GeoNetwork Developer Manual

Release 2.8.0

GeoNetwork

February 22, 2013

Contents

1 Software development **3**
- 1.1 System Requirements . . . 3
- 1.2 Tools . . . 4
- 1.3 Check out source code . . . 4
- 1.4 Build GeoNetwork . . . 5
- 1.5 Creating the installer . . . 6
- 1.6 Eclipse setup . . . 7

2 Create GeoNetwork releases **11**
- 2.1 Create a stable release for GeoNetwork . . . 11

3 Harvesting **17**
- 3.1 Structure . . . 17
- 3.2 Data storage . . . 19
- 3.3 Guidelines . . . 19

4 Schema Plugins **21**
- 4.1 Contents of a GeoNetwork schema . . . 21
- 4.2 Preparation . . . 23
- 4.3 Example - ISO19115/19139 Marine Community Profile (MCP) . . . 23

5 Metadata Exchange Format **51**
- 5.1 Introduction . . . 51
- 5.2 MEF v1 file format . . . 51
- 5.3 MEF v2 file format . . . 52
- 5.4 The info.xml file . . . 53

6 XML Services **57**
- 6.1 Calling specifications . . . 57
- 6.2 Login and Logout services . . . 60
- 6.3 Group services . . . 62
- 6.4 User services . . . 67
- 6.5 Category services . . . 79
- 6.6 Search and Retrieve Metadata services . . . 84

6.7 Metadata insert, update and delete services . . . 94
6.8 Metadata Select services . . . 101
6.9 Metadata Privilege services . . . 103
6.10 Metadata Ownership services . . . 106
6.11 Metadata Status services . . . 113
6.12 Metadata Category services . . . 118
6.13 Metadata Versioning services . . . 121
6.14 Metadata Processing services . . . 124
6.15 Metadata Relation services . . . 130
6.16 Metadata Validation services . . . 135
6.17 System configuration . . . 139
6.18 Site Information and Request Forwarding Services . . . 146
6.19 File download services . . . 154
6.20 Harvesting services . . . 157
6.21 Schema Services . . . 170
6.22 MEF services . . . 176
6.23 CSW service . . . 178
6.24 Java development with XML services . . . 186

7 Settings hierarchy **195**
7.1 Introduction . . . 195
7.2 System node . . . 195
7.3 Harvesting nodes . . . 197

Index **219**

Welcome to the GeoNetwork Developer Manual v2.8.0. The manual is for those who want to help with the development process, including source code, software releasing, and other administrative work.

Other documents:

GeoNetwork User Manual

GeoNetwork Developer Manual (PDF)

CHAPTER 1

Software development

1.1 System Requirements

GeoNetwork is a Java application that runs as a servlet so the Java Runtime Environment (JRE) must be installed in order to run it. You can get the JRE from http://www.oracle.com/technetwork/java/javase/downloads and you have to download the Java 6 Standard Edition (SE). GeoNetwork won't run with Java 1.4. It will run with:

- Java 5 - but few people should be using that now as it is unsupported.
- Java 6 - most testing has taken place under Java 6, so we recommend Java 6.
- OpenJDK 7 - note: problems have been reported with OpenJDK 6.

Being written in Java, GeoNetwork can run on any platform that supports Java, so it can run on Windows, Linux and Mac OSX. For MacOSX, make sure you use version 10.4 (Tiger) or newer. Version 10.3 (Panther) has only Java 1.4 so it cannot run GeoNetwork.

Next, you need a servlet container. GeoNetwork comes with an embedded container (Jetty) which is fast and well suited for most applications. If you need a stronger one, you can install Tomcat from the Apache Software Foundation (http://tomcat.apache.org). It provides load balancing, fault tolerance and other production features. If you work for an organisation, it is probable that you already use Tomcat. The tested version is 6.0.x but GeoNetwork should work with all other versions (>= 5.5).

Regarding storage, you need a Database Management System (DBMS) like Oracle, MySQL, Postgresql etc. GeoNetwork comes with an embedded DBMS (H2) which is used by default during installation. This DBMS can be used for small or desktop installations of no more than a few thousand metadata records with one or two users. If you have heavier demands then you should use a professional, stand alone DBMS.

GeoNetwork does not require a powerful machine. Good performance can be obtained even with 1GB of RAM. The suggested amount is 2GB. For hard disk space, you have to consider the space required for the application itself (about 350 MB) and the space required for data, which can require 50 GB or more. A simple disk of 250 GB should be OK. You also need some space for the search index which is located in `GEONETWORK_DATA_DIR/index` (by default GEONETWORK_DATA_DIR is `INSTALL_DIR/web/geonetwork/WEB_INF/data`. However, even with a few thousand metadata records, the index is small so usually 500 MB of space is more than enough.

The software is run in different ways depending on the servlet container you are using:

- **Tomcat** - GeoNetwork is available as a WAR file which you can put into the Tomcat webapps directory. Tomcat will deploy the WAR file when it is started. You can then use the Tomcat manager web application to stop/start GeoNetwork. You can also use the startup.* and shutdown.* scripts located in the Tomcat bin directory (.* means .sh or .bat depending on your OS) but if you have other web applications in the tomcat container, then they will also be affected.
- **Jetty** - If you use the provided container you can use the scripts in GeoNetwork's bin directory. The scripts are start-geonetwork.* and stop-geonetwork.* and you must be inside the bin directory to run them. You can use these scripts just after installation.

1.2 Tools

The following tools are required to be installed to setup a development environment for GeoNetwork:

- **Java** - Developing with GeoNetwork requires Java Development Kit (JDK) 1.6 or greater (scroll down to find 1.6) OR the OpenJDK (version 7 or higher only) for Linux users.
- **Maven** - GeoNetwork uses Maven to manage the build process and the dependencies. Once is installed, you should have the mvn command in your path (on Windows systems, you have to open a shell to check).
- **Git** - GeoNetwork source code is stored and versioned in a Git repository on Github. Depending on your operating system a variety of git clients are avalaible. Check in http://git-scm.com/downloads/guis for some alternatives. Good documentation can be found on the git website: http://git-scm.com/documentation and on the Github website https://help.github.com/.
- **Ant** - GeoNetwork uses Ant to build the installer. Version 1.6.5 works but any other recent version should be OK. Once installed, you should have the ant command in your path (on Windows systems, you have to open a shell to check).
- **Sphinx** - To create the GeoNetwork documentation in a nice format Sphinx is used.

1.3 Check out source code

If you just want to quickly get the code the fastest way is to download the zip bundle: https://github.com/geonetwork/core-geonetwork/zipball/master

However, it is recommended that if you want to contribute back to Geonetwork you create a Github account, fork the Geonetwork repository and work on your fork. This is a huge benefit because you can push your changes to your repository as much as you want and when a feature is complete you can make a 'Pull Request'. Pull requests are the recommended method of contributing back to Geonetwork because Github has code review tools and merges are much easier than trying to apply a patch attached to a ticket.

The Geonetwork Repository is at: https://github.com/geonetwork/core-geonetwork.

Follow the instructions on the Github website to get started (make accounts, how to fork etc...) http://help.github.com/

Once you have the repository forked or cloned locally you can begin to work.

A clone contains all branches so you can list the branches with:

```
$ git branch -a
```

Just look at last section (ignoring remotes/origin/). To checkout a branch just:

```
$ git checkout 2.8.x
```

Typically work is done on branches and merged back so when developing normally you will go change to the branch you want to work on, create a branch from there, work and then merge the changes back (or make a Pull Request on Github). There are many great guides (See the links above) but here is a quick sequence illustrating how to make a change and commit the change.

```
$ git checkout master
   # master is the 'trunk' and main development branch
   # the checkout command "checks out" the requested branch
$ git checkout -b myfeature
   # the -b requests that the branch be created
   # ``git branch`` will list all the branches you have checked out locally at some poir
   # ``git branch -a`` will list all branches in repository (checked out or not)
# work work work
$ git status
   # See what files have been modified or added
$ git add <new or modified files>
   # Add all files to be committed ``git add -u`` will add all modified (but not untrack
$ git commit
   # Commit often.  it is VERY fast to commit
   # NOTE: doing a commit is a local operation.  It does not push the change to Github
# more work
# another commit
$ git push origin myfeature
   # this pushed your new branch to Github now you are ready to make a Pull Request to g
```

1.4 Build GeoNetwork

Once you checked out the code from Github repository, go inside the GeoNetwork's root folder and execute the maven build command:

```
$ mvn clean install
```

If the build is succesful you'll get an output like:

```
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO] ------------------------------------------------------------------------
[INFO] GeoNetwork opensource ..............................  SUCCESS [1.825s]
[INFO] Caching xslt module ................................  SUCCESS [1.579s]
[INFO] Jeeves modules .....................................  SUCCESS [1.140s]
[INFO] Oaipmh modules .....................................  SUCCESS [0.477s]
[INFO] ArcSDE module (dummy-api) ..........................  SUCCESS [0.503s]
[INFO] GeoNetwork Web module ..............................  SUCCESS [31.758s]
[INFO] GeoServer module ...................................  SUCCESS [16.510s]
[INFO] Gast module ........................................  SUCCESS [24.961s]
[INFO] ------------------------------------------------------------------------
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
```

```
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1 minute 19 seconds
[INFO] Finished at: Tue Aug 03 16:49:15 CEST 2010
[INFO] Final Memory: 79M/123M
[INFO] ------------------------------------------------------------------------
```

and your local maven repository should contain the GeoNetwork artifacts created (`$HOME/.m2/repository/org/geonetwork-opensource`).

Note: Many Maven build options are available. Please refer to the maven documentation for any other options, Maven: The Complete Reference

---

For instance, you might like to use following options :

```
-- Skip test
$ mvn install -Dmaven.test.skip=true

-- Offline use
$ mvn install -o
```

Please refer to the maven documentation for any other options, Maven: The Complete Reference

1.4.1 Run embedded jetty server

Maven comes with built-in support for Jetty via a plug-in.

To run GeoNetwork with embedded jetty server you have to change directory to the root of the **web** module, and then execute the following maven command:

```
$ mvn jetty:run
```

After a moment, GeoNetwork should be accessible at: http://localhost:8080/geonetwork

1.4.2 Source code documentation

The GeoNetwork Java source code is based on Javadoc. Javadoc is a tool for generating API documentation in HTML format from doc comments in source code. To see documentation generated by the Javadoc tool, go to:

- GeoNetwork opensource Javadoc

1.5 Creating the installer

To run the build script that creates the installer you need the Ant tool. You can generate an installer by running the ant command inside the **installer** directory:

```
$ ant

Buildfile: build.xml
setProperties:
...
```

```
BUILD SUCCESSFUL
Total time: 31 seconds
```

Both platform independent and Windows specific installers are generated by default.

Make sure you update version number and other relevant properties in the `installer/build.xml` file

You can also create an installer that includes a Java Runtime Environment (JRE) for Windows. This will allow GeoNetwork to run on a compatible, embedded JRE and thus avoid error messages caused by JRE incompatibilities on the PC.

Creating an installer with an embedded JRE requires you to first download and unzip the JRE in a folder jre1.5.0_12 at the project root level. Refer to the installer-config-win-jre.xml file for exact configuration.

1.5.1 Packaging GeoNetwork using Maven

Using Maven, you have the ability to package GeoNetwork in two different ways :

- WAR files (geonetwork.war, geoserver.war)
- Binary ZIP package (with Jetty embedded)

The Assembly Plugin is used to create the packages using

```
$ mvn package assembly:assembly
```

The Assembly Plugin configuration is in the release module (See bin.xml and zip-war.xml).

1.6 Eclipse setup

1.6.1 Setting eclipse preferences

- **M2_REPO** Classpath Variable:
- Navigate to **Java> Build Path> Classpath Variable**
- Press New.. button
- In Name field enter M2_REPO
- In Path field enter the path to your .m2/repository_directory
- Example: “C:Documents and Settingsm.coudert.m2repository”

An alternative to set up this variable directly using maven could to run the following command into your workspace directory

```
$ mvn -Declipse.workspace=. eclipse:add-maven-repo
```

- Generate Eclipse project files

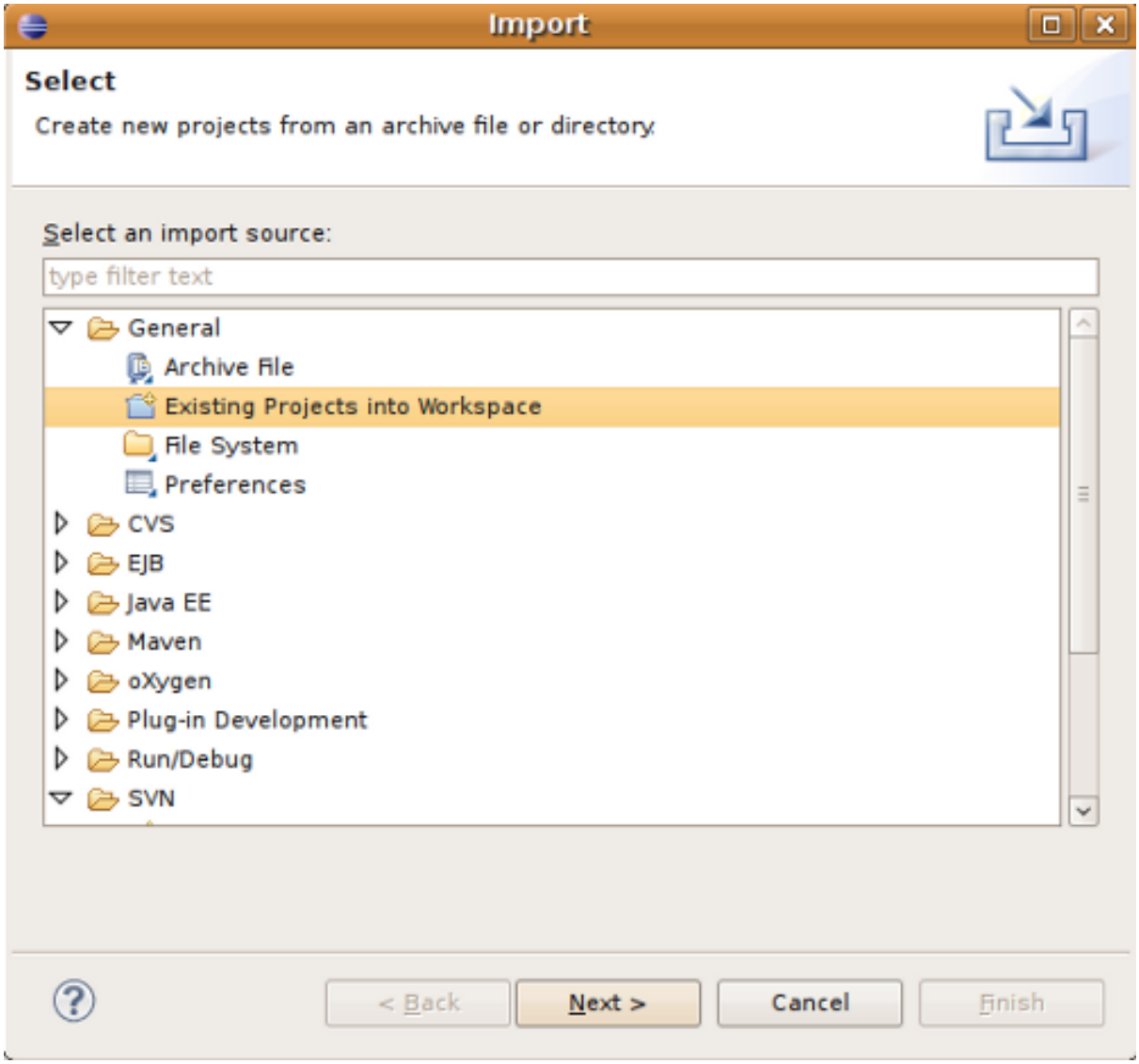
To generate all the .classpath and .project files execute the following command at the project root directory

```
$ mvn eclipse:eclipse
```

1.6.2 Import source code

In order to import the source code, follow instructions below :

- Press **File> Import** Menu item
- In new dialog Select **General> Existing Projects into Workspace**
- Press Next



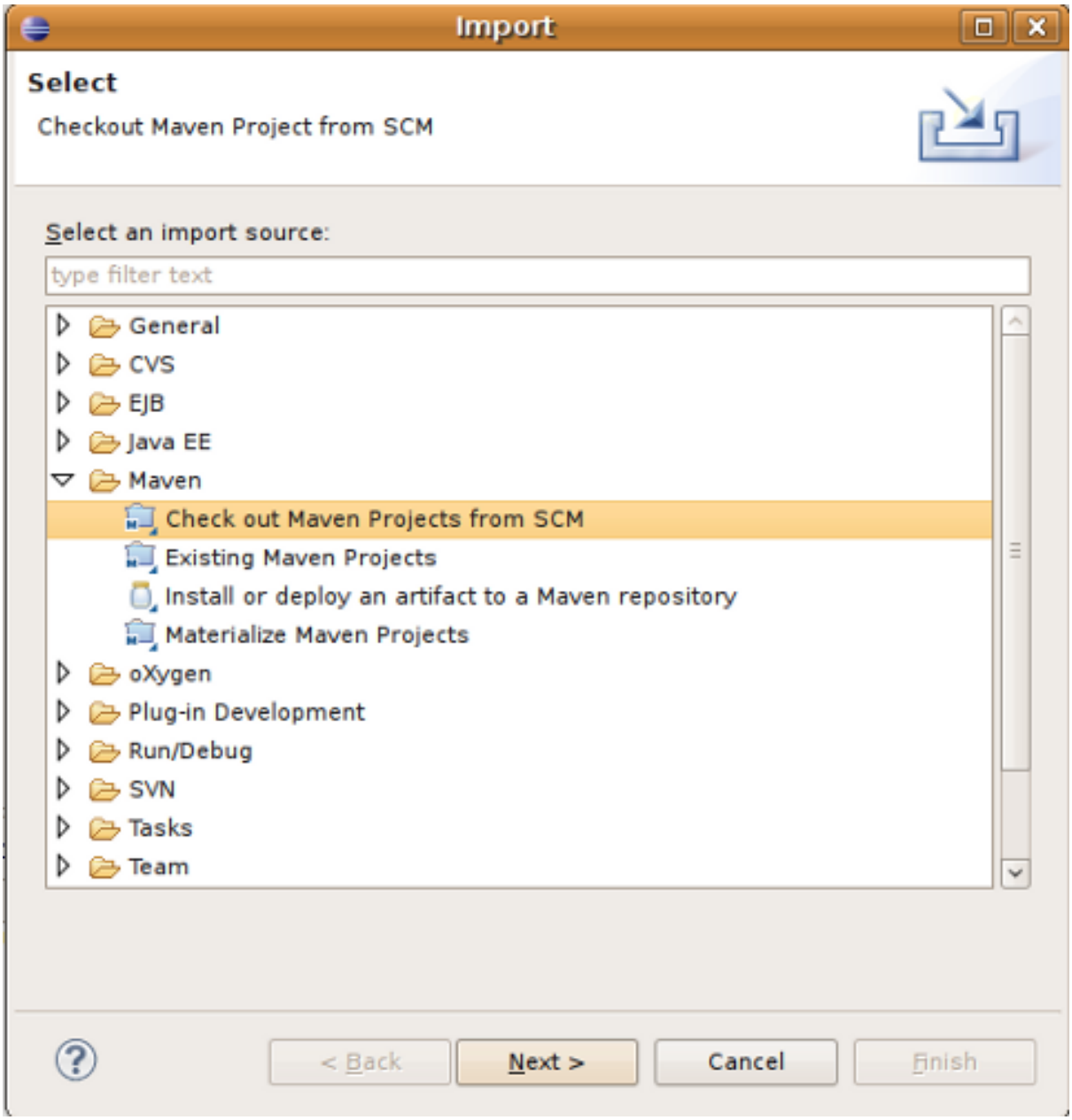
- In **Select root directory** field enter where your code is:
- example: C:devgeonetworktrunk
- Select All projects and Press **Finish** button.

1.6.3 Setting m2eclipse plugin

To install m2eclipse, please refer to the following documentation.

Then click on **File > Import > Maven > Check out Maven Projects From SCM** Choose svn and enter your Github fork as **SCM URL** options. (If you have not made a fork you can use:

git://github.com/geonetwork/core-geonetwork.git)



Note: It is also possible to import existing Maven projects using Maven (m2eclipse) import facilities choosing the **Existing Maven projects** option.

1.6.4 Debugging into eclipse

- Tomcat Server :

TODO

- Remote debuging :
- How do I configure Tomcat to support remote debugging?

- How do I remotely debug Tomcat using Eclipse?

CHAPTER 2

Create GeoNetwork releases

2.1 Create a stable release for GeoNetwork

This guide details the process of performing a GeoNetwork release.

Note:

- BRANCH: Branches are created for major stables releases and end with .x (for example 2.8.x)
- VERSION (for tag): version to release (for example 2.8.1)
- NEW_VERSION (for branch): next version (for example 2.8.2)

2.1.1 Release committee

To create new releases a committee of 3-4 persons should be chosen. The members of the committee are responsible for creating the releases following the steps described in this section.

A rotation policy can be use to select a person from the committee who will be responsible for creating each release.

2.1.2 Notify developer lists

It is good practice to notify the GeoNetwork developer list of the intention to make the release a few days in advance.

On the day the release is being made a warning mail **must** be sent to the list asking that developers refrain from committing until the release tag has been created.

2.1.3 Prerequisites

1. Commit access to GeoNetwork git
2. Administration rights to SourceForge server to publish the release

2.1.4 Update source code from GIT

Warning: These steps must be performed on a branch

1. Check out the **branch** to be released

```
$ git clone --recursive https://github.com/geonetwork/core-geonetwork.git geonetwork-[VE
$ cd geonetwork-[VERSION]
$ git checkout -b [VERSION] origin/[VERSION]
$ git remote-url set origin git@github.com:geonetwork/core-geonetwork.git
```

2. Set submodules to correct **branch**: Submodules are in subdirectories:`docs`, `gast`, `geoserver`, `installer`, `maven_repo`, `release`. The procedure for each submodule is the same:

```
$ cd <submodule>
$ git checkout -b [VERSION] origin/[VERSION]
```

Tests and issues must be resolved for the new release

1. Build the application and run the integration tests in `web-itests`

```
$ mvn clean install -Pitests
```

If tests are passed then proceed with the next step. Otherwise:

- If a critical bug is detected, fix before continuing.
- If a non-critical bug is detected, open a new ticket and set its version number to the next release and continue

2. Create an installer

```
$ cd installer
$ ant
```

3. Using the installer, install GeoNetwork in a folder called `geonetwork-[VERSION]`
4. Test any issues relating to the release. Check GeoNetwork trac.

If all tests pass, proceed with the release. Otherwise:

- If a critical bug is detected, fix before continuing with the release.
- If a non-critical bug is detected, set the version number of the ticket to the next release and continue

---

Note: to discuss and get feedback

This approach requires a code freeze on the branch until the release is done, if all test are ok (no bugs found in tests or non critical bugs), 1 day or less is ok.

If a critical bug is detected, the code freeze can take some time if a bug is complicated to fix. To avoid a prolonged code freeze some alternatives can be considered:

1. If there is a commit on the branch before the critical fix is committed then it will be part of the release (properly tested in a new release cycle)

2. Fork the repository and commit the critical fix to that fork, then pull the fix into the branch. This allows other developers to commit on the branch while the critical bug is being solved.

2.1.5 Update changes.txt file

1. Add an entry to `docs/changes.txt` describing the changes in this new release, using the following template.

Comments from the GIT commits are used to extract the most important changes (use `git help log` to work out what you need eg. git log –since="2 weeks ago"). Some cleanup may be required before the log output can be added to the changes.txt document)

```
================================================================================
===
=== GeoNetwork [VERSION]: List of changes
===
================================================================================
--------------------------------------------------------------------------------
--- Bug fixes
--------------------------------------------------------------------------------
- Fix fo issue #NUMBER: Description of fix
- Fix fo issue #NUMBER: Description of fix
- ...

--------------------------------------------------------------------------------
--- Changes
--------------------------------------------------------------------------------
- Description of change
- Description of change
- ...
```

2. Commit `docs/changes.txt` file

```
$ git commit -m "Updating CHANGES for [VERSION]" docs/changes.txt
$ git push
```

2.1.6 Update version numbers for a release

A *NIX (Linux, OSX etc..) operating system can use the following shell script.

1. Execute in root of the **branch** source tree `updateReleaseVersions.sh`. Example to create version 2.6.1 from 2.6.1-SNAPSHOT

```
$ ./updateReleaseVersions.sh 2.6.1
```

2. Commit updated files

```
$ git commit -a -m "Updated files version to [VERSION]"
$ git push
```

2.1.7 Create release tag

1. Create a tag for the release

```
$ git tag -a [VERSION] -m "Create tag for release [VERSION]"
```

2. Push the tag to github:

```
$ git push origin [VERSION]
```

2.1.8 Build release artifacts

Warning: All operations for the remainder of this guide must be performed from the release tag unless otherwise stated.

