systems 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. This configuration works for the original workflow system as well as for the configurable xml workflow system.

JSPUI: Per item visual indicators for browse and search results

Visual indicators per item allow users to mark items in browse and search results. This could be useful in many scenarios, some of them follow:

1. If your repository contains items of different type (articles, book chapters, pictures) you can mark the type of each item using an icon.
2. If your repository has items with bitstreams but also has items with no bitstream, you could indicate this fact to the users using the visual indicators
3. If you have applied copyright licences in the bitstreams or items, you could notify users about that in the browse or result list
4. If you want your users to spot some items out of the list easily or if you want to differentiate some items from the others you could use the visual indicators

The visual indicators extension has the following specs:

1. Multiple marks can be added per item (i.e. mark the type of the item and the availability of the bitstreams)
2. Easy configuration of the strategy of what mark to display in every item
3. Marks based on images or a generic class (i.e. a glyphicon icon for bootstrap)
4. Display tooltip when hovering the mark + localization of the tooltip

5. Easy addtion of new strategies for any type of mark the user desires
6. Add css styles for the user to configure the position of the marks in the list row

Some theory:

A mark is an instance of the class: **org.dspace.app.itemmarking.ItemMarkingInfo**.

Each mark can have the following properties:

- *imageName*: a path to the image that will be displayed for the specific mark
- *classInfo*: the css class to be applied in the mark (useful if you do not want to add an image but just an icon from the bootstrap glyph icons)
- *link*: the link to be applied in the mark (optional)
- *tooltip*: the tooltip to be shown when hovering over the mark (optional)

When you need to add a mark in an Item then you need to create a strategy that determined what mark to display per item. Strategy classes need to implement the interface:

```
org.dspace.app.itemmarking.ItemMarkingExtractor
```

Your strategy class just needs to implement the following method from the above Interface:

```
public ItemMarkingInfo getItemMarkingInfo(Context context, Item item)
throws SQLException;
```

Which is, given an item, return the Mark info to display.

Currently, there are three Strategies included by default:

ItemMarkingMetadataStrategy

This strategy decides the mark to display per item based on a value of a metadata field (i.e. dc:type)

It accepts two properties:

- *metadataField*: the metadata field to be used for searching the value in the form “schema.element.qualifier”
- *mapping*: a Java Map of **String**s to **ItemMarkingInfo**s

If the String (key of the map) is found as a value in the metadataField field, then the mark denoted by the value of the map will be displayed.

ItemMarking Collection Strategy

This strategy decides the mark to display per item based on the collection this item belongs to.

It accepts one property:

- *mapping*: a Java Map of **String**s to **ItemMarkingInfo**s

The String (key of the map) is the collection handle (i.e.: 123456789/1) and if an items belongs to this collection, the mark denoted by the object of the map will be displayed

It emMarking AvailabilityBit Str eamStrategy

This strategy decides the mark to display per item based on the availability (exists or not) of a bitstream within the item.

It accepts to properties:

- *nonAvailableImageName*: the image to display for the mark if no bitstreams exist for the item
- *nonAvailableImageName*: the image to display for the mark if at least one bistream exist for the item

Moreover, this strategy add a link in the mark (in case there are bitstreams in the item) to the first bitstream of the item

How to:

In order to enable a mark for the result or browse list you need to change the option:

```
webui.itemlist.columns
```

of the **dspace.cfg** file.

You need to include a 'mark_[value]' key in any column order you like. Do not add the brackets and you can replace the "value" with any word has a meaning for your marking type. You may add multiple marks (i.e.: one in the first column and one at the last)

For example, the following line is a valid option value:

```
webui.itemlist.columns = mark_type, dc.date.issued(date), dc.title, dc.
contributor.*, mark_availability
```

In the aforementioned case, you just added two marks, one in the first column for the type of the item and one in the last item for the availability.

Now it's time to declare what "**mark_type**" and "**mark_availability**" means. This is done in the Spring configuration file **config/sping/api/item-marking.xml**, via the dependency injection feature.

In this file, for each "**mark_[value]**" key you add in the *dspace.cfg* file, you need to add a Spring bean with **id=org.dspace.app.itemmarking.ItemMarkingExtractor.[value].** The class of this bean must be an implementation of **org.dspace.app.itemmarking.ItemMarkingExtractor.**

That's all!

For our example, we need to declare two beans (one for **"mark_type"** and one for **"mark_availability"**).

```
<!-- Enable this strategy in order to mark item based on the value of a
metadata field -->
<bean class="org.dspace.app.itemmarking.ItemMarkingMetadataStrategy" id="
org.dspace.app.itemmarking.ItemMarkingExtractor.type">
     <property name="metadataField" value="dc.type" />
     <property name="mapping" ref="typeMap"/>
</bean>

<!-- Enable this strategy in order to mark items based on the availability
of their bitstreams -->
<bean class="org.dspace.app.itemmarking.
ItemMarkingAvailabilityBitstreamStrategy" id="org.dspace.app.itemmarking.
ItemMarkingExtractor.availability">
     <property name="availableImageName" value="image/available.png" />
     <property name="nonAvailableImageName" value="image/nonavailable.png"
/>
</bean>
```

For the "**mark_type**", we have declared the strategy to be **ItemMarkingMetadataStrategy** which means that the value of a metadata field (dc.type in our case) will determine the mark of each item. Here is the mapping:

```
<bean class="java.util.HashMap" id="typeMap">
       <constructor-arg>
```

```
            <map>
                     <entry>
                  <key>
                                              <value>image</value>
                  </key>
                  <ref bean="type1MarkingInfo"/>
               </entry>
               <entry>
                  <key>
                                            <value>video</value>
                  </key>
                  <ref bean="type2MarkingInfo"/>
                </entry>
             </map>
       </constructor-arg>
</bean>
```

Thus, if the value of **dc.type** field is equal to image the "**type1MarkingInfo**" bean will be used for the marking, if it is equal to video the "**type2MarkingInfo**" bean will be used, otherwise, no mark will be displayed.

```
<bean class="org.dspace.app.itemmarking.ItemMarkingInfo" id="
type1MarkingInfo">
       <property name="classInfo" value="glyphicon glyphicon-picture"/>
       <property name="tooltip" value="itemlist.mark.type1MarkingInfo"/>
</bean>
<bean class="org.dspace.app.itemmarking.ItemMarkingInfo" id="
type2MarkingInfo">
       <property name="imageName" value="image/type2.png"/>
       <property name="tooltip" value="itemlist.mark.type2MarkingInfo"/>
</bean>
```

Tooltip property contains the localized key to display.

Keep in mind that the Strategy that you may write can have its own logic on how to create the **ItemMarkingInfo** per item. The only requirement of the feature is to add in the Spring configuration file the initial beans one for each mark you have declared in the dspace.cfg file.

Styling:

The title for the column of each mark is titled based on the localized key "**itemlist.mark_[value]**", so you just need to add the specific keys in the messages.propertied files.

Moreover, the following CSS styles are applied to the various aspects of the mark:

- **mark_[value]_th**: a style applied to the column header
- **mark_[value]_tr**: a style applied to the each row

Add these classes to the css file and apply any style you like (like centering the text or the image)

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.

webui.itemdisplay.label.restricted.bitstreams

> **Many configuration names/keys have changed!**
>
> 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 compensate for this, 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 compatible with the Enhanced Configuration Scheme. 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 in the next section, you'll likely want to do that anyways in order to take full advantage of the new `local.cfg` file.

`[dspace]/config/config-definition.xml`

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.