1. Uncomment doc sections in `web/pom.xml`

```
<webResources>
  <resource>
              <directory>../docs/eng/users/build/html</directory>
              <targetPath>docs/eng/users</targetPath>
              </resource>
              ...
</webResources>
```

2. Build documentation. In `docs` folder, execute

```
$ mvn clean install
```

Note: Building the GeoNetwork documentation requires the following be installed:

- Sphinx, version 0.6 or greater (sphinx-doc on ubuntu/debian)
- TeX Live (texlive-full on ubuntu/debian)
- make utility

---

3. Compile and build from the root of the source tree

```
$ mvn clean install
```

2.1.9 WAR distribution

After building the release artifacts in the previous steps, the war distribution of the new release will be: `web/target/geonetwork.war`

2.1.10 Build installers

To build the Windows and platform independent installers, execute the next command in the `installer` folder

```
$ ant
```

The installers (exe and jar) are created in a folder `geonetwork-[VERSION]`

2.1.11 Upload and release on SourceForge

All of the artifacts generated so far need to be uploaded to the SourceForce File release System:

1. WAR distribution
2. Installers (exe and jar)

Note: This step requires administrative privileges in SourceForge for the GeoNetwork opensource project.

---

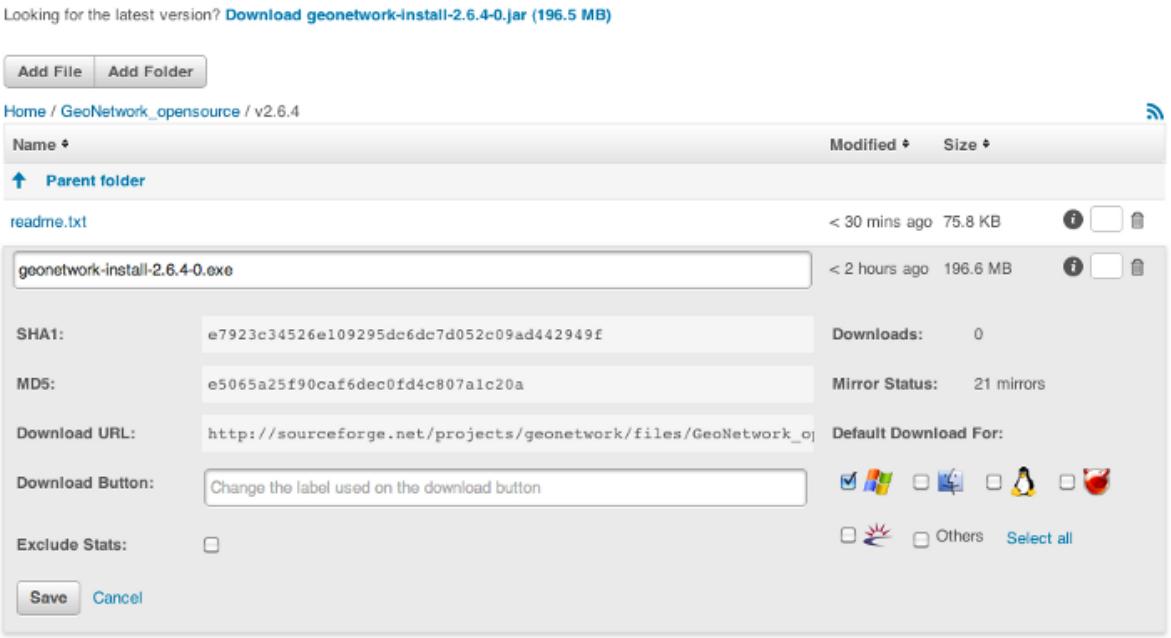
1. Log in to SourceForge.
2. Go to the ‘ GeoNetwork Files section <https://sourceforge.net/projects/geonetwork/files/GeoNetwork_opensource/
3. Add the new v[VERSION] folder for this release.

4.a. Using the commandline secure copy is the simplest way for developers working under a *NIX like system:

```
$ scp geonetwork.war username@frs.sourceforge.net:/home/frs/project/g/ge/geonetwork/GeoN
$ scp geonetwork-[VERSION].jar username@frs.sourceforge.net:/home/frs/project/g/ge/geone
$ scp geonetwork-[VERSION].exe username@frs.sourceforge.net:/home/frs/project/g/ge/geone
$ scp docs/readme.txt username@frs.sourceforge.net:/home/frs/project/g/ge/geonetwork/Geo
```

4.b. The same can be accomplished in Windows using WinSCP. Or a desktop client like Cyberduck on Windows and Mac OS X

5. Once the upload of the files has been completed, use the web interface to set the default download files. The (i) button allows to set the default operating systems for each installer (.exe for Windows and .jar for all other systems).



6. The default downloads are ready now.

2.1.12 Update geonetwork-opensource website

The website requires updates to reflect the new release. Update the version number and add a new news entry in the following files:

```
website/docsrc/conf.py
website/docsrc/docs.rst
website/docsrc/downloads.rst
website/docsrc/index.rst
website/docsrc/news.rst
website/checkup_docs.sh
```

Commit the changes and build the website using the Hudson deployment system

2.1.13 Announce the release

Mailing lists

Send an email to both the developers list and users list announcing the release.

`TODO`: Template mail?

SourceForge

`TODO`: Do we create SourceForge notifications?

2.1.14 Upgrade branch pom versions

> **Warning:** This steps must be performed using branch code.

After a release has been created, the branch version number must be set to the version number of the next release. On a *NIX (Linux, OSX etc..) operating system you can use the shell script `updateBranchVersions.sh` to do this.

1. From the root of the **branch** source tree execute the script `updateBranchVersions.sh`. To update from version 2.6.1-SNAPSHOT to 2.6.2-SNAPSHOT for example

   ```
   $ ./updateBranchVersions.sh 2.6.1 2.6.2
   ```

2. Commit the updated files

   ```
   $ git commit -a -m "Updated files version to [VERSION]-SNAPSHOT"
   $ git push
   ```

CHAPTER 3

Harvesting

3.1 Structure

The harvesting capability is built around 3 areas: JavaScript code, Java code and XSL stylesheets (on both the server and client side).

3.1.1 JavaScript code

This refers to the web interface. The code is located in the web/geonetwork/scripts/harvesting folder. Here, there is a subfolder for each harvesting type plus some classes for the main page. These are:

1. *harvester.js*: This is an abstract class that must be implemented by harvesting types. It defines some information retrieval methods (getType, getLabel, etc...) used to handle the harvesting type, plus one getUpdateRequest method used to build the XML request to insert or update entries.
2. *harvester-model.js*: Another abstract class that must be implemented by harvesting types. When creating the XML request, the only method substituteCommon takes care of adding common information like privileges and categories taken from the user interface.
3. *harvester-view.js*: This is an important abstract class that must be implemented by harvesting types. It takes care of many common aspects of the user interface. It provides methods to add group's privileges, to select categories, to check data for validity and to set and get common data from the user interface.
4. *harvesting.js*: This is the main JavaScript file that takes care of everything. It starts all the sub-modules, loads XML strings from the server and displays the main page that lists all harvesting nodes.
5. *model.js*: Performs all XML requests to the server, handles errors and decode responses.
6. *view.js*: Handles all updates and changes on the main page.
7. *util.js*: just a couple of utility methods.

3.1.2 Java code

The harvesting package is located in `web/src/main/java/org/fao/geonet/kernel/harvest`. Here too, there is one subfolder for each harvesting type. The most important classes for the implemen-

tor are:

1. *AbstractHarvester*: This is the main class that a new harvesting type must extends. It takes care of all aspects like adding, updating, removing, starting, stopping of harvesting nodes. Some abstract methods must be implemented to properly tune the behaviour of a particular harvesting type.
2. *AbstractParams*: All harvesting parameters must be enclosed in a class that extends this abstract one. Doing so, all common parameters can be transparently handled by this abstract class.

All others are small utility classes used by harvesting types.

3.1.3 XSL stylesheets

Stylesheets are spread in some folders and are used by both the JavaScript code and the server. The main folder is located at `web/src/webapp/xsl/harvesting`. Here there are some general stylesheets, plus one subfolder for each harvesting type. The general stylesheets are:

1. *buttons.xsl*: Defines all button present in the main page (*activate*, *deactivate*, *run*, *remove*, *back*, *add*, *refresh*), buttons present in the "add new harvesting" page (*back* and *add*) and at the bottom of the edit page (*back* and *save*).
2. *client-error-tip.xsl*: This stylesheet is used by the browser to build tooltips when an harvesting error occurred. It will show the error class, the message and the stacktrace.
3. *client-node-row.xsl*: This is also used by the browser to add one row to the list of harvesting nodes in the main page.
4. *harvesting.xsl*: This is the main stylesheet. It generates the HTML page of the main page and includes all panels from all the harvesting nodes.

In each subfolder, there are usually 4 files:

1. *xxx.xsl*: This is the server stylesheets who builds all panels for editing the parameters. XXX is the harvesting type. Usually, it has the following panels: site information, search criteria, options, privileges and categories.
2. *client-privil-row.xsl*: This is used by the JavaScript code to add rows in the group's privileges panel.
3. *client-result-tip.xsl*: This is used by the JavaScript code (which inherits from harvester-view.js) to show the tool tip when the harvesting has been successful.
4. *client-search-row.xsl*: Used in some harvesting types to generate the HTML for the search criteria panel.

As you may have guessed, all client side stylesheets (those used by JavaScript code) start with the prefix client-.

Another set of stylesheets are located in `web/src/webapp/xsl/xml/harvesting` and are used by the xml.harvesting.get service. This service is used by the JavaScript code to retrieve all the nodes the system is currently harvesting from. This implies that a stylesheet (one for each harvesting type) must be provided to convert from the internal setting structure to an XML structure suitable to clients.

The last file to take into consideration contains all localised strings and is located at `web/src/webapp/loc/XX/xml/harvesting.xml` (where XX refers to a language code). This file is used by both JavaScript code and the server.

3.2 Data storage

Harvesting nodes are stored inside the Settings table. Further useful information can be found in the chapter Harvesting.

The SourceNames table is used to keep track of the uuid/name couple when metadata get migrated to different sites.

3.3 Guidelines

To add a new harvesting type, follow these steps:

1. Add the proper folder in `web/src/webapp/scripts/harvesting`, maybe copying an already existing one.
2. Edit the harvesting.js file to include the new type (edit both constructor and init methods).
3. Add the proper folder in `web/src/webapp/xsl/harvesting` (again, it is easy to copy from an already existing one).
4. Edit the stylesheet `web/src/webapp/xsl/harvesting/harvesting.xsl` and add the new type
5. Add the transformation stylesheet in `web/src/webapp/xsl/xml/harvesting`. Its name must match the string used for the harvesting type.
6. Add the Java code in a package inside `org.fao.geonet.kernel.harvest.harvester`.
7. Add proper strings in `web/src/webapp/loc/XX/xml/harvesting.xml`.

Here is a list of steps to follow when adding a new harvesting type:

1. Every harvesting node (not type) must generate its UUID. This UUID is used to remove metadata when the harvesting node is removed and to check if a metadata (which has another UUID) has been already harvested by another node.
2. If a harvesting type supports multiple searches on a remote site, these must be done sequentially and results merged.
3. Every harvesting type must save in the folder images/logos a GIF image whose name is the node's UUID. This image must be deleted when the harvesting node is removed. This is necessary to propagate harvesting information to other GeoNetwork nodes.
4. When a harvesting node is removed, all collected metadata must be removed too.
5. During harvesting, take in mind that a metadata could have been removed just after being added to the result list. In this case the metadata should be skipped and no exception raised.
6. The only settable privileges are: view, dynamic, featured. It does not make sense to use the others.
7. If a node raises an exception during harvesting, that node will be deactivated.
8. If a metadata already exists (its UUID exists) but belong to another node, it must not be updated even if it has been changed. This way the harvesting will not conflict with the other one. As a side effect, this prevent locally created metadata from being changed.
9. The harvesting engine stores results in the database as part of the harvest history.

10. When harvesting parameters are changed, the new harvesting type must use them during the next harvesting without requiring server restart.

CHAPTER 4

Schema Plugins

A schema in GeoNetwork is a directory with stylesheets, XML schema descriptions (XSDs) and other information necessary for GeoNetwork to index, view and possibly edit content from XML metadata records.

To be used in GeoNetwork, a schema directory can be manually placed in the `config/schema_plugins` sub directory of the geonetwork data directory. The default geonetwork data directory location is `INSTALL_DIR/web/geonetwork/WEB-INF/data`. The contents of these schemas are parsed during GeoNetwork initialization. If valid, they will be available for use when GeoNetwork starts up.

Schemas can also added to GeoNetwork dynamically if a zip archive of the schema directory is created and then uploaded to GeoNetwork in one of following ways using functions in the Administration menu:

1. Server file path (specified using file chooser)
2. HTTP URL (eg. http://somehost/somedirectory/iso19139.mcp.zip)
3. As an online resource attached to an ISO19115/19139 metadata record

Uploaded schemas are also stored in the `config/schema_plugins` sub directory of the geonetwork data directory.

4.1 Contents of a GeoNetwork schema

When installed, a GeoNetwork schema is a directory.

The following subdirectories can be present:

- **convert**: (*Mandatory*) Directory of XSLTs to convert metadata from or to this schema. This could be to convert metadata to other schemas or to convert metadata from other schemas and formats to this schema. Eg. `convert/oai_dc.xsl`
- **loc**: (*Optional*) Directory of localized information: labels, codelists or schema specific strings. Eg. `loc/eng/codelists.xml`
- **present**: (*Mandatory*) contains XSLTs for presenting metadata in the viewer/editor and in response to CSW requests for brief, summary and full records.
- **process**: (*Optional*) contains XSLTs for processing metadata elements by metadata suggestions mechanism (see **suggest.xsl** below).

- **sample-data**: (*Mandatory*) Sample metadata for this schema. The metadata samples are in MEF format so that samples can have thumbnails or browse graphics as well as online resources.
- **schema**: (*Optional*) Directory containing the official XSDs of the metadata schema. If the schema is described by a DTD then this directory is optional. Note that schemas described by a DTD cannot be edited by GeoNetwork.
- **templates**: (*Optional*) Directory containing template and subtemplate metadata records for this schema. Template metadata records are usually metadata records with the set of elements (and content) that will be used for a specific purpose. Eg. iso19139.mcp schema has a 'Minimum Element' template that has the mandatory elements for the schema and a example of the content that is expected.

The following stylesheets can be present:

- **extract-date-modified.xsl**: (*Mandatory*) Extract the date of modification from the metadata record.
- **extract-gml.xsl**: (*Mandatory*) Extract the spatial extent from the metadata record as a GML GeometryCollection element.
- **extract-thumbnails.xsl**: (*Optional*) Extract the browse graphic/thumbnail from the metadata record.
- **extract-uuid.xsl**: (*Mandatory*) Extract the UUID of the metadata record.
- **index-fields.xsl**: (*Mandatory*) Index the metadata record content in Lucene. Creates the Lucene document used by GeoNetwork to index the metadata record content.
- **schematron-rules-*.xsl**: (*Optional*) XSLT created from schematron rules when building the schema plugin (see schematrons directory).
- **set-thumbnail.xsl**: (*Optional*) Set the browse graphic/thumbnail in the metadata record.
- **set-uuid.xsl**: (*Optional*) Set the UUID of the metadata record.
- **suggest.xsl**: (*Optional*) XSLT run by metadata suggestions service. The XSLT contains processes that can be registered and run on different elements of a metadata record. eg. expand keyword field with comma separated content into multiple keyword fields. See http://trac.osgeo.org/geonetwork/wiki/proposals/MetadataEditorSuggestion for more info.
- **unset-thumbnail.xsl**: (*Optional*) Remove the browse graphic/thumbnail from the metadata record.
- **update-child-from-parent-info.xsl**: (*Optional*) XSLT to specify which elements in a child record are updated from a parent record. Used to manage hierarchical relationships between metadata records.
- **update-fixed-info.xsl**: (*Optional*) XSLT to update 'fixed' content in metadata records.

The following configuration files can be present:

- **oasis-catalog.xml**: (*Optional*) An oasis catalog describing any mappings that should be used for this schema eg. mapping URLs to local copies such as schemaLocations eg. http://www.isotc211.org/2005/gmd/gmd.xsd is mapped to `schema/gmd/gmd.xsd`. Path names used in the oasis catalog are relative to the location of this file which is the schema directory.
- **schema.xsd**: (*Optional*) XML schema directory file that includes the XSDs used by this metadata schema. If the schema uses a DTD then this file should not be present. Metadata records from

schemas that use DTDs cannot be edited in GeoNetwork.

- **schema-conversions.xml**: (*Optional*) XML file that describes the converters that can be applied to records belonging to this schema. This information is used to show these conversions as options for the user to choose when a metadata record belonging to this schema is shown in the search results.
- **schema-ident.xml**: (*Mandatory*) XML file that contains the schema name, identifier, version number and details on how to recognise metadata records that belong to this schema. This file has an XML schema definition in `INSTALL_DIR/web/geonetwork/xml/validation/schemaPlugins/schema-ident.xsd` which is used to validate it when the schema is loaded.
- **schema-substitutes.xml**: (*Optional*) XML file that redefines the set of elements that can be used as substitutes for a specific element.
- **schema-suggestions.xml**: (*Optional*) XML file that tells the editor which child elements of a complex element to automatically expand in the editor.

To help in understanding what each of these components is and what is required, we will now give a step-by-step example of how to build a schemaPlugin for GeoNetwork.

4.2 Preparation

In order to create a schema plugin for GeoNetwork, you should check out the schemaPlugins directory from the GeoNetwork sourceforge subversion repository. You can do this by installing subversion on your workstation and then executing the following command:

```
svn co https://github.com/geonetwork/schema-plugins/branches/2.8.x schemaPlugins
```

This will create a directory called schemaPlugins with some GeoNetwork schema plugins in it. To work with the example shown here, you should create your new schema plugin in a subdirectory of this directory.

4.3 Example - ISO19115/19139 Marine Community Profile (MCP)

The Marine Community Profile (MCP) is a profile of ISO19115/19139 developed for and with the Marine Community. The profile extends the ISO19115 metadata standard and is implemented using an extension of the XML implementation of ISO19115 described in ISO19139. Both the ISO19115 metadata standard and its XML implementation, ISO19139, are available through ISO distribution channels.

The documentation for the Marine Community Profile can be found at http://www.aodc.gov.au/files/MarineCommunityProfilev1.4.pdf. The implementation of the Marine Community Profile as XML schema definitions is based on the approach described at https://www.seegrid.csiro.au/wiki/AppSchemas/MetadataProfiles. The XML schema definitions (XSDs) are available at the URL http://bluenet3.antcrc.utas.edu.au/mcp-1.4.

Looking at the XML schema definitions, the profile adds a few new elements to the base ISO19139 standard. So the basic idea in defining a plugin Marine Community Profile schema for GeoNetwork is to use as much of the basic ISO19139 schema definition supplied with GeoNetwork as possible.

We'll now describe in basic steps how to create each of the components of a plugin schema for GeoNetwork that implements the MCP.

4.3.1 Creating the schema-ident.xml file

Now we need to provide the information necessary to identify the schema and metadata records that belong to the schema. The schema-ident.xml file for the MCP is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://geonetwork-opensource.org/schemas/schema-ident"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        .....>
  <name>iso19139.mcp</name>
  <id>19c9a2b2-dddb-11df-9df4-001c2346de4c</id>
  <version>1.5</version>
  <schemaLocation>
    http://bluenet3.antcrc.utas.edu.au/mcp
    http://bluenet3.antcrc.utas.edu.au/mcp-1.5-experimental/schema.xsd
    http://www.isotc211.org/2005/gmd
    http://www.isotc211.org/2005/gmd/gmd.xsd
    http://www.isotc211.org/2005/srv
    http://schemas.opengis.net/iso/19139/20060504/srv/srv.xsd
  </schemaLocation>
  <autodetect xmlns:mcp="http://bluenet3.antcrc.utas.edu.au/mcp"
              xmlns:gmd="http://www.isotc211.org/2005/gmd"
              xmlns:gco="http://www.isotc211.org/2005/gco">
    <elements>
      <gmd:metadataStandardName>
        <gco:CharacterString>
          Australian Marine Community Profile of ISO 19115:2005/19139|
          Marine Community Profile of ISO 19115:2005/19139
        </gco:CharacterString>
      </gmd:metadataStandardName>
      <gmd:metadataStandardVersion>
        <gco:CharacterString>
          1.5-experimental|
          MCP:BlueNet V1.5-experimental|
          MCP:BlueNet V1.5
        </gco:CharacterString>
      </gmd:metadataStandardVersion>
    </elements>
  </autodetect>
</schema>
```

Each of the elements is as follows:

- **name** - the name by which the schema will be known in GeoNetwork. If the schema is a profile of a base schema already added to GeoNetwork then the convention is to call the schema <base_schema_name>.<namespace_of_profile>.
- **id** - a unique identifier for the schema.
- **version** - the version number of the schema. Multiple versions of the schema can be present in GeoNetwork.
- **schemaLocation** - a set of pairs, where the first member of the pair is a namespace URI and the second member is the official URL of the XSD. The contents of this element will be added to the root element of any metadata record displayed by GeoNetwork as a schemaLocation/noNamespaceSchemaLocation attribute, if such as attribute does not already exist. It will also be used whenever an official schemaLocation/noNamespaceSchemaLocation is required (eg. in response to a ListMetadataFormats OAI request).

- **autodetect** - contains elements or attributes (with content) that must be present in any metadata record that belongs to this schema. This is used during schema detection whenever GeoNetwork receives a metadata record of unknown schema.

After creating this file you can validate it manually using the XML schema definition (XSD) in `INSTALL_DIR/web/geonetwork/xml/validation/schemaPlugins/schema-ident.xsd`. This XSD is also used to validate this file when the schema is loaded. If schema-ident.xml failes validation, the schema will not be loaded.

More on autodetect

The autodetect section of schema-ident.xml is used when GeoNetwork needs to identify which metadata schema a record belongs to.

The five rules that can be used in this section in order of evaluation are:

1. Attributes - Find one or more attributes and/or namespaces in the document. An example use case is a profile of ISO19115/19139 that adds optional elements under a new namespace to gmd:identificationInfo/gmd:MD_DataIdentification. To detect records that belong to this profile the autodetect section in the schema-ident.xml file could look something like the following:

```
<autodetect xmlns:cmar="http://www.marine.csiro.au/schemas/cmar.xsd">
        <!-- catch all cmar records that have the cmar vocab element -->
        <attributes cmar:vocab="http://www.marine.csiro.au/vocabs/projectCodes.xml"/>
</autodetect>
```

Some other points about attributes autodetect:

- multiple attributes can be specified - all must be match for the record to be recognized as belonging to this schema.
- if the attributes have a namespace then the namespace should be specified on the autodetect element or somewhere in the schema-ident.xml document.

2. Elements - Find one or more elements in the document. An example use case is the one shown in the example schema-ident.xml file earlier:

```
<autodetect xmlns:mcp="http://bluenet3.antcrc.utas.edu.au/mcp"
            xmlns:gmd="http://www.isotc211.org/2005/gmd"
            xmlns:gco="http://www.isotc211.org/2005/gco">
  <elements>
    <gmd:metadataStandardName>
      <gco:CharacterString>
        Australian Marine Community Profile of ISO 19115:2005/19139|
        Marine Community Profile of ISO 19115:2005/19139
      </gco:CharacterString>
    </gmd:metadataStandardName>
    <gmd:metadataStandardVersion>
      <gco:CharacterString>
        1.5-experimental|
        MCP:BlueNet V1.5-experimental|
        MCP:BlueNet V1.5
      </gco:CharacterString>
    </gmd:metadataStandardVersion>
  </elements>
</autodetect>
```

Some other points about elements autodetect:

- multiple elements can be specified - eg. as in the above, both metadataStandardName and metadataStandardVersion have been specified - all must be match for the record to be recognized as belonging to this schema.
- multiple values for the elements can be specified. eg. as in the above, a match for gmd:metadataStandardVersion will be found for `1.5-experimental` OR `MCP:BlueNet V1.5-experimental` OR `MCP:BlueNet V1.5` - the vertical line or pipe character '|' is used to separate the options here.
- if the elements have a namespace then the namespace(s) should be specified on the autodetect element or somewhere in the schema-ident.xml document before the element in which they are used - eg. in the above there are there namespace declarations on the autodetect element so as not to clutter the content.

3. Root element - root element of the document must match. An example use case is the one used for the eml-gbif schema. Documents belonging to this schema always have root element of eml:eml so the autodetect section for this schema is:

```
<autodetect xmlns:eml="eml://ecoinformatics.org/eml-2.1.1">
        <elements type="root">
                <eml:eml/>
        </elements>
</autodetect>
```

Some other points about root element autodetect:

- multiple elements can be specified - any element in the set that matches the root element of the record will trigger a match.
- if the elements have a namespace then the namespace(s) should be specified on the autodetect element or somewhere in the schema-ident.xml document before the element that uses them - eg. as in the above there is a namespace declaration on the autodetect element for clarity.

4. Namespaces - Find one or more namespaces in the document. An example use case is the one used for the csw:Record schema. Records belonging to the csw:Record schema can have three possible root elements: csw:Record, csw:SummaryRecord and csw:BriefRecord, but instead of using a multiple element root autodetect, we could use the common csw namespace for autodetect as follows:

```
<autodetect>
  <namespaces xmlns:csw="http://www.opengis.net/cat/csw/2.0.2"/>
</autodetect>
```

Some other points about namespaces autodetect:

- multiple namespaces can be specified - all must be present for the record to be recognized as belonging to this schema.
- the prefix is ignored. A namespace match occurs if the namespace URI found in the record matches the namespace URI specified in the namespaces autodetect element.

5. Default schema - This is the fail-safe provision for records that don't match any of the installed schemas. The value for the default schema is specified in the appHandler configuration of the `INSTALL_DIR/web/geonetwork/WEB-INF/config.xml` config file or it could be a default specified by the operation calling autodetect (eg. a value parsed from a user bulk loading some metadata records). For flexibility and accuracy reasons it is preferable that records be detected using the autodetect information of an installed schema. The default schema is just a 'catch all' method of assigning records to a specific schema. The config element in `INSTALL_DIR/web/geonetwork/WEB-INF/config.xml` looks like the following:

```
<appHandler class="org.fao.geonet.Geonetwork">
        .....
        <param name="preferredSchema" value="iso19139" />
        .....
</appHandler>
```

More on autodetect evaluation

The rules for autodetect are evaluated as follows:

```
for-each autodetect rule type in ( 'attributes/namespaces', 'elements',
                                   'namespaces', 'root element' )
        for-each schema
                if schema has autodetect rule type then
                        check rule for a match
                        if match add to list of previous matches
                end if
        end for-each

        if more than one match throw 'SCHEMA RULE CONFLICT EXCEPTION'
        if one match then set matched = first match and break loop
end for-each

if no match then
  if namespaces of record and default schema overlap then
                set match = default schema
        else throw 'NO SCHEMA MATCHES EXCEPTION'
end if

return matched schema
```

As an example, suppose we have three schemas iso19139.mcp, iso19139.mcp-1.4 and iso19139.mcp-cmar with the following autodetect elements:

iso19139.mcp-1.4:

```
<autodetect xmlns:mcp="http://bluenet3.antcrc.utas.edu.au/mcp"
            xmlns:gmd="http://www.isotc211.org/2005/gmd"
            xmlns:gco="http://www.isotc211.org/2005/gco">
  <elements>
    <gmd:metadataStandardName>
      <gco:CharacterString>
        Australian Marine Community Profile of ISO 19115:2005/19139
      </gco:CharacterString>
    </gmd:metadataStandardName>
    <gmd:metadataStandardVersion>
      <gco:CharacterString>MCP:BlueNet V1.4</gco:CharacterString>
    </gmd:metadataStandardVersion>
  </elements>
</autodetect>
```

iso19139.mcp-cmar:

```
<autodetect>
    <attributes xmlns:mcp-cmar="http://www.marine.csiro.au/schemas/mcp-cmar">
</autodetect>
```

iso19139.mcp:

```
<autodetect xmlns:mcp="http://bluenet3.antcrc.utas.edu.au/mcp">
        <elements type="root">
                <mcp:MD_Metadata/>
        </elements>
</autodetect>
```

A record going through autodetect processing (eg. on import) would be checked against:

- iso19139.mcp-cmar first as it has an 'attributes' rule
- then iso19139.mcp-1.4 as it has an 'elements' rules
- then finally against iso19139.mcp, as it has a 'root element' rule.

The idea behind this processing algorithm is that base schemas will use a 'root element' rule (or the more difficult to control 'namespaces' rule) and profiles will use a finer or more specific rule such as 'attributes' or 'elements'.

After setting up schema-ident.xml, our new GeoNetwork plugin schema for MCP contains:

```
schema-ident.xml
```

4.3.2 Creating the schema-conversions.xml file

This file describes the converters that can be applied to metadata records that belong to the schema. Each converter must be manually defined as a GeoNetwork (Jeeves) service that can be called to transform a particular metadata record to a different schema. The schema-conversions.xml file for the MCP is as follows:

```
<conversions>
   <converter name="xml_iso19139.mcp"
              nsUri="http://bluenet3.antcrc.utas.edu.au/mcp"
              schemaLocation="http://bluenet3.antcrc.utas.edu.au/mcp-1.5-experimental/sc
              xslt="xml_iso19139.mcp.xsl"/>
   <converter name="xml_iso19139.mcp-1.4"
              nsUri="http://bluenet3.antcrc.utas.edu.au/mcp"
              schemaLocation="http://bluenet3.antcrc.utas.edu.au/mcp/schema.xsd"
              xslt="xml_iso19139.mcp-1.4.xsl"/>
   <converter name="xml_iso19139.mcpTooai_dc"
              nsUri="http://www.openarchives.org/OAI/2.0/"
              schemaLocation="http://www.openarchives.org/OAI/2.0/oai_dc.xsd"
              xslt="oai_dc.xsl"/>
   <converter name="xml_iso19139.mcpTorifcs"
              nsUri="http://ands.org.au/standards/rif-cs/registryObjects"
              schemaLocation="http://services.ands.org.au/home/orca/schemata/registryObj
              xslt="rif.xsl"/>
</conversions>
```

Each converter has the following attributes:

- **name** - the name of the converter. This is the service name of the GeoNetwork (Jeeves) service and should be unique (prefixing the service name with xml_<schema_name> is a good way to make this name unique).
- **nsUri** - the primary namespace of the schema produced by the converter. eg. xml_iso19139.mcpTorifcs transforms metadata records from iso19139.mcp to the RIFCS

schema. Metadata records in the RIFCS metadata schema have primary namespace URI of http://ands.org.au/standards/rif-cs/registryObjects.

- **schemaLocation** - the location (URL) of the XML schema definition (XSD) corresponding to the nsURI.
- **xslt** - the name of the XSLT that actually carries out the transformation. This XSLT should be located in the convert subdirectory of the schema plugin.

After setting up schema-conversions.xml, our new GeoNetwork plugin schema for MCP contains:

```
schema-conversions.xml  schema-ident.xml
```

4.3.3 Creating the schema directory and schema.xsd file

The schema and schema.xsd components are used by the GeoNetwork editor and validation functions.

GeoNetwork's editor uses the XSDs to build a form that will not only order the elements in a metadata document correctly but also offer options to create any elements that are not in the metadata document. The idea behind this approach is twofold. Firstly, the editor can use the XML schema definition rules to help the user avoid creating a document that is structurally incorrect eg. missing mandatory elements or elements not ordered correctly. Secondly, the same editor code can be used on any XML metadata document with a defined XSD.

If you are defining your own metadata schema then you can create an XML schema document using the XSD language. The elements of the language can be found online at http://www.w3schools.com/schema/ or you can refer to a textbook such as Priscilla Walmsley's Definitive XML Schema (Prentice Hall, 2002). GeoNetwork's XML schema parsing code understands almost all of the XSD language with the exception of redefine, any and anyAttribute (although the last two can be handled under special circumstances).

In the case of the Marine Commuity Profile, we are basically defining a number of extensions to the base standard ISO19115/19139. These extensions are defined using the XSD extension mechanism on the types defined in ISO19139. The following snippet shows how the Marine Community Profile extends the gmd:MD_Metadata element to add a new element called revisionDate:

```
<xs:schema targetNamespace="http://bluenet3.antcrc.utas.edu.au/mcp"
           xmlns:mcp="http://bluenet3.antcrc.utas.edu.au/mcp" .....>

....

<xs:element name="MD_Metadata" substitutionGroup="gmd:MD_Metadata"
                               type="mcp:MD_Metadata_Type"/>

<xs:complexType name="MD_Metadata_Type">
  <xs:annotation>
    <xs:documentation>
     Extends the metadata element to include revisionDate
    </xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="gmd:MD_Metadata_Type">
      <xs:sequence>
        <xs:element name="revisionDate" type="gco:Date_PropertyType"
                    minOccurs="0"/>
      </xs:sequence>
```

```
      <xs:attribute ref="gco:isoType" use="required"
                    fixed="gmd:MD_Metadata"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

</xs:schema>
```

In short, we have defined a new element mcp:MD_Metadata with type mcp:MD_Metadata_Type, which is an extension of gmd:MD_Metadata_Type. By extension, we mean that the new type includes all of the elements of the old type plus one new element, mcp:revisionDate. A mandatory attribute (gco:isoType) is also attached to mcp:MD_Metadata with a fixed value set to the name of the element that we extended (gmd:MD_Metadata).

By defining the profile in this way, it is not necessary to modify the underlying ISO19139 schemas. So the schema directory for the MCP essentially consists of the extensions plus the base ISO19139 schemas. One possible directory structure is as follows:

```
extensions  gco  gmd  gml  gmx  gsr  gss  gts  resources  srv  xlink
```

The extensions directory contains a single file mcpExtensions.xsd, which imports the gmd namespace. The remaining directories are the ISO19139 base schemas.

The schema.xsd file, which is the file GeoNetwork looks for, will import the mcpExtensions.xsd file and any other namespaces not imported as part of the base ISO19139 schema. It is shown as follows:

```
<xs:schema targetNamespace="http://bluenet3.antcrc.utas.edu.au/mcp"
           elementFormDefault="qualified"
         xmlns:xs="http://www.w3.org/2001/XMLSchema"
         xmlns:mcp="http://bluenet3.antcrc.utas.edu.au/mcp"
         xmlns:gmd="http://www.isotc211.org/2005/gmd"
         xmlns:gmx="http://www.isotc211.org/2005/gmx"
         xmlns:srv="http://www.isotc211.org/2005/srv">
  <xs:include schemaLocation="schema/extensions/mcpExtensions.xsd"/>
  <!-- this is a logical place to include any additional schemas that are
       related to ISO19139 including ISO19119 -->
  <xs:import namespace="http://www.isotc211.org/2005/srv"
             schemaLocation="schema/srv/srv.xsd"/>
  <xs:import namespace="http://www.isotc211.org/2005/gmx"
             schemaLocation="schema/gmx/gmx.xsd"/>
</xs:schema>
```

At this stage, our new GeoNetwork plugin schema for MCP contains:

```
schema-conversions.xml  schema-ident.xml  schema.xsd  schema
```

4.3.4 Creating the extract-... XSLTs

GeoNetwork needs to extract certain information from a metadata record and translate it into a common, simplified XML structure that is independent of the metadata schema. Rather than do this with Java coded XPaths, XSLTs are used to process the XML and return the common, simplified XML structure.

The three xslts we'll create are:

- **extract-date-modified.xsl** - this XSLT processes the metadata record and extracts the date the metadata record was last modified. For the MCP, this information is held in the mcp:revisionDate

element which is a child of mcp:MD_Metadata. The easiest way to create this for MCP is to copy extract-date-modified.xsl from the iso19139 schema and modify it to suit the MCP namespace and to use mcp:revisionDate in place of gmd:dateStamp.

- **extract-gml.xsl** - this XSLT processes the metadata record and extracts the spatial extent as a gml GeometryCollection element. The gml is passed to geotools for insertion into the spatial index (either a shapefile or a spatial database). For ISO19115/19139 and profiles, this task is quite easy because spatial extents (apart from the bounding box) are encoded as gml in the metadata record. Again, the easiest way to create this for the MCP is to copy extract-gml.xsd from the iso19139 schema ad modify it to suit the MCP namespace.

An example bounding box fragment from an MCP metadata record is:

```
<gmd:extent>
  <gmd:EX_Extent>
    <gmd:geographicElement>
      <gmd:EX_GeographicBoundingBox>
        <gmd:westBoundLongitude>
          <gco:Decimal>112.9</gco:Decimal>
        </gmd:westBoundLongitude>
        <gmd:eastBoundLongitude>
          <gco:Decimal>153.64</gco:Decimal>
        </gmd:eastBoundLongitude>
        <gmd:southBoundLatitude>
          <gco:Decimal>-43.8</gco:Decimal>
        </gmd:southBoundLatitude>
        <gmd:northBoundLatitude>
          <gco:Decimal>-9.0</gco:Decimal>
        </gmd:northBoundLatitude>
      </gmd:EX_GeographicBoundingBox>
    </gmd:geographicElement>
  </gmd:EX_Extent>
</gmd:extent>
```

Running extract-gml.xsl on the metadata record that contains this XML will produce:

```
<gml:GeometryCollection xmlns:gml="http://www.opengis.net/gml">
  <gml:Polygon>
    <gml:exterior>
      <gml:LinearRing>
        <gml:coordinates>
          112.9,-9.0, 153.64,-9.0, 153.64,-43.8, 112.9,-43.8, 112.9,-9.0
        </gml:coordinates>
      </gml:LinearRing>
    </gml:exterior>
  </gml:Polygon>
</gml:GeometryCollection>
```

If there is more than one extent in the metadata record, then they should also appear in this gml:GeometryCollection element.

To find out more about gml, see Lake, Burggraf, Trninic and Rae, "GML Geography Mark-Up Language, Foundation for the Geo-Web", Wiley, 2004.

Finally, a note on projections. It is possible to have bounding polygons in an MCP record in a projection other than EPSG:4326. GeoNetwork transforms all projections known to GeoTools (and encoded in a form that GeoTools understands) to EPSG:4326 when writing the spatial extents to the shapefile or spatial database.

- **extract-uuid.xsl** - this XSLT processes the metadata record and extracts the identifier for the record. For the MCP and base ISO standard, this information is held in the gmd:fileIdentifier element which is a child of mcp:MD_Metadata.

These xslts can be tested by running them on a metadata record from the schema. You should use the saxon xslt processor. For example:

```
java -jar INSTALL_DIR/web/geonetwork/WEB-INF/lib/saxon-9.1.0.8b-patch.jar
     -s testmcp.xml -o output.xml extract-gml.xsl
```

At this stage, our new GeoNetwork plugin schema for MCP contains:

```
extract-date-modified.xsl  extract-gml.xsd   extract-uuid.xsl
schema-conversions.xml  schema-ident.xml  schema.xsd  schema
```

4.3.5 Creating the localized strings in the loc directory

The loc directory contains localized strings specific to this schema, arranged by language abbreviation in sub-directories.

You should provide localized strings in whatever languages you expect your schema to be used in.

Localized strings for this schema can be used in the presentation xslts and schematron error messages. For the presentation xslts:

- codelists for controlled vocabulary fields should be in loc/<language_abbreviation>/codelists.xml eg. `loc/eng/codelists.xml`
- label strings that replace XML element names with more intelligible/alternative phrases and rollover help strings should be in loc/<language_abbreviation>/labels.xml eg. `loc/eng/labels.xml`.
- all other localized strings should be in loc/<language_abbreviation>/strings.xml eg. `loc/eng/strings.xml`

Note that because the MCP is a profile of ISO19115/19139 and we have followed the GeoNetwork naming convention for profiles, we need only include the labels and codelists that are specific to the MCP or that we want to override. Other labels and codelists will be retrieved from the base schema iso19139.

More on codelists.xml

Typically codelists are generated from enumerated lists in the metadata schema XSDs such as the following from http://www.isotc211.org/2005/gmd/identification.xsd for gmd:MD_TopicCategoryCode in the iso19139 schema:

```
<xs:element name="MD_TopicCategoryCode" type="gmd:MD_TopicCategoryCode_Type"/>

<xs:simpleType name="MD_TopicCategoryCode_Type">
   <xs:restriction base="xs:string">
     <xs:enumeration value="farming"/>
     <xs:enumeration value="biota"/>
     <xs:enumeration value="boundaries"/>
     <xs:enumeration value="climatologyMeteorologyAtmosphere"/>
     <xs:enumeration value="economy"/>
     <xs:enumeration value="elevation"/>
```

```
      <xs:enumeration value="environment"/>
      <xs:enumeration value="geoscientificInformation"/>
      <xs:enumeration value="health"/>
      <xs:enumeration value="imageryBaseMapsEarthCover"/>
      <xs:enumeration value="intelligenceMilitary"/>
      <xs:enumeration value="inlandWaters"/>
      <xs:enumeration value="location"/>
      <xs:enumeration value="oceans"/>
      <xs:enumeration value="planningCadastre"/>
      <xs:enumeration value="society"/>
      <xs:enumeration value="structure"/>
      <xs:enumeration value="transportation"/>
      <xs:enumeration value="utilitiesCommunication"/>
    </xs:restriction>
  </xs:simpleType>
```

The following is part of the codelists.xml entry manually created for this element:

```
<codelist name="gmd:MD_TopicCategoryCode">
  <entry>
    <code>farming</code>
    <label>Farming</label>
    <description>Rearing of animals and/or cultivation of plants. Examples: agriculture,
      irrigation, aquaculture, plantations, herding, pests and diseases affecting crops
      livestock</description>
  </entry>
  <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
  <entry>
    <code>biota</code>
    <label>Biota</label>
    <description>Flora and/or fauna in natural environment. Examples: wildlife, vegetati
      biological sciences, ecology, wilderness, sealife, wetlands, habitat</description>
  </entry>
  <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
  <entry>
    <code>boundaries</code>
    <label>Boundaries</label>
    <description>Legal land descriptions. Examples: political and administrative
    boundaries</description>
  </entry>

  .....

</codelist>
```

The codelists.xml file maps the enumerated values from the XSD to a localized label and a description via the code element.

A localized copy of codelists.xml is made available on an XPath to the presentation XSLTs eg. /root/gui/schemas/iso19139/codelist for the iso19139 schema.

The XSLT metadata.xsl which contains templates used by all metadata schema presentation XSLTs, handles the creation of a select list/drop down menu in the editor and display of the code and description in the metadata viewer.

The iso19139 schema has additional codelists that are managed external to the XSDs in catalog/vocabulary files such as http://www.isotc211.org/2005/resources/Codelist/gmxCodelists.xml These have also been added to the codelists.xml file so that they can be localized, overridden in profiles and

include an extended description to provide more useful information when viewing the metadata record.

The iso19139 schema has additional templates in its presentation xslts to handlese codelists because they are specific to that schema. These are discussed in the section on presentation XSLTs later in this manual.

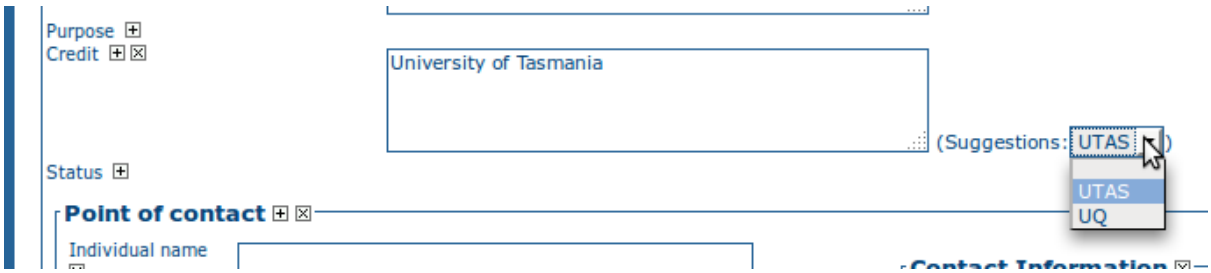
More on labels.xml

A localized copy of labels.xml is made available on an XPath to the presentation XSLTs eg. /root/gui/schemas/iso19139/labels for the iso19139 schema.

The `labels.xml` file can also be used to provide helper values in the form of a drop down/select list for free text fields. As an example:

```
<element name="gmd:credit" id="27.0">
  <label>Credit</label>
  <description>Recognition of those who contributed to the resource(s)</description>
  <helper>
    <option value="University of Tasmania">UTAS</option>
    <option value="University of Queensland">UQ</option>
  </helper>
</element>
```

This would result in the Editor (through the XSLT metadata.xsl) displaying the credit field with these helper options listed beside it in a drop down/select menu something like the following:



More on strings.xml

A localized copy of `strings.xml` is made available on an XPath to the presentation XSLTs eg. /root/gui/schemas/iso19139/strings for the iso19139 schema.

After adding the localized strings, our new GeoNetwork plugin schema for MCP contains:

```
extract-date-modified.xsl  extract-gml.xsd  extract-uuid.xsl
loc  present  schema-conversions.xml  schema-ident.xml  schema.xsd
schema
```

4.3.6 Creating the presentations XSLTs in the present directory

Each metadata schema should contain XSLTs that display and possibly edit metadata records that belong to the schema. These XSLTs are held in the `present` directory.

To be be used in the XSLT include/import hierarchy these XSLTs must follow a naming convention: metadata-<schema-name>.xsl. So for example, the presentation xslt for the iso19139 schema is

`metadata-iso19139.xsl`. For the MCP, since our schema name is iso19139.mcp, the presentation XSLT would be called `metadata-iso19193.mcp.xsl`.

Any XSLTs included by the presentation XSLT should also be in the present directory (this is a convention for clarity - it is not mandatory as include/import URLs can be mapped in the oasis-catalog.xml for the schema to other locations).

There are certain XSLT templates that the presentation XSLT must have:

- the **main** template, which must be called: metadata-<schema-name>. For the MCP profile of iso19139 the main template would look like the following (taken from metadata-iso19139.mcp.xsl):

```
<xsl:template name="metadata-iso19139.mcp">
  <xsl:param name="schema"/>
  <xsl:param name="edit" select="false()"/>
  <xsl:param name="embedded"/>

  <xsl:apply-templates mode="iso19139" select="." >
    <xsl:with-param name="schema" select="$schema"/>
    <xsl:with-param name="edit"   select="$edit"/>
    <xsl:with-param name="embedded" select="$embedded" />
  </xsl:apply-templates>
</xsl:template>
```

Analyzing this template:

1. The name=”metadata-iso19139.mcp” is used by the main element processing template in metadata.xsl: elementEP. The main metadata services, show and edit, end up calling metadata-show.xsl and metadata-edit.xsl respectively with the metadata record passed from the Java service. Both these XSLTs, process the metadata record by applying the elementEP template from metadata.xsl to the root element. The elementEP template calls this main schema template using the schema name iso19139.mcp.
2. The job of this main template is set to process all the elements of the metadata record using templates declared with a mode name that matches the schema name or the name of the base schema (in this case iso19139). This modal processing is to ensure that only templates intended to process metadata elements from this schema or the base schema are applied. The reason for this is that almost all profiles change or add a small number of elements to those in the base schema. So most of the metadata elements in a profile can be processed in the mode of the base schema. We’ll see later in this section how to override processing of an element in the base schema.

- a **completeTab** template, which must be called: <schema-name>CompleteTab. This template will display all tabs, apart from the ‘default’ (or simple mode) and the ‘XML View’ tabs, in the left hand frame of the editor/viewer screen. Here is an example for the MCP:

```
<xsl:template name="iso19139.mcpCompleteTab">
  <xsl:param name="tabLink"/>

  <xsl:call-template name="displayTab"> <!-- non existent tab - by profile -->
    <xsl:with-param name="tab"     select="''"/>
    <xsl:with-param name="text"    select="/root/gui/strings/byGroup"/>
    <xsl:with-param name="tabLink" select="''"/>
  </xsl:call-template>

  <xsl:call-template name="displayTab">
    <xsl:with-param name="tab"     select="'mcpMinimum'"/>
    <xsl:with-param name="text"    select="/root/gui/strings/iso19139.mcp/mcpMinimum"/>
```

```
    <xsl:with-param name="indent"  select="'   '"/>
    <xsl:with-param name="tabLink" select="$tabLink"/>
  </xsl:call-template>

  <xsl:call-template name="displayTab">
    <xsl:with-param name="tab"     select="'mcpCore'"/>
    <xsl:with-param name="text"    select="/root/gui/strings/iso19139.mcp/mcpCore"/>
    <xsl:with-param name="indent"  select="'   '"/>
    <xsl:with-param name="tabLink" select="$tabLink"/>
  </xsl:call-template>

  <xsl:call-template name="displayTab">
    <xsl:with-param name="tab"     select="'complete'"/>
    <xsl:with-param name="text"    select="/root/gui/strings/iso19139.mcp/mcpAll"/>
    <xsl:with-param name="indent"  select="'   '"/>
    <xsl:with-param name="tabLink" select="$tabLink"/>
  </xsl:call-template>

  ...... (same as for iso19139CompleteTab in
 GEONETWORK_DATA_DIR/schema_plugins/iso19139/present/
 metadata-iso19139.xsl) ......

</xsl:template>
```

This template is called by the template named "tab" (which also adds the "default" and "XML View" tabs) in `INSTALL_DIR/web/geonetwork/xsl/metadata-tab-utils.xsl` using the schema name. That XSLT also has the code for the "displayTab" template.

'mcpMinimum', 'mcpCore', 'complete' etc are the names of the tabs. The name of the current or active tab is stored in the global variable "currTab" available to all presentation XSLTs. Logic to decide what to display when a particular tab is active should be contained in the root element processing tab.

- a **root element** processing tab. This tab should match on the root element of the metadata record. For example, for the iso19139 schema:

```
<xsl:template mode="iso19139" match="gmd:MD_Metadata">
  <xsl:param name="schema"/>
  <xsl:param name="edit"/>
  <xsl:param name="embedded"/>

  <xsl:choose>

  <!-- metadata tab -->
  <xsl:when test="$currTab='metadata'">
    <xsl:call-template name="iso19139Metadata">
      <xsl:with-param name="schema" select="$schema"/>
      <xsl:with-param name="edit"   select="$edit"/>
    </xsl:call-template>
  </xsl:when>

  <!-- identification tab -->
  <xsl:when test="$currTab='identification'">
    <xsl:apply-templates mode="elementEP" select="gmd:identificationInfo|geonet:child[st
      <xsl:with-param name="schema" select="$schema"/>
      <xsl:with-param name="edit"   select="$edit"/>
    </xsl:apply-templates>
  </xsl:when>
```

```
  .........

</xsl:template>
```

This template is basically a very long "choose" statement with "when" clauses that test the value of the currently defined tab (in global variable currTab). Each "when" clause will display the set of metadata elements that correspond to the tab definition using "elementEP" directly (as in the "when" clause for the 'identification' tab above) or via a named template (as in the 'metadata' tab above). For the MCP our template is similar to the one above for iso19139, except that the match would be on "mcp:MD_Metadata" (and the processing mode may differ - see the section 'An alternative XSLT design for profiles' below for more details).

- a **brief** template, which must be called: <schema-name>Brief. This template processes the metadata record and extracts from it a format neutral summary of the metadata for purposes such as displaying the search results. Here is an example for the eml-gbif schema (because it is fairly short!):

```
<xsl:template match="eml-gbifBrief">
 <xsl:for-each select="/metadata/*[1]">
  <metadata>
    <title><xsl:value-of select="normalize-space(dataset/title[1])"/></title>
    <abstract><xsl:value-of select="dataset/abstract"/></abstract>

    <xsl:for-each select="dataset/keywordSet/keyword">
      <xsl:copy-of select="."/>
    </xsl:for-each>

    <geoBox>
        <westBL><xsl:value-of select="dataset/coverage/geographicCoverage/boundingCoordi
        <eastBL><xsl:value-of select="dataset/coverage/geographicCoverage/boundingCoordi
        <southBL><xsl:value-of select="dataset/coverage/geographicCoverage/boundingCoord
        <northBL><xsl:value-of select="dataset/coverage/geographicCoverage/boundingCoord
    </geoBox>
    <xsl:copy-of select="geonet:info"/>
  </metadata>
 </xsl:for-each>
</xsl:template>
```

Analyzing this template:

1. The template matches on an element eml-gbifBrief, created by the mode="brief" template in metadata-utils.xsl. The metadata record will be the first child in the /metadata XPath.
2. Then process metadata elements to produce a flat XML structure that is used by search-results-xhtml.xsl to display a summary of the metadata record found by a search.

Once again, for profiles of an existing schema, it makes sense to use a slighlty different approach so that the profile need not duplicate templates. Here is an example from metadata-iso19139.mcp.xsl:

```
<xsl:template match="iso19139.mcpBrief">
  <metadata>
    <xsl:for-each select="/metadata/*[1]">
      <!-- call iso19139 brief -->
      <xsl:call-template name="iso19139-brief"/>
      <!-- now brief elements for mcp specific elements -->
      <xsl:call-template name="iso19139.mcp-brief"/>
    </xsl:for-each>
```

```
  </metadata>
</xsl:template>
```

This template splits the processing between the base iso19139 schema and a brief template that handles elements specific to the profile. This assumes that:

1. The base schema has separated the <metadata> element from the remainder of its brief processing so that it can be called by profiles
2. The profile includes links to equivalent elements that can be used by the base schema to process common elements eg. for ISO19139, elements in the profile have gco:isoType attributes that give the name of the base element and can be used in XPath matches such as "gmd:MD_DataIdentification|*[@gco:isoType='gmd:MD_DataIdentification']".

- templates that match on elements specific to the schema. Here is an example from the eml-gbif schema:

```
<!-- keywords are processed to add thesaurus name in brackets afterwards
     in view mode -->

<xsl:template mode="eml-gbif" match="keywordSet">
  <xsl:param name="schema"/>
  <xsl:param name="edit"/>

  <xsl:choose>
    <xsl:when test="$edit=false()">
      <xsl:variable name="keyword">
        <xsl:for-each select="keyword">
          <xsl:if test="position() &gt; 1">,  </xsl:if>
          <xsl:value-of select="."/>
        </xsl:for-each>
        <xsl:if test="keywordThesaurus">
          <xsl:text> (</xsl:text>
          <xsl:value-of select="keywordThesaurus"/>
          <xsl:text>)</xsl:text>
        </xsl:if>
      </xsl:variable>
      <xsl:apply-templates mode="simpleElement" select=".">
        <xsl:with-param name="schema" select="$schema"/>
        <xsl:with-param name="edit"   select="$edit"/>
        <xsl:with-param name="text"    select="$keyword"/>
      </xsl:apply-templates>
    </xsl:when>
    <xsl:otherwise>
      <xsl:apply-templates mode="complexElement" select=".">
        <xsl:with-param name="schema" select="$schema"/>
        <xsl:with-param name="edit"   select="$edit"/>
      </xsl:apply-templates>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Analyzing this template:

1. In view mode the individual keywords from the set are concatenated into a comma separated string with the name of the thesaurus in brackets at the end.
2. In edit mode, the keywordSet is handled as a complex element ie. the user can add individual

keyword elements with content and a single thesaurus name.

3. This is an example of the type of processing that can be done on an element in a metadata record.

For profiles, templates for elements can be defined in the same way except that the template will process in the mode of the base schema. Here is an example showing the first few lines of a template for processing the mcp:revisionDate element:

```
<xsl:template mode="iso19139" match="mcp:revisionDate">
   <xsl:param name="schema"/>
   <xsl:param name="edit"/>

   <xsl:choose>
     <xsl:when test="$edit=true()">
       <xsl:apply-templates mode="simpleElement" select=".">
         <xsl:with-param name="schema"  select="$schema"/>
         <xsl:with-param name="edit"   select="$edit"/>

   ......
```

If a template for a profile is intended to override a template in the base schema, then the template can be defined in the presentation XSLT for the profile with a priority attribute set to a high number and an XPath condition that ensures the template is processed for the profile only. For example in the MCP, we can override the handling of gmd:EX_GeographicBoundingBox in metadata-iso19139.xsl by defining a template in metadata-iso19139.mcp.xsl as follows:

```
<xsl:template mode="iso19139" match="gmd:EX_GeographicBoundingBox[starts-with(//geonet:i

......
```

Finally, a profile may also extend some of the existing codelists in the base schema. These extended codelists should be held in a localized codelists.xml. As an example, in iso19139 these codelists are often attached to elements like the following:

```
<gmd:role>
  <gmd:CI_RoleCode codeList="http://www.isotc211.org/2005/resources/Codelist/gmxCodelist
</gmd:role>
```

Templates for handling these elements are in the iso19139 presentation XSLT `GEONETWORK_DATA_DIR/schema_plugins/iso19139/present/metadata-iso19139.xsl`. These templates use the name of the element (eg. gmd:CI_RoleCode) and the codelist XPath (eg. /root/gui/schemas/iso19139/codelists) to build select list/drop down menus when editing and to display a full description when viewing. See templates near the template named 'iso19139Codelist'. These templates can handle the extended codelists for any profile because they:

- match on any element that has a child element with attribute codeList
- use the schema name in the codelists XPath
- fall back to the base iso19139 schema if the profile codelist doesn't have the required codelist

However, if you don't need localized codelists, it is often easier and more direct to extract codelists directly from the `gmxCodelists.xml` file. This is in fact the solution that has been adopted for the MCP. The `gmxCodelists.xml` file is included in the presentation xslt for the MCP using a statement like:

```
<xsl:variable name="codelistsmcp"
              select="document('../schema/resources/Codelist/gmxCodelists.xml')"/>
```

Check the codelist handling templates in `metadata-iso19139.mcp.xsl` to see how this works.

An alternative XSLT design for profiles

In all powerful languages there will be more than one way to achieve a particular goal. This alternative XSLT design is for processing profiles. The idea behind the alternative is based on the following observations about the GeoNetwork XSLTs:

1. All elements are initially processed by apply-templates in mode "elementEP".
2. The template "elementEP" (see `INSTALL_DIR/web/geonetwork/xsl/metadata.xsl`) eventually calls the **main** template of the schema/profile.
3. The main template can initially process the element in a mode particular to the profile and if this is not successful (ie. no template matches and thus no HTML elements are returned), process the element in the mode of the base schema.

The advantage of this design is that overriding a template for an element in the base schema does not need the priority attribute or an XPath condition check on the schema name.

Here is an example for the MCP (iso19139.mcp) with base schema iso19139:

- the **main** template, which must be called: metadata-iso19139.mcp.xsl:

```
<!-- main template - the way into processing iso19139.mcp -->
<xsl:template name="metadata-iso19139.mcp">
  <xsl:param name="schema"/>
  <xsl:param name="edit" select="false()"/>
  <xsl:param name="embedded"/>

    <!-- process in profile mode first -->
    <xsl:variable name="mcpElements">
      <xsl:apply-templates mode="iso19139.mcp" select="." >
        <xsl:with-param name="schema" select="$schema"/>
        <xsl:with-param name="edit"   select="$edit"/>
        <xsl:with-param name="embedded" select="$embedded" />
      </xsl:apply-templates>
    </xsl:variable>

    <xsl:choose>

      <!-- if we got a match in profile mode then show it -->
      <xsl:when test="count($mcpElements/*)>0">
        <xsl:copy-of select="$mcpElements"/>
      </xsl:when>

      <!-- otherwise process in base iso19139 mode -->
      <xsl:otherwise>
        <xsl:apply-templates mode="iso19139" select="." >
          <xsl:with-param name="schema" select="$schema"/>
          <xsl:with-param name="edit"   select="$edit"/>
          <xsl:with-param name="embedded" select="$embedded" />
        </xsl:apply-templates>
      </xsl:otherwise>
    </xsl:choose>
</xsl:template>
```

Analyzing this template:

1. The name=”metadata-iso19139.mcp” is used by the main element processing template in metadata.xsl: elementEP. The main metadata services, show and edit, end up calling metadata-show.xsl and metadata-edit.xsl respectively with the metadata record passed from the Java service. Both these XSLTs, process the metadata record by applying the elementEP template from metadata.xsl to the root element. elementEP calls the appropriate main schema template using the schema name.
2. The job of this main template is set to process all the elements of the metadata profile. The processing takes place in one of two modes. Firstly, the element is processed in the profile mode (iso19139.mcp). If a match is found then HTML elements will be returned and copied to the output document. If no HTML elements are returned then the element is processed in the base schema mode, iso19139.

- templates that match on elements specific to the profile have mode iso19139.mcp:

```
<xsl:template mode="iso19139.mcp" match="mcp:taxonomicElement">
  <xsl:param name="schema"/>
  <xsl:param name="edit"/>

  .....
</xsl:template>
```

- templates that override elements in the base schema are processed in the profile mode iso19139.mcp

```
<xsl:template mode="iso19139.mcp" match="gmd:keyword">
  <xsl:param name="schema"/>
  <xsl:param name="edit"/>

  .....
</xsl:template>
```

Notice that the template header of the profile has a simpler design than that used for the original design? Neither the priority attribute or the schema XPath condition is required because we are using a different mode to the base schema.

- To support processing in two modes we need to add a null template to the profile mode iso19139.mcp as follows:

```
<xsl:template mode="iso19139.mcp" match="*|@*"/>
```

This template will match all elements that we don’t have a specific template for in the profile mode iso19139.mcp. These elements will be processed in the base schema mode iso19139 instead because the null template returns nothing (see the main template discussion above).

The remainder of the discussion in the original design relating to tabs etc applies to the alternative design and is not repeated here.

CSW Presentation XSLTs

The CSW server can be asked to provide records in a number of output schemas. The two supported by GeoNetwork are:

- **ogc** - http://www.opengis.net/cat/csw/2.0.2 - a dublin core derivative
- **iso** - http://www.isotc211.org/2005/gmd - ISO19115/19139

From each of these output schemas, a **brief**, **summary** or **full** element set can be requested.

These output schemas and element sets are implemented in GeoNetwork as XSLTs and they are stored in the 'csw' subdirectory of the 'present' directory. The ogc output schema XSLTs are implemented as ogc-brief.xsl, ogc-summary and ogc-full.xsl. The iso output schema XSLTs are implemented as iso-brief.xsl, iso-summary.xsl and iso-full.xsl.

To create these XSLTs for the MCP, the best option is to copy and modify the csw presentation XSLTs from the base schema iso19139.

After creating the presentation XSLTs, our new GeoNetwork plugin schema for MCP contains:

```
extract-date-modified.xsl  extract-gml.xsd  extract-uuid.xsl
loc  present  schema-conversions.xml  schema-ident.xml  schema.xsd
schema
```

4.3.7 Creating the index-fields.xsl to index content from the metadata record

This XSLT indexes the content of elements in the metadata record. The essence of this XSLT is to select elements from the metadata record and map them to lucene index field names. The lucene index field names used in GeoNetwork are as follows:

Lucene Index Field Name	Description
abstract	Metadata abstract
any	Content from all metadata elements (for free text)
changeDate	Date that the metadata record was modified
createDate	Date that the metadata record was created
denominator	Scale denominator in data resolution
download	Does the metadata record have a downloadable resource attached? (0 or 1)
digital	Is the metadata record distributed/available in a digital format? (0 or 1)
eastBL	East bounding box longitude
keyword	Metadata keywords
metadataStandardName	Metadata standard name
northBL	North bounding box latitude
operatesOn	Uuid of metadata record describing dataset that is operated on by a service
orgName	Name of organisation listed in point-of-contact information
parentUuid	Uuid of parent metadata record
paper	Is the metadata record distributed/available in a paper format? (0 or 1)
protocol	On line resource access protocol
publicationDate	Date resource was published
southBL	South bounding box latitude
spatialRepresentationType	vector, raster, etc
tempExtentBegin	Beginning of temporal extent range
tempExtentEnd	End of temporal extent range
title	Metadata title
topicCat	Metadata topic category
type	Metadata hierarchy level (should be dataset if unknown)
westBL	West bounding box longitude

For example, here is the mapping created between the metadata element mcp:revisionDate and the lucene index field changeDate:

```
<xsl:for-each select="mcp:revisionDate/*">
  <Field name="changeDate" string="{string(.)}" store="true" index="true"/>
</xsl:for-each>
```

Notice that we are creating a new XML document. The Field elements in this document are read by GeoNetwork to create a Lucene document object for indexing (see the SearchManager class in the GeoNetwork source).

Once again, because the MCP is a profile of ISO19115/19139, it is probably best to modify `index-fields.xsl` from the schema iso19139 to handle the namespaces and additional elements of the MCP.

At this stage, our new GeoNetwork plugin schema for MCP contains:

```
extract-date-modified.xsl  extract-gml.xsd  extract-uuid.xsl
index-fields.xsl  loc  present  schema-conversions.xml  schema-ident.xml
schema.xsd  schema
```

4.3.8 Creating the sample-data directory

This is a simple directory. Put MEF files with sample metadata in this directory. Make sure they have a `.mef` suffix.

A MEF file is a zip archive with the metadata, thumbnails, file based online resources and an info file describing the contents. The contents of a MEF file are discussed in more detail in the next section of this manual.

Sample data in this directory can be added to the catalog using the Administration menu.

At this stage, our new GeoNetwork plugin schema for MCP contains:

```
extract-date-modified.xsl  extract-gml.xsd  extract-uuid.xsl
index-fields.xsl  loc  present  sample-data schema-ident.xml  schema.xsd
schema
```

4.3.9 Creating schematrons to describe MCP conditions

Schematrons are rules that are used to check conditions and content in the metadata record as part of the two stage validation process used by GeoNetwork.

Schematron rules are created in the schematrons directory that you checked out earlier - see *Preparation* above.

An example rule is:

```
<!-- anzlic/trunk/gml/3.2.0/gmd/spatialRepresentation.xsd-->
<!-- TEST 12 -->
<sch:pattern>
  <sch:title>$loc/strings/M30</sch:title>
  <sch:rule context="//gmd:MD_Georectified">
    <sch:let name="cpd" value="(gmd:checkPointAvailability/gco:Boolean='1' or gmd:checkP
      (not(gmd:checkPointDescription) or count(gmd:checkPointDescription[@gco:nilReason=
    <sch:assert
      test="$cpd = false()"
      >$loc/strings/alert.M30</sch:assert>
    <sch:report
```

```
      test="$cpd = false()"
      >$loc/strings/report.M30</sch:report>
  </sch:rule>
</sch:pattern>
```

As for most of GeoNetwork, the output of this rule can be localized to different languages. The corresponding localized strings are:

```
<strings>

  .....

  <M30>[ISOFTDS19139:2005-TableA1-Row15] - Check point description required if available

  .....

  <alert.M30><div>'checkPointDescription' is mandatory if 'checkPointAvailability' = 1 c

  .....

  <report.M30>Check point description documented.</report.M30>

  .....

</strings>
```

Procedure for adding schematron rules, working within the schematrons directory:

1. Place your schematron rules in 'rules'. Naming convetion is 'schematron-rules-<suffix>.sch' eg. `schematron-rules-iso-mcp.sch`. Place localized strings for the rule assertions into 'rules/loc/<language_prefix>'.
2. Edit `build.xml`.
3. Add a "clean-schema-dir" target for your plugin schema directory. This target will remove the schematron rules from plugin schema directory (basically removes all files with pattern schematron-rules-*.xsl).
4. Add a "compile-schematron" target for your rules - value attribute is the suffix used in the rules name. eg. for `schematron-rules-iso-mcp.sch` the value attribute should be "iso-mcp". This target will turn the .sch schematron rules into an XSLT using the saxon XSLT engine and 'resources/iso_svrl_for_xslt2.xsl'.
5. Add a "publish-schematron" target. This target copies the compiled rules (in XSLT form) into the plugin schema directory.
6. Run 'ant' to create the schematron XSLTs.

At this stage, our new GeoNetwork plugin schema for MCP contains:

```
extract-date-modified.xsl  extract-gml.xsd  extract-uuid.xsl
index-fields.xsl  loc  present  sample-data  schema-conversions.xml
schema-ident.xml  schema.xsd  schema  schematron-rules-iso-mcp.xsl
```

4.3.10 Adding the components necessary to create and edit MCP metadata

So far we have added all the components necessary for GeoNetwork to identify, view and validate MCP metadata records. Now we will add the remaining components necessary to create and edit MCP metadata records.

We'll start with the XSLTs that set the content of various elements in the MCP metadata records.

Creating set-uuid.xsl

- **set-uuid.xsl** - this XSLT takes as a parameter the UUID of the metadata record and writes it into the appropriate element of the metadata record. For the MCP this element is the same as the base ISO schema (called iso19139 in GeoNetwork), namely gmd:fileIdentifier. However, because the MCP uses a different namespace on the root element, this XSLT needs to be modified.

Creating the extract, set and unset thumbnail XSLTs

If your metadata record can have a thumbnail or browse graphic link, then you will want to add XSLTs that extract, set and unset this information so that you can use the GeoNetwork thumbnail editing interface.

The three XSLTs that support this interface are:

- **extract-thumbnails.xsl** - this XSLT extracts the thumbnails/browse graphics from the metadata record, turning it into generic XML that is the same for all metadata schemas. The elements need to have content that GeoNetwork understands. The following is an example of what the thumbnail interface for iso19139 expects (we'll duplicate this requirement for MCP):

```
<gmd:graphicOverview>
  <gmd:MD_BrowseGraphic>
    <gmd:fileName>
      <gco:CharacterString>bluenet_s.png</gco:CharacterString>
    </gmd:fileName>
    <gmd:fileDescription>
      <gco:CharacterString>thumbnail</gco:CharacterString>
    </gmd:fileDescription>
    <gmd:fileType>
      <gco:CharacterString>png</gco:CharacterString>
    </gmd:fileType>
  </gmd:MD_BrowseGraphic>
</gmd:graphicOverview>
<gmd:graphicOverview>
  <gmd:MD_BrowseGraphic>
    <gmd:fileName>
      <gco:CharacterString>bluenet.png</gco:CharacterString>
    </gmd:fileName>
    <gmd:fileDescription>
      <gco:CharacterString>large_thumbnail</gco:CharacterString>
    </gmd:fileDescription>
    <gmd:fileType>
      <gco:CharacterString>png</gco:CharacterString>
    </gmd:fileType>
  </gmd:MD_BrowseGraphic>
</gmd:graphicOverview>
```

When `extract-thumbnails.xsl` is run, it creates a small XML hierarchy from this information which looks something like the following:

```
<thumbnail>
  <large>
    bluenet.png
  </large>
  <small>
    bluenet_s.png
  </small>
</thumbnail>
```

- **set-thumbnail.xsl** - this XSLT does the opposite of extract-thumbnails.xsl. It takes the simplified, common XML structure used by GeoNetwork to describe the large and small thumbnails and creates the elements of the metadata record that are needed to represent them. This is a slightly more complex XSLT than extract-thumbnails.xsl because the existing elements in the metadata record need to be retained and the new elements need to be created in their correct places.
- **unset-thumbnail.xsl** - this XSLT targets and removes elements of the metadata record that describe a particular thumbnail. The remaining elements of the metadata record are retained.

Because the MCP is a profile of ISO19115/19139, the easiest path to creating these XSLTs is to copy them from the iso19139 schema and modify them for the changes in namespace required by the MCP.

Creating the update-... XSLTs

- **update-child-from-parent-info.xsl** - this XSLT is run when a child record needs to have content copied into it from a parent record. It is an XSLT that changes the content of a few elements and leaves the remaining elements untouched. The behaviour of this XSLT would depend on which elements of the parent record will be used to update elements of the child record.
- **update-fixed-info.xsl** - this XSLT is run after editing to fix certain elements and content in the metadata record. For the MCP there are a number of actions we would like to take to 'hard-wire' certain elements and content. To do this the XSLT the following processing logic:

```
if the element is one that we want to process then
  add a template with a match condition for that element and process it
else copy the element to output
```

Because the MCP is a profile of ISO19115/19139, the easiest path to creating this XSLT is to copy update-fixed-info.xsl from the iso19139 schema and modify it for the changes in namespace required by the MCP and then to include the processing we want.

A simple example of MCP processing is to make sure that the gmd:metadataStandardName and gmd:metadataStandardVersion elements have the content needed to ensure that the record is recognized as MCP. To do this we can add two templates as follows:

```
<xsl:template match="gmd:metadataStandardName" priority="10">
  <xsl:copy>
    <gco:CharacterString>Australian Marine Community Profile of ISO 19115:2005/19139</gc
  </xsl:copy>
</xsl:template>

<xsl:template match="gmd:metadataStandardVersion" priority="10">
  <xsl:copy>
    <gco:CharacterString>MCP:BlueNet V1.5</gco:CharacterString>
```

```
  </xsl:copy>
</xsl:template>
```

Processing by `update-fixed-info.xsl` can be enabled/disabled using the *Automatic Fixes* check box in the System Configuration menu. By default, it is enabled.

Some important tasks handled in `upgrade-fixed-info.xsl`:

- creating URLs for metadata with attached files (eg. onlineResources with 'File for download' in iso19139)
- setting date stamp/revision date
- setting codelist URLs to point to online ISO codelist catalogs
- adding default spatial reference system attributes to spatial extents

A specific task required for the MCP `update-fixed-info.xsl` was to automatically create an online resource with a URL pointing to the metadata.show service with parameter set to the metadata uuid. This required some changes to the update-fixed-info.xsl supplied with iso19139. In particular:

- the parent elements may not be present in the metadata record
- processing of the online resource elements for the metadata point of truth URL should not interfere with other processing of online resource elements

Rather than describe the individual steps required to implement this and the decisions required in the XSLT language, take a look at the `update-fixed-info.xsl` already present for the MCP schema in the iso19139.mcp directory and refer to the dot points above.

Creating the templates directory

This is a simple directory. Put XML metadata files to be used as templates in this directory. Make sure they have a `.xml` suffix. Templates in this directory can be added to the catalog using the Administration menu.

Editor behaviour: Adding schema-suggestions.xml and schema-substitutes.xml

- **schema-suggestions.xml** - The default behaviour of the GeoNetwork advanced editor when building the editor forms is to show elements that are not in the metadata record as unexpanded elements. To add these elements to the record, the user will have to click on the '+' icon next to the element name. This can be tedious especially as some metadata standards have elements nested in others (ie. complex elements). The schema-suggestions.xml file allows you to specify elements that should be automatically expanded by the editor. An example of this is the online resource information in the ISO19115/19139 standard. If the following XML was added to the `schema-suggestions.xml` file:

```
<field name="gmd:CI_OnlineResource">
  <suggest name="gmd:protocol"/>
  <suggest name="gmd:name"/>
  <suggest name="gmd:description"/>
</field>
```

The effect of this would be that when an online resource element was expanded, then input fields for the protocol (a drop down/select list), name and description would automatically appear in the editor.

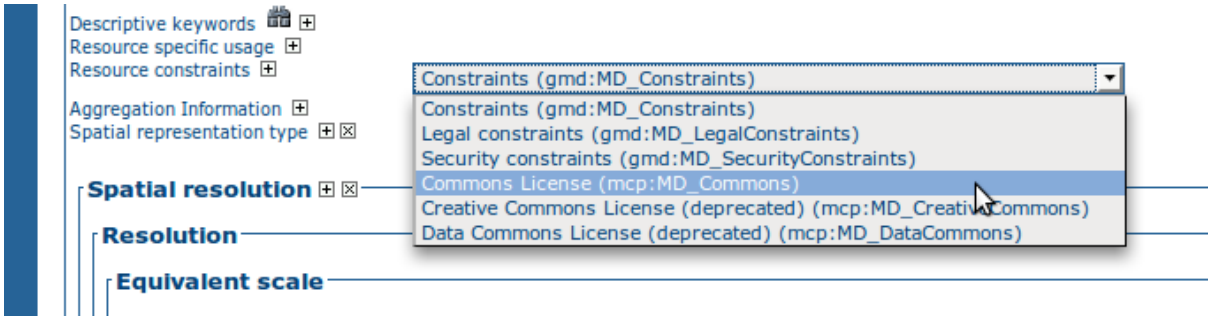
Once again, a good place to start when building a `schema-suggestions.xml` file for the MCP is the `schema-suggestions.xml` file for the iso19139 schema.

- **schema-substitutes.xml** - Recall from the earlier *Creating the schema directory and schema.xsd file* section, that the method we used to extend the base ISO19115/19139 schemas is to extend the base type, define a new element with the extended base type and allow the new element to substitute for the base element. So for example, in the MCP, we want to add a new resource constraint element that holds Creative Commons and other commons type licensing information. This requires that the MD_Constraints type be extended and a new mcp:MD_Commons element be defined which can substitute for gmd:MD_Constraints. This is shown in the following snippet of XSD:

```
<xs:complexType name="MD_CommonsConstraints_Type">
  <xs:annotation>
    <xs:documentation>
      Add MD_Commons as an extension of gmd:MD_Constraints_Type
    </xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="gmd:MD_Constraints_Type">
      <xs:sequence minOccurs="0">
        <xs:element name="jurisdictionLink" type="gmd:URL_PropertyType" minOccurs="1"/>
        <xs:element name="licenseLink" type="gmd:URL_PropertyType" minOccurs="1"/>
        <xs:element name="imageLink" type="gmd:URL_PropertyType" minOccurs="1"/>
        <xs:element name="licenseName" type="gco:CharacterString_PropertyType" minOccurs
        <xs:element name="attributionConstraints" type="gco:CharacterString_PropertyType
        <xs:element name="derivativeConstraints" type="gco:CharacterString_PropertyType'
        <xs:element name="commercialUseConstraints" type="gco:CharacterString_PropertyTy
        <xs:element name="collectiveWorksConstraints" type="gco:CharacterString_Property
        <xs:element name="otherConstraints" type="gco:CharacterString_PropertyType" minO
      </xs:sequence>
      <xs:attribute ref="mcp:commonsType" use="required"/>
      <xs:attribute ref="gco:isoType" use="required" fixed="gmd:MD_Constraints"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="MD_Commons" substitutionGroup="gmd:MD_Constraints" type="mcp:MD_Commor
```

For MCP records, the GeoNetwork editor will show a choice of elements from the substitution group for gmd:MD_Constraints when adding 'Resource Constraints' to the metadata document. This will now include mcp:MD_Commons.

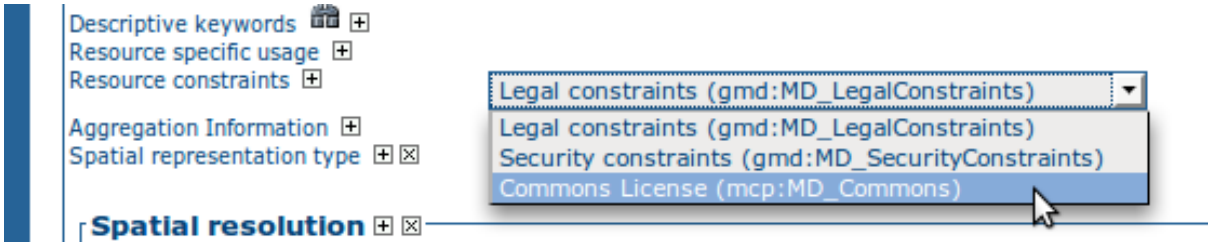


Note that by similar process, two other elements, now deprecated in favour of MD_Commons, were also added as substitutes for MD_Constraints. If it was necessary to constrain the choices shown in this

menu, say to remove the deprecated elements and limit the choices to just legal, security and commons, then this can be done by the following piece of XML in the schema-substitutes.xml file:

```
<field name="gmd:MD_Constraints">
  <substitute name="gmd:MD_LegalConstraints"/>
  <substitute name="gmd:MD_SecurityConstraints"/>
  <substitute name="mcp:MD_Commons"/>
</field>
```

The result of this change is shown below.



Once again, a good place to start when building a schema-substitutes.xml file for the MCP is the schema-substitutes.xml file for the iso19139 schema.

4.3.11 Adding components to support conversion of metadata records to other schemas

Creating the convert directory

If the new GeoNetwork plugin schema is to support on the fly translation of metadata records to other schemas, then the convert directory should be created and populated with appropriate XSLTs.

Supporting OAIPMH conversions

The OAIPMH server in GeoNetwork can deliver metadata records from any of the schemas known to GeoNetwork. It can also be configured to deliver schemas not known to GeoNetwork if an XSLT exists to convert a metadata record to that schema. The file `INSTALL_DIR/web/geonetwork/WEB-INF/config-oai-prefixes.xml` describes the schemas (known as prefixes in OAI speak) that can be produced by an XSLT. A simple example of the content of this file is shown below:

```
<schemas>
  <schema prefix="oai_dc" nsUrl="http://www.openarchives.org/OAI/2.0/"
          schemaLocation="http://www.openarchives.org/OAI/2.0/oai_dc.xsd"/>
</schemas>
```

In the case of the prefix oai_dc shown above, if a schema converter with prefix *oai_dc* exists in the `schema-conversions.xml` file of a GeoNetwork schema, then records that belong to this schema will be transformed and included in OAIPMH requests for the *oai_dc* prefix. See *Creating the schema-conversions.xml file* for more info.

To add oai_dc support for the MCP, the easiest method is to copy oai_dc.xsl from the convert directory of the iso19139 schema, modify it to cope with the different namespaces and additional elements of the MCP and add it to the `schema-conversions.xml` file for the MCP.

CHAPTER 5

Metadata Exchange Format

5.1 Introduction

The metadata exchange format (MEF in short) is a specially designed file format for the purpose of metadata exchange between different platforms. A metadata exported as a MEF can be imported by any platform which is able to understand MEF. This format has been developed with GeoNetwork in mind so the information it contains is mainly related to GeoNetwork. Nevertheless, it can be used as an interoperability format between different platforms.

This format has been designed with the following needs in mind:

1. Export a metadata record for backup purposes
2. Import a metadata record from a previous backup
3. Import a metadata record from a different GeoNetwork version to allow a smooth migration from one version to another.
4. Capture metadata plus thumbnails and any data uploaded with the metadata record.

In the paragraphs below, some terms should be intended as follows:

1. the term actor is used to indicate any system (application, service etc...) that operates on metadata.
2. the term reader will be used to indicate any actor that can import metadata from a MEF file.
3. the term writer will be used to indicate any actor that can generate a MEF file.

5.2 MEF v1 file format

A MEF file is simply a ZIP file which contains the following files:

```
Root
 |
 +--- metadata.xml
 +--- info.xml
 +--- public
 |        +---- all public documents and thumbnails
 +--- private
          +---- all private documents and thumbnails
```

1. *metadata.xml*: this file contains the metadata itself, in XML format. The text encoding of the metadata (eg. UTF-8) is specified in the XML declaration.
2. *info.xml*: this is a special XML file which contains information related to the metadata (metadata about the metadata). Examples of the information in the info.xml file are: creation date, modification date, privileges This information is needed by GeoNetwork.
3. *public*: this is a directory used to store the metadata thumbnails and other public files. There are no restrictions on the image format but it is strongly recommended to use the portable network graphics (PNG), JPEG or GIF format.
4. *private*: this is a directory used to store all data (maps, shape files etc...) uploaded with the metadata in the GeoNetwork editor. Files in this directory are *private* in the sense that authorisation is required to access them. There are no restrictions on the file types that can be stored into this directory.

Any other file or directory present in the MEF file should be ignored by readers that don't recognise them. This allows actors to add custom extensions to the MEF file.

A MEF file can have empty public and private folders depending upon the export format, which can be:

- *simple*: both public and private are omitted.
- *partial*: only public files are provided.
- *full*: both public and private files are provided.

It is recommended to use the .mef extension when naming MEF files.

5.3 MEF v2 file format

MEF version 2 support the following:

- multi-metadata support: more than one metadata record and data can be stored in a single MEF file.
- multi-schema support: be able to store in a single MEF n formats (eg. for an ISO profile, also store a version of that record in the base ISO19115/ISO19139 schema).

Current export services that create MEF files from a metadata record with related records (eg. paent, feature catalog etc), can include these related metadata records in the MEF. See *mef.export*.

MEF v2 format structure is the following:

```
Root
  |
 + 0..n metadata
      |
      +--- metadata
      |     +--- metadata.xml
      |     +--- (optional) metadata.iso19139.xml
      +--- info.xml
      +--- applschema
      |     +--- (optional) metadata.xml (ISO19110 Feature Catalog)
      +--- public
      |     +---- all public documents and thumbnails
      +--- private
            +---- all private documents and thumbnails
```

Note: metadata.iso19139.xml is generated by GeoNetwork actors on export if the metadata record in metadata.xml is an ISO19115/19139 profile. On import, this record may be selected for loading if the ISO19115/19139 profile is not present.

5.4 The info.xml file

This file contains general information about a metadata. It must have an info root element with a mandatory version attribute. This attribute must be in the X.Y form, where X represents the major version and Y the minor one. The purpose of this attribute is to allow future changes of this format maintaining compatibility with older readers. The policy behind the version is this:

1. A change to Y means a minor change. All existing elements in the previous version must be left unchanged: only new elements or attributes may be added. A reader capable of reading version X.Y is also capable of reading version X.Y' with Y'>Y.
2. A change to X means a major change. Usually, a reader of version X.Y is not able to read version X'.Y with X'>X.

The root element must have the following children:

1. *general*: a container for general information. It must have the following children:
    - *uuid*: this is the universally unique identifier assigned to the metadata and must be a valid UUID. This element is optional and, when omitted, the reader should generate one. A metadata without a UUID can be imported several times into the same system without breaking uniqueness constraints. When missing, the reader should also generate the siteId value.
    - *createDate*: This date indicates when the metadata was created.
    - *changeDate*: This date keeps track of the most recent change to the metadata.
    - *siteId*: This is an UUID that identifies the actor that created the metadata and must be a valid UUID. When the UUID element is missing, this element should be missing too. If present, it will be ignored.
    - *siteName*: This is a human readable name for the actor that created the metadata. It must be present only if the siteId is present.
    - *schema*: The name of the schema for the metadata record in metadata.xml. When the MEF is imported by a GeoNetwork actor, this name should be the name of a metadata schema handled by the actor (eg. iso19139). If the GeoNetwork actor does not have such a schema, it may try and select another metadata with a schema that is present (eg. the metadata in metadata-iso19139.xml could be loaded because the iso19139 schema is present).
    - *format*: Indicates the MEF export format. The element's value must belong to the following set: { *simple*, *partial*, *full* }.
    - *localId*: This is an optional element. If present, indicates the id used locally by the sourceId actor to store the metadata. Its purpose is just to allow the reuse of the same local id when reimporting a metadata.
    - *isTemplate*: A boolean field that indicates if this metadata is a template used to create new ones. There is no real distinction between a real metadata and a template but some actors use it to allow fast metadata creation. The value must be: { *true*, *false* }.

- *rating*: This is an optional element. If present, indicates the users' rating of the metadata ranging from 1 (a bad rating) to 5 (an excellent rating). The special value 0 means that the metadata has not been rated yet. Can be used to sort search results.
- *popularity*: Another optional value. If present, indicates the popularity of the metadata. The value must be positive and high values mean high popularity. The criteria used to set the popularity is left to the writer. Its main purpose is to provide a metadata ordering during a search.

2. *categories*: a container for categories associated to this metadata. A category is just a name, like 'audio-video' that classifies the metadata to allow an easy search. Each category is specified by a category element which must have a name attribute. This attribute is used to store the category's name. If there are no categories, the categories element will be empty.
3. *privileges*: a container for privileges associated to this metadata. Privileges are operations that a group (which represents a set of users) can do on a metadata and are specified by a set of group elements. Each one of these, has a mandatory name attribute to store the group's name and a set of operation elements used to store the operations allowed on the metadata. Each operation element must have a name attribute which value must belong to the following set: { *view*, *download*, *notify*, *dynamic*, *featured* }. If there are no groups or the actor does not have the concept of group, the privileges element will be empty. A group element without any operation element must be ignored by readers.
4. *public*: All metadata thumbnails (and any other public file) must be listed here. This container contains a file element for each file. Mandatory attributes of this element are name, which represents the file's name and changeDate, which contains the date of the latest change to the file. The public element is optional but, if present, must contain all the files present in the metadata's public directory and any reader that imports these files must set the latest change date on these using the provided ones. The purpose of this element is to provide more information in the case the MEF format is used for metadata harvesting.
5. *private*: This element has the same purpose and structure of the public element but is related to maps and all other private files.

Any other element or attribute should be ignored by readers that don't understand them. This allows actors to add custom attributes or subtrees to the XML.

5.4.1 Date format

Unless differently specified, all dates in this file must be in the ISO/8601 format. The pattern must be YYYY-MM-DDTHH:mm:SS and the timezone should be the local one. Example of info file:

```
<info version="1.0">
    <general>
        <uuid>0619abc0-708b-eeda-8202-000d98959033</uuid>
        <createDate>2006-12-11T10:33:21</createDate>
        <changeDate>2006-12-14T08:44:43</changeDate>
        <siteId>0619cc50-708b-11da-8202-000d9335906e</siteId>
        <siteName>FAO main site</siteName>
        <schema>iso19139</schema>
        <format>full</format>
        <localId>204</localId>
        <isTemplate>false</isTemplate>
    </general>
    <categories>
```

```
        <category name="maps"/>
        <category name="datasets"/>
    </categories>
    <privileges>
        <group name="editors">
            <operation name="view"/>
            <operation name="download"/>
        </group>
    </privileges>
    <public>
        <file name="small.png" changeDate="2006-10-07T13:44:32"/>
        <file name="large.png" changeDate="2006-11-11T09:33:21"/>
    </public>
    <private>
        <file name="map.zip" changeDate="2006-11-12T13:23:01"/>
    </private>
</info>
```

CHAPTER 6

XML Services

6.1 Calling specifications

6.1.1 Calling XML services

GeoNetwork provides access to its functions through the use of XML services. These services are much like HTML addresses but return XML instead of HTML. The advantage of using XML, is that XML services can be used in machine-to-machine interfaces. As an example, consider the `xml.info` service - *Site Information (xml.info)*: you might have an application which can use this service to get information about GeoNetwork users and metadata schemas in XML. The user information returned by this service could then be used by your application to decide how to authenticate with GeoNetwork so that metadata records from a particular metadata schema could be retrieved from other GeoNetwork XML services and processed by your application.

As a general rule, XML services provided by GeoNetwork usually have a `xml.` prefix in their address. To keep things simple and uniform, GeoNetwork XML services accept XML documents (or convert parameters to XML documents) and return information, status and errors as XML documents (except for a few services that relate to file download and must return binary data).

Request

Each service accepts a set of parameters, which must be embedded in the request. A service can be called using different HTTP methods, depending on the structure of its request:

- **GET** The parameters are sent as part of the URL address. On the server side, these parameters are grouped into a flat XML document with one root and several simple children. A service can be called this way only if the parameters it accepts are not structured. An example of such a request and the parameters encoded in XML is:

```
Url Request:
http://localhost:8080/geonetwork/srv/eng/main.search?hitsPerPage=10&any=

Encoding:
<request>
    <hitsPerPage>10</hitsPerPage>
    <any />
</request>
```

- **POST** There are 3 variants of this method:
  1. **ENCODED** The request has one of the following content types: application/x-www-form-urlencoded or multipart/form-data. The first case is very common when sending web forms while the second one is used to send binary data (usually files) to the server. In these cases, the parameters are not structured so the rules of the GET method applies. Even if the second case could be used to send XML documents, this possibility is not considered on the server side.
  2. **XML** The content type is application/xml. This is the common case when the client is not a browser but a specialised client. The request is a pure XML document in string form, encoded using the encoding specified into the prologue of the XML document. Using this form, any type of request can be made (structured or not) so any service can be called.
  3. **SOAP** The content type is application/soap+xml. SOAP is a simple protocol used to access objects and services using XML. Clients that use this protocol can embed XML requests into a SOAP structure. On the server side, GeoNetwork will remove the SOAP structure and feed the content to the service. Its response will be embedded again into a SOAP structure and sent back to the caller. It makes sense to use this protocol if it is the only protocol understood by the client.

Response

The response of an XML service always has a content type of application/xml (the only exception are those services which return binary data). The document encoding is the one specified in the document prologue which is UTF-8 (all GeoNetwork services return documents in the UTF-8 encoding).

On a GET request, the client can force a SOAP response by adding the application/soap+xml content type to the Accept header parameter.

6.1.2 Exception handling

A response document having an error root element means that the XML service raised an exception. This can happen under several conditions: bad parameters, internal errors et cetera. In this cases the returned XML document has the following structure:

- **error**: This is the root element of the document. It has a mandatory id attribute that represents the identifier of the error. See below for a list of identifier values.
  - **message**: A message related to the error. It can be a short description about the error type or it can contain some other information that details the id code.
  - **class**: The Java class name of the Exception that occurred.
  - **stack**: Execution path from method where Exception occurred to earliest method called by GeoNetwork. Each level in the execution path has an `at` child.
    - **at**: Information about the code being called when the exception occurred. It has the following mandatory attributes:
      - **class** Java class name of the method that was called.
      - **file** Source file where the class is defined.
      - **method** Method name in **class**.

· **line** Source code line number in **file**.

– **object**: An optional container for parameters or other values that caused the exception. In case a parameter is an XML object, this container will contain that object in XML form.

– **request**: A container for request information.

* **language**: Language used when the service was called.

* **service**: Name of the service that was called.

Summary of error ids:

id	Meaning of message element	Meaning of object element
error	General message, human readable	x
bad-format	Reason	x
bad-parameter	Name of the parameter	Parameter value
file-not-found	x	File name
file-upload-too-big	x	x
missing-parameter	Name of the parameter	XML container where the parameter should have been present.
object-not-found	x	Object name
operation-aborted	Reason of abort	If present, the object that caused the abort
operation-not-allowed	x	x
resource-not-found	x	Resource name
service-not-allowed	x	Service name
service-not-found	x	Service name
user-login	User login failed message	User name
user-not-found	x	User id or name
metadata-not-found	The requested metadata was not found	Metadata id

Below is an example of exception generated by the mef.export service. The service complains about a missing parameter, as you can see from the content of the id attribute. The object element contains the xml request with an unknown test parameter while the mandatory UUID parameter (as specified by the message element) is missing.

An example of generated exception:

```
<error>
    <message>UUID</message>
    <class>MissingParameterEx</class>
    <stack>
        <at class="jeeves.utils.Util" file="Util.java" line="66"
            method="getParam"/>
        <at class="org.fao.geonet.services.mef.Export" file="Export.java"
            line="60" method="exec"/>
        <at class="jeeves.server.dispatchers.ServiceInfo" file="ServiceInfo.java"
```

```
            line="226" method="execService"/>
        <at class="jeeves.server.dispatchers.ServiceInfo" file="ServiceInfo.java"
            line="129" method="execServices"/>
        <at class="jeeves.server.dispatchers.ServiceManager" file="ServiceManager.java"
            line="370" method="dispatch"/>
    </stack>
    <object>
        <request>
            <asd>ee</asd>
        </request>
    </object>
    <request>
        <language>en</language>
        <service>mef.export</service>
    </request>
</error>
```

6.2 Login and Logout services

6.2.1 Login services

GeoNetwork standard login (xml.user.login)

The **xml.user.login** service is used to authenticate the user in GeoNetwork. Authenticated users can use XML services that require authentication such as those used to maintain group or user information.

Request

Parameters:

- **username** (mandatory): Login for the user to authenticate
- **password** (mandatory): Password for the user to authenticate

Login request example:

```
Url:
http://localhost:8080/geonetwork/srv/en/xml.user.login

Mime-type:
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request>
    <username>admin</username>
    <password>admin</password>
</request>
```

Response

When user authentication is successful HTTP status code 200 is returned along with an XML response as follows:

```
<ok/>
```

If the response headers are examined, they will look something like the following::

```
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: JSESSIONID=1xh3kpownhmjh;Path=/geonetwork
Content-Type: application/xml; charset=UTF-8
Pragma: no-cache
Cache-Control: no-cache
Transfer-Encoding: chunked
```

The authentication process sets the **JSESSIONID** cookie with the authentication token. This token should be sent as part of the request to all services that need authentication.

If the execution of the login request is not successful then an HTTP 500 status code error is returned along with an XML document that describes the exception/what went wrong. An example of such a response is::

```
<error id="user-login">
  <message>User login failed</message>
  <class>UserLoginEx</class>
  <stack>
    <at class="org.fao.geonet.services.login.Login" file="Login.java" line="90" method="
    <at class="jeeves.server.dispatchers.ServiceInfo" file="ServiceInfo.java" line="238"
    <at class="jeeves.server.dispatchers.ServiceInfo" file="ServiceInfo.java" line="141"
    <at class="jeeves.server.dispatchers.ServiceManager" file="ServiceManager.java" line
    <at class="jeeves.server.JeevesEngine" file="JeevesEngine.java" line="621" method="d
    <at class="jeeves.server.sources.http.JeevesServlet" file="JeevesServlet.java" line=
    <at class="jeeves.server.sources.http.JeevesServlet" file="JeevesServlet.java" line=
    <at class="javax.servlet.http.HttpServlet" file="HttpServlet.java" line="727" method
    <at class="javax.servlet.http.HttpServlet" file="HttpServlet.java" line="820" method
    <at class="org.mortbay.jetty.servlet.ServletHolder" file="ServletHolder.java" line="
  </stack>
  <object>admin2</object>
  <request>
    <language>en</language>
    <service>user.login</service>
  </request>
</error>
```

See *Exception handling* for more details.

Errors

- **Missing parameter (error id: missing-parameter)**, when mandatory parameters are not send. Returns 500 HTTP code
- **bad-parameter XXXX**, when an empty username or password is provided. Returns 500 HTTP code
- **User login failed (error id: user-login)**, when login information is not valid. Returns 500 HTTP code

6.2.2 Logout service

Logout (xml.user.logout)

The **xml.user.logout** service clears the user authentication session, removing the **JSESSIONID** cookie.

Request

Parameters:

- **None**:This request requires no parameters however the **JSESSIONID** token obtained from `xml.user.login` should be included as this is the session that will be cleared..

Logout request example:

```
Url:
http://localhost:8080/geonetwork/srv/en/xml.user.logout

Mime-type:
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request/>
```

Response

Logout response example:

```
<ok />
```

6.3 Group services

6.3.1 Group List (xml.info&type=groups)

The **xml.info** service can be used to retrieve the user groups available in GeoNetwork. See *Site Information (xml.info)*.

6.3.2 Group maintenance

Create/update a group (xml.group.create.update)

The **xml.group.create.update** service can be used to create a new group and update the information about an existing group. Only users with **Administrator** profile can create/update groups.

Requires authentication: Yes

Request

Parameters:

- **id**: Group identifier to update. If not provided a new group is created with the name, description and email parameters provided.
- **name**: (mandatory) Name of the group
- **description**: Group description
- **email**: email address for group notifications

Group update request example:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.group.create.update

Mime-type:
application/xml

Post request:
<request>
    <id>2</id>
    <name>sample</name>
    <description>Demo group</description>
    <email>group@mail.net</email>
</request>
```

Response

If the request executed successfully, then an HTTP 200 status code is returned along with an XML document that confirms the operation that has taken place. An example of a response to an update request is::

```
<response>
  <operation>updated</operation>
</response>
```

An example of a response to a create request is::

```
<response>
  <operation>added</operation>
</response>
```

If the request fails, then a HTTP 500 status code error is returned and the response contains an XML document with the details of the exception/what went wrong. An example of such a response is::

```
<error id="missing-parameter">
   <message>name</message>
   <class>MissingParameterEx</class>
   .....
</error>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or their profile is not permitted to execute the service. Returns 500 HTTP code
- **Missing parameter (error id: missing-parameter)**, when mandatory parameters are not provided. Returns 500 HTTP code
- **bad-parameter name**, when **name** it's empty. Returns 500 HTTP code
- **ERROR: duplicate key violates unique constraint "groups_name_key"**, when trying to create a new group using an existing group name. Returns 500 HTTP code

Update label translations (xml.group.update)

The **xml.group.update** service can be used to update translations of a group name. Only users with **Administrator** profile can update group name translations.

Requires authentication: Yes

Request

Parameters:

- **group**: Container for group information
- **id**: (mandatory) Group identifier to update
- **label**: (mandatory) This is just a container to hold the group names translated in the languages supported by GeoNetwork. Each translated label is enclosed in a tag that identifies the language code

Group label update request example:

```
Url:
http://localhost:8080/geonetwork/srv/en/xml.group.update

Mime-type:
application/xml

Post request:
<request>
    <group id="2">
        <label>
            <es>Grupo de ejemplo</es>
        </label>
    </group>
</request>
```

Response

Group label update response example:

```
<?xml version="1.0" encoding="UTF-8"?>
<ok />
```

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or their profile has no rights to execute the service. Returns 500 HTTP code
- **Missing parameter (error id: missing-parameter)**, when mandatory parameters are not provided. Returns 500 HTTP code

Get a group (xml.group.get)

The **xml.group.get** service can be used to retrieve information on an existing group.

Requires authentification: Yes

Request

Parameters:

- **id**: (mandatory) Group identifier to retrieve

Group get request example:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.group.get

Mime-type:
application/xml

Post request:
<request>
    <id>2</id>
</request>
```

Response

If the request executed succesfully then an HTTP 200 status code is returned and an XML document containing the group information is returned. An example response is::

```
<response>
  <record>
    <id>1</id>
    <name>all</name>
    <description/>
    <email/>
    <referrer/>
    <label>
      <ara>All</ara>
      <cat>All</cat>
      <chi>All</chi>
      <dut>Iedereen</dut>
      <eng>All</eng>
      .....
    </label>
```

```
  </record>
</response>
```

If the request fails then an HTTP 500 status code error is returned and the response contains an XML document with the details of the exception/what went wrong. An example error response is::

```
<error id="missing-parameter">
  <message>id</message>
  <class>MissingParameterEx</class>
  .....
</error>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or their profile has no rights to execute the service. Returns 500 HTTP code
- **Missing parameter (error id: missing-parameter)**, when mandatory parameters are not provided. Returns 500 HTTP code
- **bad-parameter id**, when **id** parameter is empty/invalid. Returns 500 HTTP code

Remove a group (xml.group.remove)

The **xml.group.remove** service can be used to remove an existing group. Only users with **Administrator** profile can delete groups.

Requires authentification: Yes

Request

Parameters:

- **id**: (mandatory) Group identifier to delete

Group remove request example:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.group.remove

Mime-type:
application/xml

Post request:
<request>
    <id>2</id>
</request>
```

Response

If the request executed succesfully then an HTTP 200 status code is returned and an XML document confirming the remove operation is returned. An example response is::

```
<response>
  <operation>removed</operation>
</response>
```

If the request fails then an HTTP 500 status code error is returned and the response contains an XML document with the details of the exception/what went wrong. An example error response is::

```
<error id="missing-parameter">
  <message>id</message>
  <class>MissingParameterEx</class>
  .....
</error>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or their profile has no rights to execute the service. Returns 500 HTTP code
- **Missing parameter (error id: missing-parameter)**, when mandatory parameters are not provided. Returns 500 HTTP code
- **bad-parameter id**, when **id** parameter is empty/invalid. Returns 500 HTTP code

6.4 User services

6.4.1 User Retrieval Services

List of Users (xml.info?type=users)

The **xml.info** service can be used to retrieve the users defined in GeoNetwork. See *Site Information (xml.info)*.

User groups list (xml.usergroups.list)

The **xml.usergroups.list** service can be used to retrieve the list of groups that a user belongs to.

Requires authentication: Yes

Request

Parameters:

- **id:** User identifier (multiple id elements can be specified)

User groups list request example:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.usergroups.list

Mime-type:
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <id>3</id>
<request>
```

Response

If the request executes successfully then HTTP status code 200 is returned with an XML document containing the groups that the user belongs to. The elements in the response are:

- **group:** This is the container for each user group element returned
- **id**: Group identifier
- **name**: Group name
- **description**: Group description

User groups list response example:

```
<groups>
  <group>
    <id>3</id>
    <name>RWS</name>
    <description />
  </group>
</groups>
```

If the request fails then an HTTP 500 status code error is returned and the response contains an XML document with the details of the exception/what went wrong. An example error response is::

```
<error id="missing-parameter">
  <message>id</message>
  <class>MissingParameterEx</class>
  .....
</error>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or their profile has no rights to execute the service. Returns 500 HTTP code
- **User XXXX doesn't exist**, if a user with provided **id** value does not exist. Returns 500 HTTP code

User information (xml.user.get)

The **xml.user.get** service returns information on a specified user.

Requires authentication: Yes

Request

Parameters:

- **id**: Identifier of user to retrieve

User get request example:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.user.get

Mime-type:
application/xml

Post request:
<request>
    <id>2</id>
</request>
```

Response

If the request executed succesfully then an HTTP 200 status code is returned and an XML document containing the user information (including the groups they belong to) is returned. An example response is::

```
<response>
  <record>
    <id>2</id>
    <username>bullshot</username>
    <password>112c535b861a904569285c941277d0c642eea4bb</password>
    <surname>Shot</surname>
    <name>Bull</name>
    <profile>RegisteredUser</profile>
    <address>41 Shot Street</address>
    <city>Kunnanurra</city>
    <state>Western Australia</state>
    <zip>8988</zip>
    <country>Australia</country>
    <email>gan@gan.com</email>
    <organisation>B7</organisation>
    <kind>gov</kind>
  </record>
  <groups>
    <id>2</id>
  </groups>
</response>
```

If the request fails then an HTTP 500 status code error is returned and the response contains an XML document with the details of the exception/what went wrong. An example error response is::

```
<error id="missing-parameter">
  <message>id</message>
  <class>MissingParameterEx</class>
  .....
</error>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or their profile has no rights to execute the service. Returns 500 HTTP code
- **Missing parameter (error id: missing-parameter)**, when mandatory parameters are not provided. Returns 500 HTTP code
- **bad-parameter id**, when **id** parameter is empty/invalid. Returns 500 HTTP code

6.4.2 User Maintenance Services

Create a user (xml.user.update)

The **xml.user.update** service can be used to create new users, update user information and reset user password, depending on the value of the **operation** parameter. Only users with profiles **Administrator** or **UserAdmin** can create new users.

Users with profile **Administrator** can create users in any group, while users with profile **UserAdmin** can create users only in the groups to which they belong.

Requires authentication: Yes

Request

Parameters:

- **operation**: (mandatory) **newuser**
- **username**: (mandatory) User login name
- **password**: (mandatory) User password
- **profile**: (mandatory) User profile
- **surname**:User surname
- **name**: User name
- **address**: User physical address
- **city**: User address city
- **state**: User address state
- **zip**: User address zip
- **country**: User address country

- **email**: User email
- **org**: User organisation/departament
- **kind**: Kind of organisation
- **groups**: Group identifier to set for the user, can be multiple **groups** elements
- **groupid**: Group identifier

User create request example:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.user.update

Mime-type:
application/xml

Post request:
<request>
  <operation>newuser</operation>
  <username>samantha</username>
  <password>editor2</password>
  <profile>Editor</profile>
  <name>Samantha</name>
  <city>Amsterdam</city>
  <country>Netherlands</country>
  <email>samantha@mail.net</email>
  <groups>2</groups>
  <groups>4</groups>
</request>
```

Response

If the request executed successfully then HTTP 200 status code is returned with an XML document containing an empty response element.

If the request fails, then an HTTP 500 status code error is returned with an XML document describing the exception/what went wrong. An example of such a response is::

```
<error id="error">
  <message>User with username samantha already exists</message>
  <class>IllegalArgumentException</class>
  <stack>...</stack>
  .....
</error>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or their profile has no rights to execute the service. Returns 500 HTTP code
- **Missing parameter (error id: missing-parameter)**, when mandatory parameters are not provided. Returns 500 HTTP code

- **bad-parameter**, when a mandatory fields is empty. Returns 500 HTTP code
- **User with username XXXX already exists (error id: error)**, when a user with that username is already present. Returns 500 HTTP code
- **Unknown profile XXXX (error id: error)**, when the profile is not valid. Returns 500 HTTP code
- **ERROR: duplicate key violates unique constraint "users_username_key"**, when trying to create a new user using an existing username. Returns 500 HTTP code
- **ERROR: insert or update on table "usergroups" violates foreign key constraint "usergroups_groupid_fkey"**, when group identifier is not an existing group identifier. Returns 500 HTTP code
- **ERROR: tried to add group id XX to user XXXX - not allowed because you are not a member of that group**, when the authenticated user has profile **UserAdmin** and tries to add the user to a group they do not manage. Returns 500 HTTP code
- **ERROR: you don't have rights to do this**, when the authenticated user has a profile that is not **Administrator** or **UserAdmin**. Returns 500 HTTP code

Update user information (xml.user.update)

The **xml.user.update** service can be used to create new users, update user information and reset user password, depending on the value of the **operation** parameter. Only users with profiles **Administrator** or **UserAdmin** can update users information.

Users with profile **Administrator** can update any user, while users with profile **UserAdmin** can update users only in the groups where they belong.

Requires authentication: Yes

Request

Parameters:

- **operation**: (mandatory) **editinfo**
- **id**: (mandatory) Identifier of the user to update
- **username**: (mandatory) User login name
- **password**: (mandatory) User password
- **profile**: (mandatory) User profile
- **surname**: User surname
- **name**: User name
- **address**: User physical address
- **city**: User address city
- **state**: User address state
- **zip**: User address zip
- **country**: User address country

- **email**: User email
- **org**: User organisation/departament
- **kind**: Kind of organisation
- **groups**: Group identifier to set for the user, can be multiple **groups** elements
- **groupid**: Group identifier

Note: If an optional parameter is not provided, the value is updated in the database with an empty string.

Update user information request example:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.user.update

Mime-type:
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <operation>editinfo</operation>
  <id>5</id>
  <username>samantha</username>
  <password>editor2</password>
  <profile>Editor</profile>
  <name>Samantha</name>
  <city>Rotterdam</city>
  <country>Netherlands</country>
  <email>samantha@mail.net</email>
</request>
```

Response

If the request executed successfully then HTTP 200 status code is returned with an XML document containing an empty response element.

If the request fails, then an HTTP 500 status code error is returned with an XML document describing the exception/what went wrong. An example of such a response is::

```
<error id="missing-parameter">
  <message>username</message>
  <class>MissingParameterEx</class>
  <stack>...</stack>
  .....
</error>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or their profile has no rights to execute the service. Returns 500 HTTP code
- **Missing parameter (error id: missing-parameter)**, when mandatory parameters are not provided. Returns 500 HTTP code
- **bad-parameter**, when a mandatory field is empty. Returns 500 HTTP code
- **Unknown profile XXXX (error id: error)**, when the profile is not valid. Returns 500 HTTP code
- **ERROR: duplicate key violates unique constraint "users_username_key"**, when trying to create a new user using an existing username. Returns 500 HTTP code
- **ERROR: insert or update on table "usergroups" violates foreign key constraint "usergroups_groupid_fkey"**, when the group identifier is not an existing group identifier. Returns 500 HTTP code
- **ERROR: tried to add group id XX to user XXXX - not allowed because you are not a member of that group**, when the authenticated user has profile **UserAdmin** and tries to add the user to a group in which they do not manage. Returns 500 HTTP code
- **ERROR: you don't have rights to do this**, when the authenticated user has a profile that is not **Administrator** or **UserAdmin**. Returns 500 HTTP code

Reset user password (xml.user.update)

The **xml.user.update** service can be used to create new users, update user information and reset user password, depending on the value of the **operation** parameter. Only users with profiles **Administrator** or **UserAdmin** can reset users password.

Users with profile **Administrator** can reset the password for any user, while users with profile **UserAdmin** can reset the password for users only in the groups where they belong.

Requires authentication: Yes

Request

Parameters:

- **operation**: (mandatory) **resetpw**
- **id**: (mandatory) Identifier of the user to reset the password
- **username**: (mandatory) User login name
- **password**: (mandatory) User new password
- **profile**: (mandatory) User profile

Reset user password request example:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.user.update

Mime-type:
```

```
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <operation>resetpw</operation>
  <id>2</id>
  <username>editor</username>
  <password>newpassword</password>
  <profile>Editor</profile>
</request>
```

Response

If the request executed successfully then HTTP 200 status code is returned with an XML document containing an empty response element.

If the request fails, then an HTTP 500 status code error is returned with an XML document describing the exception/what went wrong. An example of such a response is::

```
<error id="missing-parameter">
  <message>username</message>
  <class>MissingParameterEx</class>
  <stack>...</stack>
  .....
</error>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or their profile has no rights to execute the service. Returns 500 HTTP code
- **Missing parameter (error id: missing-parameter)**, when mandatory parameters are not provided. Returns 500 HTTP code
- **bad-parameter**, when a mandatory field is empty. Returns 500 HTTP code
- **Unknown profile XXXX (error id: error)**, when the profile is not valid. Returns 500 HTTP code
- **ERROR: you don't have rights to do this**, when the authenticated user is not an **Administrator** or **UserAdmin**. Returns 500 HTTP code

Update current authenticated user information (xml.user.infoupdate)

The **xml.user.infoupdate** service can be used to update the information related to the current authenticated user.

Requires authentication: Yes

Request

Parameters:

- **surname**: (mandatory) User surname
- **name**: (mandatory) User name
- **address**: User physical address
- **city**: User address city
- **state**: User address state
- **zip**: User address zip
- **country**: User address country
- **email**: User email
- **org**: User organisation/departament
- **kind**: Kind of organisation

Note: If an optional parameter is not provided the value is updated in the database with an empty string.

Current user info update request example:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.user.infoupdate

Mime-type:
application/xml

Post request:
<request>
  <name>admin</name>
  <surname>admin</surname>
  <address>address</address>
  <city>Amsterdam</city>
  <zip>55555</zip>
  <country>Netherlands</country>
  <email>user@mail.net</email>
  <org>GeoCat</org>
  <kind>gov</kind>
</request>
```

Response

If the request executed successfully then HTTP 200 status code is returned with an XML document containing an empty response element.

If the request fails, then an HTTP 500 status code error is returned with an XML document describing the exception/what went wrong. An example of such a response is::

```
<error id="missing-parameter">
  <message>surname</message>
  <class>MissingParameterEx</class>
```

```
  <stack>...</stack>
  .....
</error>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated. Returns 500 HTTP code
- **Missing parameter (error id: missing-parameter)**, when mandatory parameters are not provided. Returns 500 HTTP code

Change current authenticated user password (xml.user.pwupdate)

The **xml.user.pwupdate** service can be used to change the password of the current authenticated user.

Requires authentication: Yes

Request

Parameters:

- **password**: Actual user password
- **newPassword**: New password to set for the user

Example:

```
<request>
    <password>admin</password>
    <newPassword>admin2</newPassword>
</request>
```

Response

If the request executed successfully then HTTP 200 status code is returned with an XML document containing an empty response element.

If the request fails, then an HTTP 500 status code error is returned with an XML document describing the exception/what went wrong. An example of such a response is::

```
<error id="error">
  <message>Old password is not correct</message>
  <class>IllegalArgumentException</class>
  <stack>...</stack>
  .....
</error>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated. Returns 500 HTTP code
- **Old password is not correct**. Returns 500 HTTP code
- **Bad parameter (newPassword)**, when an empty password is provided. Returns 500 HTTP code

Remove a user (xml.user.remove)

The **xml.user.remove** service can be used to remove an existing user. Only users with profiles **Administrator** or **UserAdmin** can delete users.

Users with profile **Administrator** can delete any user (except themselves), while users with profile **UserAdmin** can delete users only in the groups where they belong (except themselves).

Requires authentification: Yes

Request

Parameters:

- **id**: (mandatory) Identifier of user to delete

User remove request example:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.user.remove

Mime-type:
application/xml

Post request:
<request>
    <id>2</id>
</request>
```

Response

If the request executed successfully then HTTP 200 status code is returned with an XML document containing an empty response element.

If the request fails, then an HTTP 500 status code error is returned with an XML document describing the exception/what went wrong. An example of such a response is::

```
<error id="error">
  <message>You cannot delete yourself from the user database</message>
  <class>IllegalArgumentException</class>
  <stack>...</stack>
  .....
</error>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or their profile has no rights to execute the service. Returns 500 HTTP code
- **Missing parameter (error id: missing-parameter)**, when the **id** parameter is not provided. Returns 500 HTTP code
- **You cannot delete yourself from the user database (error id: error)**. Returns 500 HTTP code
- **You don't have rights to delete this user (error id: error)**, when authenticated user is not an **Administrator** or **User administrator**. Returns 500 HTTP code
- **You don't have rights to delete this user because the user is not part of your group (error id: error)**, when trying to delete a user that is not in the same group as the authenticated user and the authenticated user is a **User administrator**. Returns 500 HTTP code

6.5 Category services

6.5.1 Category List (xml.info&type=categories)

The **xml.info** service can be used to retrieve the categories available in GeoNetwork. See *Site Information (xml.info)*.

6.5.2 Category maintenance

Create/update a category (xml.category.create.update)

The **xml.category.create.update** service can be used to create a new category and update the information about an existing category. Only users with **Administrator** profile can create/update categories.

Requires authentication: Yes

Request

Parameters:

- **id**: Category identifier to update. If not provided a new category is created with name provided.
- **name**: (mandatory) Name of the category

Category update request example:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.category.create.update

Mime-type:
application/xml

Post request:
<request>
    <id>2</id>
```

```
    <name>folios</name>
</request>
```

Response

If the request executed successfully, then an HTTP 200 status code is returned along with an XML document that confirms the operation that has taken place. An example of a response to an update request is::

```
<response>
  <operation>updated</operation>
</response>
```

An example of a response to a create request is::

```
<response>
  <operation>added</operation>
</response>
```

If the request fails, then a HTTP 500 status code error is returned and the response contains an XML document with the details of the exception/what went wrong. An example of such a response is::

```
<error id="missing-parameter">
   <message>name</message>
   <class>MissingParameterEx</class>
   .....
</error>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or their profile is not permitted to execute the service. Returns 500 HTTP code
- **Missing parameter (error id: missing-parameter)**, when mandatory parameters are not provided. Returns 500 HTTP code
- **bad-parameter name**, when **name** it's empty. Returns 500 HTTP code
- **ERROR: duplicate key violates unique constraint "categories_name_key"**, when trying to create a new category using an existing category name. Returns 500 HTTP code

Update label translations (xml.category.update)

The **xml.category.update** service can be used to update translations of a category name. Only users with **Administrator** profile can update category name translations.

Requires authentication: Yes

Request

Parameters:

- **category**: Container for category information
- **id**: (mandatory) Category identifier to update
- **label**: (mandatory) This is just a container to hold the category names translated in the languages supported by GeoNetwork. Each translated label is enclosed in a tag that identifies the language code

Category label update request example:

```
Url:
http://localhost:8080/geonetwork/srv/en/xml.category.update

Mime-type:
application/xml

Post request:
<request>
    <category id="2">
        <label>
            <eng>folios</eng>
        </label>
    </category>
</request>
```

Response

Category label update response example:

```
<ok />
```

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or their profile has no rights to execute the service. Returns 500 HTTP code
- **Missing parameter (error id: missing-parameter)**, when mandatory parameters are not provided. Returns 500 HTTP code

Get a category (xml.category.get)

The **xml.category.get** service can be used to retrieve information on an existing category.

Requires authentification: Yes

Request

Parameters:

- **id**: (mandatory) Category identifier to retrieve

Category get request example:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.category.get

Mime-type:
application/xml

Post request:
<request>
    <id>2</id>
</request>
```

Response

If the request executed succesfully then an HTTP 200 status code is returned and an XML document containing the category information is returned. An example response is::

```
<response>
  <record>
    <id>2</id>
    <name>datasets</name>
    <label>
      <ara>Datasets</ara>
      <cat>Conjunts de dades</cat>
      <eng>Datasets</eng>
      .....
    </label>
  </record>
</response>
```

If the request fails then an HTTP 500 status code error is returned and the response contains an XML document with the details of the exception/what went wrong. An example error response is::

```
<error id="missing-parameter">
  <message>id</message>
  <class>MissingParameterEx</class>
  .....
</error>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or their profile has no rights to execute the service. Returns 500 HTTP code
- **Missing parameter (error id: missing-parameter)**, when mandatory parameters are not provided. Returns 500 HTTP code
- **bad-parameter id**, when **id** parameter is empty/invalid. Returns 500 HTTP code

Remove a category (xml.category.remove)

The **xml.category.remove** service can be used to remove an existing category. Only users with **Administrator** profile can delete categories.

Requires authentification: Yes

Request

Parameters:

- **id**: (mandatory) Category identifier to delete

Category remove request example:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.category.remove

Mime-type:
application/xml

Post request:
<request>
    <id>2</id>
</request>
```

Response

If the request executed succesfully then an HTTP 200 status code is returned and an XML document confirming the remove operation is returned. An example response is::

```
<response>
  <operation>removed</operation>
</response>
```

If the request fails then an HTTP 500 status code error is returned and the response contains an XML document with the details of the exception/what went wrong. An example error response is::

```
<error id="missing-parameter">
  <message>id</message>
  <class>MissingParameterEx</class>
  .....
</error>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or their profile has no rights to execute the service. Returns 500 HTTP code
- **Missing parameter (error id: missing-parameter)**, when mandatory parameters are not provided. Returns 500 HTTP code
- **bad-parameter id**, when **id** parameter is empty/invalid. Returns 500 HTTP code

6.6 Search and Retrieve Metadata services

6.6.1 Search metadata (xml.search)

The **xml.search** service can be used to retrieve metadata records from GeoNetwork.

Requires authentication: Optional

Request

Search configuration parameters (all values are optional)

- **remote**: Search in local catalog or in a remote catalog. Values: off (default), on
- **extended**: Values: on, off (default)
- **timeout**: Timeout for request in seconds (default: 20)
- **hitsPerPage**: Results per page (default: 10)
- **similarity**: Lucene accuracy for searches (default 0.8)
- **sortBy**: Sorting criteria. Values: relevance (default), rating, popularity, changeDate, title

Search parameters (all values are optional):

- **eastBL, southBL, northBL, westBL**: Bounding box to restrict the search
- **relation**: Bounding box criteria. Values: equal, overlaps (default), encloses, fullyOutsideOf, intersection, crosses, touches, within
- **any**: Text to search in a free text search
- **title**: Metadata title
- **abstract**: Metadata abstract
- **themeKey**: Metadata keywords. To search for several use a value like “Global” or “watersheds”
- **template**: Indicates if search for templates or not. Values: n (default), y
- **dynamic**: Map type. Values: off (default), on
- **download**: Map type. Values: off (default), on
- **digital**: Map type. Values: off (default), on
- **paper**: Map type. Values: off (default), on
- **group**: Filter metadata by group, if missing search in all groups
- **attrset**:
- **dateFrom**: Filter metadata created after specified date
- **dateTo**: Filter metadata created before specified date
- **category**: Metadata category. If not specified, search all categories

Request to search for all metadata example:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.search

Mime-type:
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request />
```

Request with free text search example:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.search

Mime-type:
application/xml

Post request:s
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <any>africa</any>
</request>
```

Request with a geographic search example:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.search

Mime-type:
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <any>africa</any>
  <eastBL>74.91574</eastBL>
  <southBL>29.40611</southBL>
  <northBL>38.47198</northBL>
  <westBL>60.50417</westBL>
  <relation>overlaps</relation>
  <sortBy>relevance</sortBy>
  <attrset>geo</attrset>
</request>
```

Request to search using dates and keywords example:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.search

Mime-type:
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <title>africa</title>
  <themekey>"Global" or "World"</themekey>
```

```
  <dateFrom>2000-02-03T12:47:00</dateFrom>
  <dateTo>2010-02-03T12:49:00</dateTo>
</request>
```

Response

The response is the metadata record with additional **geonet:info** section. The main fields are:

- **response**: Response container.
  - **summary**: Attribute **count** indicates the number of metadata records retrieved
    - **keywords**: List of keywords that are part of the metadata resultset. Each keyword contains the value and the number of occurences in the retrieved metadata records.
  - **metadata**: Container for each metadata record found. Each container has a **geonet:info** element with the following information:
    - **id**: Metadata internal identifier
    - **uuid** : Metadata Universally Unique Identifier (UUID)
    - **schema**: Metadata schema
    - **createDate**: Metadata creation date
    - **changeDate**: Metadata last modification date
    - **source**: Source catalogue the metadata
    - **category**: Metadata category (Can be multiple elements)
    - **score**: Value indicating the accuracy of search

Metadata search response example:

```
<?xml version="1.0" encoding="UTF-8"?>
<response from="1" to="7">
  <summary count="7" type="local">
    <keywords>
      <keyword count="2" name="Global"/>
      <keyword count="2" name="World"/>
      <keyword count="2" name="watersheds"/>
      <keyword count="1" name="Biology"/>
      <keyword count="1" name="water resources"/>
      <keyword count="1" name="endangered plant species"/>
      <keyword count="1" name="Africa"/>
      <keyword count="1" name="Eurasia"/>
      <keyword count="1" name="endangered animal species"/>
      <keyword count="1" name="Antarctic ecosystem"/>
    </keywords>
  </summary>
  <metadata xmlns:gmx="http://www.isotc211.org/2005/gmx">
    <geonet:info xmlns:geonet="http://www.fao.org/geonetwork">
      <id>12</id>
      <uuid>bc179f91-11c1-4878-b9b4-2270abde98eb</uuid>
      <schema>iso19139</schema>
      <createDate>2007-07-25T12:05:45</createDate>
      <changeDate>2007-11-06T12:10:47</changeDate>
      <source>881a1630-d4e7-4c9c-aa01-7a9bbbbc47b2</source>
```

```
      <category>maps</category>
      <category>interactiveResources</category>
      <score>1.0</score>
    </geonet:info>
  </metadata>
  <metadata xmlns:gmx="http://www.isotc211.org/2005/gmx">
    <geonet:info xmlns:geonet="http://www.fao.org/geonetwork">
      <id>11</id>
      <uuid>5df54bf0-3a7d-44bf-9abf-84d772da8df1</uuid>
      <schema>iso19139</schema>
      <createDate>2007-07-19T14:45:07</createDate>
      <changeDate>2007-11-06T12:13:00</changeDate>
      <source>881a1630-d4e7-4c9c-aa01-7a9bbbbc47b2</source>
      <category>maps</category>
      <category>datasets</category>
      <category>interactiveResources</category>
      <score>0.9178859</score>
    </geonet:info>
  </metadata>
</response>
```

6.6.2 Get metadata (xml.metadata.get)

The **xml.metadata.get** service can be used to retrieve a metadata record stored in GeoNetwork.

Requires authentication: Optional

Request

One of the following parameters:

- **uuid** : Metadata Universal Unique Identifier (UUID)
- **id**: Metadata internal identifier

Get metadata request example:

```
Url:
http://localhost:8080/geonetwork/srv/en/xml.metadata.get

Mime-type:
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <uuid>aa9bc613-8eef-4859-a9eb-4df35d8b21e4</uuid>
</request>
```

Response

Successful response (HTTP status code 200) is the XML metadata record with additional **geonet:info** section. The principal fields for **geonet:info** are:

- **schema**: Metadata schema

- **createDate**: Metadata creation date
- **changeDate**: Metadata last modification date
- **isTemplate**: Indicates if the metadata returned is a template
- **title**: Metadata title
- **source**: Source catalogue the metadata
- **uuid** : Metadata Universally Unique Identifier (UUID)
- **isHarvested**: Indicates if the metadata is harvested
- **popularity**: Indicates how often the record is retrieved
- **rating**: Average rating provided by users
- State of operation on metadata for the user: view, notify, download, dynamic, featured, edit
- **owner**: Indicates if the user that executed the service is the owner of metadata
- **ownername**: Metadata owner name

Get metadata response example:

```
<?xml version="1.0" encoding="UTF-8"?>
<Metadata xmlns:geonet="http://www.fao.org/geonetwork"
  xmlns:csw="http://www.opengis.net/cat/csw/2.0.2">
  <mdFileID>aa9bc613-8eef-4859-a9eb-4df35d8b21e4</mdFileID>
  ...
  <geonet:info>
    <id>10</id>
    <schema>iso19115</schema>
    <createDate>2005-08-23T17:58:18</createDate>
    <changeDate>2007-03-12T17:49:50</changeDate>
    <isTemplate>n</isTemplate>
    <title />
    <source>881a1630-d4e7-4c9c-aa01-7a9bbbbc47b2</source>
    <uuid>aa9bc613-8eef-4859-a9eb-4df35d8b21e4</uuid>
    <isHarvested>n</isHarvested>
    <popularity>0</popularity>
    <rating>0</rating>
    <view>true</view>
    <notify>true</notify>
    <download>true</download>
    <dynamic>true</dynamic>
    <featured>true</featured>
    <edit>true</edit>
    <owner>true</owner>
    <ownername>admin</ownername>
    <subtemplates />
  </geonet:info>
</Metadata>
```

Error response (HTTP 500 status code) is an XML document with the details of what went wrong. An example of such a response is as follows:

```
<error id="operation-not-allowed">
  <message>Operation not allowed</message>
  <class>OperationNotAllowedEx</class>
```

```
  .....
</error>
```

See *Exception handling* for more details.

Errors

- **Request must contain a UUID or an ID**, if a uuid or id parameter was not provided. Returns 500 HTTP code
- **Operation not allowed (error id: operation-not-allowed)**, when the user is not allowed to view the metadata record. Returns 500 HTTP code

6.6.3 Get user metadata (xml.user.metadata)

The **xml.user.metadata** service can be used to retrieve a metadata records according to the user profile of the authenticated user running the service:

- *Administrator* profile: return all metadata records
- *Reviewer* or *User Administrator* profile: return all metadata records with groupOwner in the set of groups the user belongs to
- *Editor* profile: return all metadata records owned by the user

Requires authentication: Yes

Request

- **sortBySelect** : (optional) parameter specifying sort order of metadata records returned.

Get metadata request example:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.user.metadata

Mime-type:
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request/>
```

Response

Successful response is an XML document with a response container and the user metadata records as children of that container. Each child has a **geonet:info** element which gives GeoNetwork specific metadata about the metadata record. An example response (with some content removed for brevity) is as follows:

```
<response>
  <!-- metadata record 1 -->
  <gmd:MD_Metadata ....>
  </gmd:MD_Metadata>
```

```
  <!-- metadata record 2 -->
  <gmd:MD_Metadata ....>
  </gmd:MD_Metadata>
</response>
```

Error response is an XML document with error container and the details of the error. Example:

```
<error id="service-not-allowed">
        <message>Service not allowed</message>
  ....
  <object>xml.user.metadata</object>
  <request>
    <language>eng</language>
    <service>xml.user.metadata</service>
  </request>
</error>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, user isn't allowed to run this service. Returned 500 HTTP code.
- **Unauthorized user attempted to list editable metadata (error id: operation-not-allowed)**, when the user is not allowed to list metadata records. Returned 500 HTTP code

6.6.4 RSS Search: Search metadata and retrieve in RSS format (rss.search)

The **rss.search** service can be used to retrieve metadata records in RSS format, using regular search parameters. This service can be configured in **WEB-INF/config.xml** with the following parameters:

- **maxSummaryKeys**: Maximum number of RSS records to retrieve (default = 10)

Requires authentication: Optional. If not provided only public metadata records are retrieved

Request

Parameters:

- **georss**: valid values are simple, simplepoint and default. See also http://georss.org
  - **simple**: Bounding box in georss simple format
  - **simplepoint**: Bounding box in georss simplepoint format
  - **default**: Bounding box in georss GML format
- **eastBL, southBL, northBL, westBL**: Bounding box to restrict the search****
- **relation**: Bounding box criteria. Values: equal, overlaps (default), encloses, fullyOutsideOf, intersection, crosses, touches, within
- **any**: Text to search in a free text search
- **title**: Metadata title

- **abstract**: Metadata abstract
- themeKey: Metadata keywords. To search for several use a value like “Global” or “watersheds”
- **dynamic**: Map type. Values: off (default), on
- **download**: Map type. Values: off (default), on
- **digital**: Map type. Values: off (default), on
- **paper**: Map type. Values: off (default), on
- **group**: Filter metadata by group, if missing search in all groups
- **attrset**:
- **dateFrom**: Filter metadata created after specified date
- **dateTo**: Filter metadata created before specified date
- **category**: Metadata category. If not specified, search all categories

RSS search request example:

```
Url:
http://localhost:8080/geonetwork/srv/en/rss.search

Mime-type:
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <georss>simplepoint</georss>
  <any>africa</any>
  <eastBL>74.91574</eastBL>
  <southBL>29.40611</southBL>
  <northBL>38.47198</northBL>
  <westBL>60.50417</westBL>
  <relation>overlaps</relation>
  <sortBy>relevance</sortBy>
  <attrset>geo</attrset>
</request>
```

Response

The principal fields of the response are:

- **channel**: This is the container for the RSS response
  - **title**: RSS channel title
  - **description**: RSS channel description
  - **item**: Metadata RSS item (one item for each metadata retrieved)
    - **title**: Metadata title
    - **link**: Link to show metadata page. Additional link elements (with rel=”alternate”) to OGC WXS services, shapefile/images files, Google KML, etc. can be returned depending on metadata

    * **description**: Metadata description
    * **pubDate**: Metadata publication date
    * **media**: Metadata thumbnails
    * **georss:point**: Bounding box in georss simplepoint format

RSS search response example:

```
Mimetype:
application/rss+xml

Response:
<?xml version="1.0" encoding="UTF-8"?>
<rss xmlns:media="http://search.yahoo.com/mrss/" xmlns:georss="http://www.georss.org/geo
  <channel>
    <title>GeoNetwork opensource portal to spatial data and information</title>
    <link>http://localhost:8080/geonetwork</link>
    <description>GeoNetwork opensource provides Internet access to interactive maps, sat
    <language>en</language>
    <copyright>All rights reserved. Your generic copyright statement </copyright>
    <category>Geographic metadata catalog</category>
    <generator>GeoNetwork opensource</generator>
    <ttl>30</ttl>
    <item>
      <title>Hydrological Basins in Africa (Sample record, please remove!)</title>
      <link>http://localhost:8080/geonetwork?uuid=5df54bf0-3a7d-44bf-9abf-84d772da8df1</
      <link href="http://geonetwork3.fao.org/ows/296?SERVICE=wms$amp;VERSION=1.1.1&REQUE
      <link href="http://localhost:8080/geonetwork/srv/en/google.kml?uuid=5df54bf0-3a7d-
      <category>Geographic metadata catalog</category>
      <description><![CDATA[ ... ]]></description>
      <pubDate>06 Nov 2007 12:13:00 EST</pubDate>
      <guid>http://localhost:8080/geonetwork?uuid=5df54bf0-3a7d-44bf-9abf-84d772da8df1</
      <media:content url="/geonetwork/srv/en/resources.get?id=11&fname=thumbnail_s.gif&a
      <media:text>Major hydrological basins and their sub-basins ...</media:text>
      <!--Bounding box in georss simplepoint format (default) (http://georss.org)-->
      <georss:point>16.9 1.8</georss:point>
      </item>
  </channel>
</rss>
```

6.6.5 RSS latest: Get latest updated metadata (rss.latest)

The **rss.latest** service can be used to retrieve the most recently modified metadata records in RSS format. This service can be configured in **WEB-INF/config.xml** file with the following parameters:

- **maxItems**: Maximum number of RSS records to retrieve (default = 20)
- **timeBetweenUpdates**: Minimum time (in seconds) between queries for latest updated metadata. If a request is received less than timeBetweenUpdates seconds after the last request, it will receive the same response.

Requires authentication: Optional. If not provided only public metadata records are retrieved

Request

Parameters:

- **georss**: valid values are simple, simplepoint and default. See also http://georss.org
  - **simple**: Bounding box in georss simple format
  - **simplepoint**: Bounding box in georss simplepoint format
  - **default**: Bounding box in georss GML format

RSS latest request example:

```
Url:
http://localhost:8080/geonetwork/srv/en/rss.latest

Mime-type:
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <georss>default</georss>
  <maxItems>1</maxItems>
</request>
```

Response

The following are the principal fields of the response:

- **channel**: This is the container for the RSS response
  - **title**: RSS channel title
  - **description**: RSS channel description
  - **item**: Metadata RSS item (one item for each metadata retrieved)
    - **title**: Metadata title
    - **link**: Link to show metadata page. Additional link elements (with rel="alternate") to OGC WXS services, shapefile/images files, Google KML, etc. can be returned depending on metadata
    - **description**: Metadata description
    - **pubDate**: Metadata publication date
    - **media**: Metadata thumbnails
    - **georss:where**: Bounding box with the metadata extent

RSS latest response example:

```
Mimetype:
application/rss+xml

Response:
<?xml version="1.0" encoding="UTF-8"?>
<rss xmlns:media="http://search.yahoo.com/mrss/" xmlns:georss="http://www.georss.org/geo
```

```
  xmlns:gml="http://www.opengis.net/gml" version="2.0">
<channel>
  <title>GeoNetwork opensource portal to spatial data and information</title>
  <link>http://localhost:8080/geonetwork</link>
  <description>GeoNetwork opensource provides Internet access to interactive maps,
  satellite imagery and related spatial databases ... </description>
  <language>en</language>
  <copyright>All rights reserved. Your generic copyright statement </copyright>
  <category>Geographic metadata catalog</category>
  <generator>GeoNetwork opensource</generator>
  <ttl>30</ttl>
  <item>
    <title>Hydrological Basins in Africa (Sample record, please remove!)</title>
    <link>http://localhost:8080/geonetwork?uuid=5df54bf0-3a7d-44bf-9abf-84d772da8df1</li
    <link href="http://geonetwork3.fao.org/ows/296?SERVICE=wms$amp;VERSION=1.1.1&REQUEST
      &BBOX=-17.3,-34.6,51.1,38.2&LAYERS=hydrological_basins&SRS=EPSG:4326&WIDTH=200
      &HEIGHT=213&FORMAT=image/png&TRANSPARENT=TRUE&STYLES=default" type="image/png"
      rel="alternate" title="Hydrological basins in Africa"/>
    <link href="http://localhost:8080/geonetwork/srv/en/google.kml?
      uuid=5df54bf0-3a7d-44bf-9abf-84d772da8df1&layers=hydrological_basins"
      type="application/vnd.google-earth.kml+xml"
      rel="alternate" title="Hydrological basins in Africa"/>
    <category>Geographic metadata catalog</category>
    <description><![CDATA[ ... ]]></description>
    <pubDate>06 Nov 2007 12:13:00 EST</pubDate>
    <guid>http://localhost:8080/geonetwork?uuid=5df54bf0-3a7d-44bf-9abf-84d772da8df1</gu
    <media:content url="/geonetwork/srv/en/resources.get?id=11&fname=thumbnail_s.gif
      &access=public" type="image/gif" width="100"/>
      <media:text>Major hydrological basins and their sub-basins ...</media:text>
   <!--Bounding box in georss GML format (http://georss.org)-->
   <georss:where>
     <gml:Envelope>
       <gml:lowerCorner>-34.6 -17.3</gml:lowerCorner>
       <gml:upperCorner>38.2 51.1</gml:upperCorner>
     </gml:Envelope>
   </georss:where>
  </item>
</channel>
</rss>
```

6.7 Metadata insert, update and delete services

These services provide insert, update and delete operations for metadata records. They could be used by a metadata editing program external to GeoNetwork.

This is the Create, Update, Delete part of the metadata CRUD operations in GeoNetwork. For read/retrieve operations (the R in CRUD) see *Search and Retrieve Metadata services*.

6.7.1 Insert metadata (xml.metadata.insert)

The **xml.metadata.insert** service allows you to insert a new record into the catalogue.

Requires authentication: Yes

Request

Parameters:

- **data**: (mandatory) Contains the metadata record
- **group** (mandatory): Owner group identifier for metadata
- **isTemplate**: indicates if the metadata content is a new template or not. Default value: "n"
- **title**: Metadata title. Only required if isTemplate = "y"
- **category** (mandatory): Metadata category. Use "_none_" value to don't assign any category
- **styleSheet** (mandatory): Stylesheet name to transform the metadata before inserting in the catalog. Use "_none_" if you don't have a stylesheet to apply
- **validate**: Indicates if the metadata should be validated before inserting in the catalog. Values: on, off (default)

Insert metadata request example:

```
Url:
http://localhost:8080/geonetwork/srv/en/metadata.insert

Mime-type:
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <group>2</group>
  <category>_none_</category>
  <styleSheet>_none_</styleSheet>
  <data><![CDATA[
    <gmd:MD_Metadata xmlns:gmd="http://www.isotc211.org/2005/gmd"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    ...
       </gmd:DQ_DataQuality>
      </gmd:dataQualityInfo>
    </gmd:MD_Metadata>]]>
  </data>
</request>
```

Response

If request is executed successfully HTTP 200 status code is returned along with an XML document with id details of the record inserted. Example success response:

```
<response>
  <id>31</id>
  <uuid>9c623013-8d90-4e61-ae61-8e96800f3b08</uuid>
</response>
```

If request fails an HTTP status code 500 is returned and the response contains an XML document with the exception. Example error response:

```
<error id="error">
  <message>Unique index or primary key violation: "CONSTRAINT_INDEX_1 ON PUBLIC.METADATA
  <class>JdbcSQLException</class>
  <stack>..</stack>
  <request>...</request>
</error>
```

See *Exception handling* for more details.

If validate parameter is set to "on" and the provided metadata is not valid with respect to the XSD and schematrons in use for the metadata schema then an exception report is returned.

Example validation metadata report:

```
<?xml version="1.0" encoding="UTF-8"?>
<error id="xsd-validation-error">
  <message>XSD Validation error(s)</message>
  <class>XSDValidationErrorEx</class>
  <stack>
    <at class="org.fao.geonet.services.metadata.ImportFromDir"
      file="ImportFromDir.java" line="297" method="validateIt" />
    <at class="org.fao.geonet.services.metadata.ImportFromDir"
      file="ImportFromDir.java" line="281" method="validateIt" />
    <at class="org.fao.geonet.services.metadata.Insert"
      file="Insert.java" line="102" method="exec" />
    <at class="jeeves.server.dispatchers.ServiceInfo"
      file="ServiceInfo.java" line="238" method="execService" />
    <at class="jeeves.server.dispatchers.ServiceInfo"
      file="ServiceInfo.java" line="141" method="execServices" />
    <at class="jeeves.server.dispatchers.ServiceManager"
      file="ServiceManager.java" line="377" method="dispatch" />
    <at class="jeeves.server.JeevesEngine"
      file="JeevesEngine.java" line="621" method="dispatch" />
    <at class="jeeves.server.sources.http.JeevesServlet"
      file="JeevesServlet.java" line="174" method="execute" />
    <at class="jeeves.server.sources.http.JeevesServlet"
      file="JeevesServlet.java" line="99" method="doPost" />
    <at class="javax.servlet.http.HttpServlet"
      file="HttpServlet.java" line="727" method="service" />
  </stack>
  <object>
    <xsderrors>
      <error>
        <message>ERROR(1) org.xml.sax.SAXParseException: cvc-datatype-valid.1.2.1: '' is
        <xpath>gmd:identificationInfo/gmd:MD_DataIdentification/gmd:citation/gmd:CI_Cita
      </error>
      <error>
        <message>ERROR(2) org.xml.sax.SAXParseException: cvc-type.3.1.3: The value '' of
        <xpath>gmd:identificationInfo/gmd:MD_DataIdentification/gmd:citation/gmd:CI_Cita
      </error>
      <error>
        <message>ERROR(3) org.xml.sax.SAXParseException: cvc-datatype-valid.1.2.1: '' is
        <xpath>gmd:identificationInfo/gmd:MD_DataIdentification/gmd:spatialResolution/gm
      </error>
      <error>
        <message>ERROR(4) org.xml.sax.SAXParseException: cvc-type.3.1.3: The value '' of
        <xpath>gmd:identificationInfo/gmd:MD_DataIdentification/gmd:spatialResolution/gm
      </error>
```

```
    </xsderrors>
  </object>
  <request>
    <language>eng</language>
    <service>xml.metadata.insert</service>
  </request>
</error>
```

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or their profile has no rights to execute the service. Returns 500 HTTP code
- **Missing parameter (error id: missing-parameter)**, when mandatory parameters are not provided. Returns 500 HTTP code
- **bad-parameter XXXX**, when a mandatory parameter is empty. Returns 500 HTTP code
- **ERROR: duplicate key violates unique constraint "metadata_uuid_key"**, if another metadata record in catalog has the same uuid of the metadata record being inserted. Returns 500 HTTP code

6.7.2 Update metadata (xml.metadata.update)

The **xml.metadata.update** service allows you to update a metadata record in the catalog.

Requires authentication: Yes

Request

Parameters:

- **id** or **uuid**: (mandatory) Identifier of the metadata to update
- **version**: (mandatory) This parameter is used by the GeoNetwork editor to avoid concurrent updates to the same metadata record. This is not accessible to the service user at present so this parameter can be set to any integer value.
- **isTemplate**: indicates if the metadata content is a new template or not. Default value: "n"
- **showValidationErrors**: Indicates if the metadata should be validated before updating in the catalog.
- **minor**: If the metadata update is a minor change (changedate will not be updated, notification of change in metadata will not be sent) then this parameter should be set to "true", "false" otherwise.
- **title**: Metadata title (for templates)
- **data** (mandatory) Contains the metadata record.

Update metadata request example:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.metadata.update

Mime-type:
```

```
application/xml

Post request:

<?xml version="1.0" encoding="UTF-8"?>
<request>
  <id>11</id>
  <version>1</version>
  <data><![CDATA[
    <gmd:MD_Metadata xmlns:gmd="http://www.isotc211.org/2005/gmd"
                     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    ...

          </gmd:DQ_DataQuality>
      </gmd:dataQualityInfo>
    </gmd:MD_Metadata>]]>
  </data>
</request>
```

Response

If request is executed successfully HTTP 200 status code is returned and an XML document with details of the successful request. Example success response:

```
<response>
  <id>32</id>
  <showvalidationerrors>false</showvalidationerrors>
  <minor>false</minor>
</response>
```

If request fails an HTTP status code 500 (server error) is returned and the response is an XML document with the exception. Example error response:

```
<error id="bad-parameter">
  <message>id</message>
  <class>BadParameterEx</class>
  <stack>...</stack>
  <request>...</request>
</error>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or his profile has no rights to execute the service. Returns 500 HTTP code
- **Missing parameter (error id: missing-parameter)**, when mandatory parameters are not provided. Returns 500 HTTP code
- **bad-parameter XXXX**, when a mandatory parameter is empty or when the update id doesn't exist. Returns 500 HTTP code
- **Concurrent update (error id: client)**, when the version number provided is different from the current version number (Metadata record is in use by another user). Returns 500 HTTP code

6.7.3 Delete metadata (xml.metadata.delete)

The **xml.metadata.delete** service removes a metadata record from the catalog. The metadata record is backed up in MEF format in `GEONETWORK_DATA_DIR/removed`.

Requires authentication: Yes

Request

Parameters:

- **id** or **uuid**: (mandatory) Identifier of the metadata to delete

Example request:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.metadata.delete

Mime-type:
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <id>10</id>
</request>
```

Response

If request executed successfully HTTP 200 status code is returned and an XML document with details of what has been deleted. Example success response:

```
<response>
  <id>32</id>
</response>
```

If request fails an HTTP 500 status code error is returned and the response is an XML document with the exception. Example error response:

```
<error id="metadata-not-found">
  <message>Metadata not found</message>
  <class>MetadataNotFoundEx</class>
  <stack>...</stack>
  <request>...</request>
</error>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or their profile has no rights to execute the service. Returns 500 HTTP code
- **Metadata not found (error id: metadata-not-found)**, if the identifier provided did not correspond to an existing metadata record. Returns 500 HTTP code

- **Operation not allowed (error id: operation-not-allowed)**, when the user is not authorized to edit the metadata. To edit a metadata one of the following must be true:
  - The user is the metadata owner
  - The user is an Administrator
  - The user has edit rights over the metadata
  - The user is a Reviewer and/or UserAdmin and the metadata groupOwner is one of his groups

  Returns 500 HTTP code.

6.7.4 Batch Delete (xml.metadata.batch.delete)

The **xml.metadata.batch.delete** service deletes the metadata records in the selected set.

Note: This service requires a previous call to the `xml.metadata.select` service (see *Select metadata records (xml.metadata.select)*) to select the metadata records to delete.

Note: Only those metadata records for which the user running the service has ownership rights on will be deleted. If metadata versioning is on then deletions will be recorded in the version history.

Requires authentication: Yes

6.7.5 Request

Parameters: **None**

Example request:

POST:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.metadata.batch.delete

Mime-type:
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request/>
```

GET:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.metadata.batch.delete
```

Response

If the request executed successfully then HTTP 200 status code is returned and an XML document with a summary of how the metadata records in the selected set have been processed. An example of such a response is shown below:

```
<response>
        <done>5</done>
  <notOwner>0</notOwner>
  <notFound>0</notFound>
</response>
```

The response fields are:

- **done** - number of metadata records successfully deleted
- **notOwner** - number of metadata records skipped because the user running this service did not have ownership rights
- **notFound** - number of metadata records skipped because they were not found (may have been deleted)

If the request fails an HTTP 500 status code error is returned and the response is an XML document with the exception. An example of such a response is shown below:

```
<error id="service-not-allowed">
  Service not allowed
  <object>xml.metadata.batch.delete</object>
</error>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or their profile has no rights to execute the service. Returns 500 HTTP code

6.8 Metadata Select services

These services are for creating and managing a set of selected metadata records. The selected set is normally used by the metadata.batch services eg. *Batch update privileges (xml.metadata.batch.update.privileges)*, *Batch new owner (xml.metadata.batch.newowner)*, *Batch update status (xml.metadata.batch.update.status)*, *Batch update categories (xml.metadata.batch.update.categories)*, *Batch start versioning (xml.metadata.batch.version)*, *Batch process metadata records with an XSLT (xml.metadata.batch.processing)* and *Batch Delete (xml.metadata.batch.delete)*.

6.8.1 Select metadata records (xml.metadata.select)

This service can be used to build and manage a selected set of metadata.

Request

Parameters:

- **id**: Identifier of metadata to select (can be more than one)
- **selected**: Selection state. Values: add, add-all, remove, remove-all

Select all metadata example:

```
Url:
http://localhost:8080/geonetwork/srv/eng/metadata.select

Mime-type:
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <selected>add-all</selected>
</request>
```

Select a metadata record example:

```
Url:
http://localhost:8080/geonetwork/srv/eng/metadata.select

Mime-type:
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <id>2</id>
  <selected>add</selected>
</request>
```

Clear metadata selection example:

```
Url:
http://localhost:8080/geonetwork/srv/eng/metadata.select

Mime-type:
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <selected>remove-all</selected>
</request>
```

Response

The XML response from this service *always* contains the number of metadata records selected after applying the select operation.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <Selected>10</Selected>
</request>
```

6.9 Metadata Privilege services

6.9.1 Update privileges on a metadata record (xml.metadata.privileges)

The **xml.metadata.privileges** service updates the privileges on a metadata record using a list of groups and privileges sent as parameters.

Note: All previously assigned privileges will be deleted. If versioning for the metadata record is on, then the previously assigned privileges will be available in the version history.

Requires authentication: Yes

Request

Parameters:

- **id** or **uuid**: Identifier of metadata to update
- **_G_O**: (can be multiple elements)
  - **G**: Group identifier
  - **O**: Privilege (Operation) identifier. Privilege identifiers:
  - 0: view
  - 1: download
  - 2: editing
  - 3: notify
  - 4: dynamic
  - 5: featured
  - Group and Operation Identifiers can be obtained using *Site Information (xml.info)* service.

Request example:

POST:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.metadata.privileges

Mime-type:
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <id>6</id>
  <_1_2 />
  <_1_1 />
</request>
```

GET:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.metadata.privileges?id=6&_1_2&_1_1
```

Response

If the request executed successfully then the XML response contains the identifier of the metadata whose privileges have been updated.

Example:

```
<response>
  <id>6</id>
</response>
```

If the request was unsuccessful then the XML response contains details of the error returned. An example of such a response is:

```
<error id="metadata-not-found">
  <message>Metadata not found</message>
  <class>MetadataNotFoundEx</class>
  .....
  <object>6</object>
  <request>
    <language>eng</language>
    <service>xml.metadata.privileges</service>
  </request>
</error>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or their profile has no rights to execute the service. Returns 500 HTTP code
- **Metadata not found (error id: metadata-not-found)** if a metadata record with the identifier provided does not exist. Returns 500 HTTP code
- **ERROR: insert or update on table "operationallowed" violates foreign key 'operationallowed_operationid_fkey »**, if an operation identifier provided is not valid. Returns 500 HTTP code
- **ERROR: insert or update on table "operationallowed" violates foreign key 'operationallowed_groupid_fkey »**, if a group identifier provided is not valid. Returns 500 HTTP code

6.9.2 Batch update privileges (xml.metadata.batch.update.privileges)

The **xml.metadata.batch.update.privileges** service updates the privileges on a selected set of metadata using the list of groups and privileges sent as parameters.

Note: This service requires a previous call to the `xml.metadata.select` service (see *Select metadata records (xml.metadata.select)*) to select metadata records.

Note: Only those metadata records for which the user running the service has ownership rights on will be updated and all privileges previously assigned will be deleted.

Requires authentication: Yes

Request

Parameters:

- **_G_O**: (can be multiple elements)
  - **G**: Group identifier
  - **O**: Privilege (Operation) identifier. Privilege identifiers:
  - 0: view
  - 1: download
  - 2: editing
  - 3: notify
  - 4: dynamic
  - 5: featured
  - Group and Operation Identifiers can be obtained using *Site Information (xml.info)* service.

Example request:

POST:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.metadata.batch.update.privileges

Mime-type:
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <_1_2 />
  <_1_1 />
</request>
```

GET:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.metadata.batch.update.privileges?_1_2&_1_1
```

Response

If the request executed successfully then HTTP 200 status code is returned and an XML document with a summary of how the metadata records in the selected set have been processed. An example of such a response is shown below:

```
<response>
  <done>5</done>
  <notOwner>0</notOwner>
  <notFound>0</notFound>
</response>
```

The response fields are:

- **done** - number of metadata records successfully updated
- **notOwner** - number of metadata records skipped because the user running this service did not have ownership rights
- **notFound** - number of metadata records skipped because they were not found (may have been deleted)

If the request fails an HTTP 500 status code error is returned and the response is an XML document with the exception. An example of such a response is shown below:

```
<error id="service-not-allowed">
  <message>Service not allowed</message>
  .....
  <object>xml.metadata.batch.update.privileges</object>
  .....
</error>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or their profile has no rights to execute the service. Returns 500 HTTP code
- **ERROR: insert or update on table "operationallowed" violates foreign key 'operationallowed_operationid_fkey »**, if an operation identifier provided is not valid. Returns 500 HTTP code
- **ERROR: insert or update on table "operationallowed" violates foreign key 'operationallowed_groupid_fkey »**, if a group identifier provided is not valid. Returns 500 HTTP code

6.10 Metadata Ownership services

These services allow retrieval and management of metadata ownership where the 'owner' of a metadata record is the user who created it. Only users with **Administrator** and **UserAdmin** profiles can execute these services.

6.10.1 Batch new owner (xml.metadata.batch.newowner)

The **xml.metadata.batch.newowner** service allows you to set the owner of a selected set of metadata records.

Note: This service requires a previous call to the `xml.metadata.select` service (see *Select metadata records (xml.metadata.select)*) to select metadata records.

Note: Only those metadata records for which the user running the service has ownership rights on will be updated. If metadata versioning is on then ownership changes will be recorded in the version history.

Requires authentication: Yes

Request

Once the metadata records have been selected the **xml.metadata.batch.newowner** service can be invoked with the following parameters:

- **user**: (mandatory) New owner user identifier
- **group**: (mandatory) New owner group user identifier

Request example:

POST:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.metadata.batch.newowner

Mime-type:
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <user>2</user>
  <group>2</group>
</request>
```

GET:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.metadata.batch.newowner?user=1&group=1
```

Response

If the request executed successfully then HTTP 200 status code is returned and an XML document with a summary of how the metadata records in the selected set have been processed. An example of such a response is shown below:

```
<response>
  <done>5</done>
  <notOwner>0</notOwner>
  <notFound>0</notFound>
</response>
```

The response fields are:

- **done** - number of metadata records successfully updated
- **notOwner** - number of metadata records skipped because the user running this service did not have ownership rights

- **notFound** - number of metadata records skipped because they were not found (may have been deleted)

If the request fails an HTTP 500 status code error is returned and the response is an XML document with the exception. An example of such a response is shown below:

```
<error id="service-not-allowed">
  <message>Service not allowed</message>
  <class>ServiceNotAllowedEx</class>
  .....
</error>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or their profile has no rights to execute the service. Returns 500 HTTP code

6.10.2 Transfer ownership (xml.ownership.transfer)

The **xml.ownership.transfer** service can be used to transfer ownership and privileges of metadata from one user to another. This service should be used with data retrieved from previous invocations to the services *xml.ownership.editors* and *xml.ownership.groups* as described below.

Requires authentication: Yes

Request

Parameters:

- **sourceUser**: (mandatory) Identifier of the user whose metadata will be transferred to a new owner
- **sourceGroup**: (mandatory) Identifier of one of the user groups of sourceUser
- **targetUser**: (mandatory) Identifier of the user who will become the new owner of the metadata currently owned by sourceUser
- **targetGroup**: (mandatory) Identifier of one of the user groups of the targetUser

Example: In the next example we are going to transfer the ownership and privileges of metadata owned of user John (id=2) in group RWS (id=5) to user Samantha(id=7) in group NLR (id=6)

Transfer ownership request example:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.ownership.transfer

Mime-type:
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <sourceUser>2</sourceUser>
  <sourceGroup>5</sourceGroup>
```

```
  <targetUser>7</targetUser>
  <targetGroup>6</targetGroup>
</request>
```

Response

The response contains the following fields:

- **response**: This is the container for the response
  - **privileges**: Number of privileges transferred from source group to target group
  - **metadata**: Number of metadata records transferred from source user to target user

Transfer ownership response example:

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
  <privileges>4</privileges>
  <metadata>2</metadata>
</response>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or his profile has no rights to execute the service. Returns 500 HTTP code
- **Missing parameter (error id: missing-parameter)**, when mandatory parameters are not provided. Returns 500 HTTP code
- **bad-parameter XXXX**, when a mandatory parameter is empty or invalid. Returns 500 HTTP code

6.10.3 Retrieve metadata owners (xml.ownership.editors)

The **xml.ownership.editors** service can be used to retrieve the users with editor profile that own metadata records.

Requires authentication: Yes

Request

Parameters:

- **None**

Retrieve metadata owners request example:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.ownership.editors

Mime-type:
application/xml
```

```
Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request />
```

Response

Successful execution returns HTTP status code 200 and an XML document with the results. The elements of the response are as follows:

- **root**: This is the container for the response
  - **editor**: Container for each editor user information
    - **id**: User identifier
    - **username**: User login
    - **name**: User name
    - **surname**: User surname
    - **profile**: User profile

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <editor>
    <id>1</id>
    <username>admin</username>
    <name>admin</name>
    <surname>admin</surname>
    <profile>Administrator</profile>
  </editor>
  <editor>
    <id>2</id>
    <username>samantha</username>
    <name>Samantha</name>
    <surname>Smith</surname>
    <profile>Editor</profile>
  </editor>
</root>
```

Unsuccessful execution returns HTTP 500 status code error and an XML document describing the exception that occurred. An example of such an error response is::

```
<error id="service-not-allowed">
  <message>Service not allowed</message>
  <class>ServiceNotAllowedEx</class>
  .....
</error>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or his profile has no rights to execute the service. Returns 500 HTTP code

6.10.4 Retrieve groups & users that can be used in metadata ownership transfer (xml.ownership.groups)

The **xml.ownership.groups** service retrieves:

- all groups that have been assigned privileges over the metadata records owned by the specified user - these will be the source groups from which ownership can be transferred
- all groups to which the user running the service belongs to. A list of the users assigned to the group who have the editor profile is provided with each group. These are the target groups and editors to which ownership can be transferred.

Typically the *Retrieve metadata owners (xml.ownership.editors)* service is used to extract the user ids of editors that are used as parameters to retrieve more detailed information about source groups and target groups & editors.

Request

Parameters:

- **id**: (mandatory) User identifier of the user from whom metadata records will be transferred
- The user id of the user running this service will be used to obtain a list of target groups and editors to which the metadata records belonging to user **id** can be transferred.

Retrieve ownership groups request example:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.ownership.groups

Mime-type:
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <id>2</id>
</request>
```

Response

Successful execution returns HTTP status code 200 and an XML document with the results. The elements of the response are as follows:

- **root**: This is the container for the response
- **response**: This is the container for the response

- **group**: A group which has privileges over the metadata records owned by the user with user id **id** (can be multiple **group** elements). These groups can be used as the source group list for the transfer ownership service.
- **id, name, description, email, referrer, label**: Group information
- **targetGroup**: A user group to which the user running this service has been assigned (can be multiple **targetGroup** elements). The groups can be used as the target group list and the editors from the groups can be target editors for the transfer ownership service.
  - **id, name, description, email, referrer, label**: Group information
  - **editor**: Users from the group that can edit metadata (can be multiple **editor** elements)
  - **id,surname, name**: Metadata user owner information

Response example:

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
  <group>
    <id>3</id>
    <name>bigmetadatausers</name>
    <description>Big Metadata User Groups</description>
    <email>bigmetadatagroup@mail.net</email>
    <referrer />
    <label>
      <en>Big Metadata Users</en>
    </label>
  </group>
  <targetGroup>
    <id>2</id>
    <name>sample</name>
    <description>Demo group</description>
    <email>group@mail.net</email>
    <referrer />
    <label>
      <en>Sample group</en>
    </label>
    <editor>
      <id>12</id>
      <surname />
      <name />
    </editor>
    <editor>
      <id>13</id>
      <surname />
      <name>Samantha</name>
    </editor>
  </targetGroup>
  <targetGroup>
    <id>6</id>
    <name>RWS</name>
    <description />
    <email />
    <referrer />
    <label>
```

```
      <en>RWS</en>
    </label>
    <editor>
      <id>7</id>
      <surname />
      <name>Samantha</name>
    </editor>
  </targetGroup>
  ...
</response>
```

Unsuccessful execution returns HTTP 500 status code error and an XML document describing the exception that occurred. An example of such an error response is::

```
<error id="service-not-allowed">
  <message>Service not allowed</message>
  <class>ServiceNotAllowedEx</class>
  .....
</error>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or his profile has no rights to execute the service. Returns 500 HTTP code

6.11 Metadata Status services

6.11.1 Update Status on a metadata record (xml.metadata.status)

The **xml.metadata.status** service updates the status on a metadata record using the status and changeMessage provided as parameters.

Note: The previously assigned status will be removed. If versioning for the metadata record is on, then the previously assigned status will be available in the version history.

Requires authentication: Yes

Request

Parameters:

- **id** or **uuid**: Identifier of metadata to update
- **status**: One of the status identifiers take from the database table `statusvalues`. Status identifiers can be retrieved using the *Site Information (xml.info)* service. The core status identifiers are:
- 0: unknown
- 1: draft

- 2: approved
- 3: retired
- 4: submitted
- 5: rejected
- **changeMessage**: description of why the status has changed.

Request example:

POST:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.metadata.status

Mime-type:
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <id>6</id>
  <status>5</status>
  <changeMessage>Completely unacceptable: consistency rules ignored<changeMessage/>
</request>
```

GET:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.metadata.status?id=6&status=5&changeMessage
```

Note: URL encoding of changeMessage.

Response

If the request executes successfully then HTTP status code 200 is returned along with an XML document which contains the identifier of the metadata whose status has been updated.

Example:

```
<response>
  <id>6</id>
</response>
```

If an error occurred then HTTP status code 500 is returned along with an XML document which contains details of what went wrong. An example of such an error response is:

```
<error id="metadata-not-found">
  <message>Metadata not found</message>
  <class>MetadataNotFoundEx</class>
  .....
</error>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or their profile has no rights to execute the service. Returns 401 HTTP code
- **Metadata not found (error id: metadata-not-found)** if a metadata record with the identifier provided does not exist
- **Only the owner of the metadata can set the status. User is not the owner of the metadata**, if the user does not have ownership rights over the metadata record.

6.11.2 Batch update status (xml.metadata.batch.update.status)

The **xml.metadata.batch.update.status** service updates the status on a selected set of metadata using the status and changeMessage sent as parameters.

Note: This service requires a previous call to the `xml.metadata.select` service (see *Select metadata records (xml.metadata.select)*) to select metadata records.

Note: Only those metadata records for which the user running the service has ownership rights on will be updated and all status values previously assigned will be deleted. If metadata versioning is on then status changes will be recorded in the version history.

Requires authentication: Yes

Request

Parameters:

- **status**: One of the status identifiers take from the database table `statusvalues`. Status identifiers can be retrieved using the *Site Information (xml.info)* service. The core status identifiers are:
- 0: unknown
- 1: draft
- 2: approved
- 3: retired
- 4: submitted
- 5: rejected
- **changeMessage**: description of why the status has changed.

Example request:

POST:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.metadata.batch.update.status

Mime-type:
```

```
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <status>5</status>
  <changeMessage>Completely unacceptable: consistency rules ignored<changeMessage/>
</request>
```

GET:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.metadata.batch.update.status?&status=5&char
```

Note: URL encoding of changeMessage.

Response

If the request executed successfully then HTTP 200 status code is returned and an XML document with a summary of how the metadata records in the selected set have been processed. An example of such a response is shown below:

```
<response>
  <done>5</done>
  <notOwner>0</notOwner>
  <notFound>0</notFound>
  <noChange>0</noChange>
</response>
```

The response fields are:

- **done** - number of metadata records successfully updated
- **notOwner** - number of metadata records skipped because the user running this service did not have ownership rights
- **notFound** - number of metadata records skipped because they were not found (may have been deleted)
- **noChange** - number of metadata records whose ownership was unchanged by the operation.

If the request fails an HTTP 500 status code error is returned and the response is an XML document with the exception. An example of such a response is shown below:

```
<error id="service-not-allowed">
  <message>Service not allowed</message>
  <class>ServiceNotAllowedEx</class>
  .....
</error>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or their profile has no rights to execute the service. Returns 500 HTTP code

6.11.3 Get status of a metadata record (xml.metadata.status.get)

This service gets the status of a particular metadata record specified by id or uuid as a parameter.

Requires authentication: No.

Request

Parameters:

- **id** or **uuid**: Identifier of metadata to obtain status of.

Example request:

POST:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.metadata.status.get

Mime-type:
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <id>5</id>
</request>
```

GET:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.metadata.status.get?&id=5
```

Response

If the request executed successfully a HTTP 200 status code is returned and the XML with status values for the metadata record (note: all changesin status are returned in the response) is returned. An example follows:

```
<response>
  <record>
   <statusid>5</statusid>
   <userid>4</userid>
   <changedate>2012-12-27T14:58:04</changedate>
   <changemessage>Do it all again</changemessage>
   <name>rejected</name>
  </record>
  <record>
   <statusid>4</statusid>
   <userid>6</userid>
```

```
   <changedate>2012-12-27T14:32:10</changedate>
   <changemessage>Ready for review</changemessage>
   <name>submitted</name>
  </record>
 </response>
```

If the request did not execute successfully then HTTP 500 status code error is returned along with an XML document which includes details of the exception/what went wrong. An example of such a request is::

```
<error id="metadata-not-found">
  <message>Metadata not found</message>
  <class>MetadataNotFoundEx</class>
  .....
</error>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or their profile has no rights to execute the service. Returns 500 HTTP code
- **Metadata not found (error id: metadata-not-found)**, when the metadata record requested is not found. Returns 500 HTTP code

6.11.4 Defining status actions

The behaviour of GeoNetwork when a status changes can be defined by the programmer. See *java_metadata_status_actions*.

6.12 Metadata Category services

6.12.1 Update Categories of a metadata record (xml.metadata.category)

The **xml.metadata.category** service updates the categories of a metadata record using the list of categories provided.

Note: The previously assigned categories will be removed. If versioning for the metadata record is on, then the previously assigned categories will be available in the version history.

Requires authentication: Yes

Request

Parameters:

- **id** or **uuid**: Identifier of metadata to update
- **_C**: (can be multiple elements)

- **C**: Category identifier (integer). A list of categories and identifiers is stored in the categories table. It can be retrieved using the *Site Information (xml.info)* service.

Request example:

POST:

```
Url:
http://localhost:8080/geonetwork/srv/en/xml.metadata.category

Mime-type:
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <id>6</id>
  <_1/>
  <_2/>
</request>
```

GET:

```
Url:
http://localhost:8080/geonetwork/srv/en/xml.metadata.category?id=6&_1&_2
```

Response

Successful response (HTTP code 200) contains the identifier of the metadata whose categories have been updated.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <id>6</id>
</request>
```

Unsuccessful response (HTTP code 500) is an XML document with details of the exception/problem that occurred:

Example:

```
<error id="metadata-not-found">
  <message>Metadata not found</message>
  <class>MetadataNotFoundEx</class>
  .....
  <object>Metadata not found --> 6</object>
  <request>
     <language>eng</language>
     <service>xml.metadata.category</service>
  </request>
</error>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or their profile has no rights to execute the service. Returns 500 HTTP code.
- **Metadata not found (error id: metadata-not-found)** if a metadata record with the identifier provided does not exist. Returns 500 HTTP code.

6.12.2 Batch update categories (xml.metadata.batch.update.categories)

The **xml.metadata.batch.update.categories** service updates the categories of a selected set of metadata using the categories sent as parameters.

Note: This service requires a previous call to the `xml.metadata.select` service (see *Select metadata records (xml.metadata.select)*) to select the metadata records to update.

Note: Only those metadata records for which the user running the service has ownership rights on will be updated and all categories previously assigned will be deleted. If metadata versioning is on then category changes will be recorded in the version history.

Requires authentication: Yes

6.12.3 Request

Parameters:

- **_C**: (can be multiple elements)
  - **C**: Category identifier (integer). A list of categories and identifiers is stored in the categories table. It can be retrieved using the *Site Information (xml.info)* service.

Example request:

POST:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.metadata.batch.update.categories

Mime-type:
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <_1/>
  <_2/>
</request>
```

GET:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.metadata.batch.update.categories?_1&_2
```

Response

If the request executed successfully then HTTP 200 status code is returned and an XML document with a summary of how the metadata records in the selected set have been processed. An example of such a response is shown below:

```
<response>
        <done>5</done>
  <notOwner>0</notOwner>
  <notFound>0</notFound>
</response>
```

The response fields are:

- **done** - number of metadata records successfully updated
- **notOwner** - number of metadata records skipped because the user running this service did not have ownership rights
- **notFound** - number of metadata records skipped because they were not found (may have been deleted)

If the request fails an HTTP status code error is returned and the response is an XML document with the exception. An example of such a response is shown below:

```
<error id="service-not-allowed">
  <message>Service not allowed</message>
  .....
  <object>xml.metadata.batch.update.categories</object>
  .....
</error>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or their profile has no rights to execute the service. Returns 500 HTTP code

6.13 Metadata Versioning services

6.13.1 Start versioning a metadata record (xml.metadata.version)

The **xml.metadata.version** service creates an initial version of the metadata record and its properties (categories, status, privileges) in the subversion repository.

Requires authentication: Yes

Request

Parameters:

- **id** or **uuid**: Identifier of metadata to version

Request example:

POST:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.metadata.version

Mime-type:
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <id>6</id>
</request>
```

GET:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.metadata.version?id=6
```

Response

Successful response (HTTP status code 200) contains the identifier of the metadata for which versioning has been enabled.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <id>6</id>
</request>
```

If the service was not completed successfully, then HTTP status code 500 is returned with an XML document containing details of the exception/problem. An example of such a document is as follows::

```
<error id="metadata-not-found">
  <message>Metadata not found</message>
  <class>MetadataNotFoundEx</class>
  .....
  <object>Metadata not found --> 6</object>
  <request>
    <language>eng</language>
    <service>xml.metadata.version</service>
  </request>
</error>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or their profile has no rights to execute the service. Returns 500 HTTP code
- **Metadata not found (error id: metadata-not-found)** if a metadata record with the identifier provided does not exist. Returns 500 HTTP code

- **Operation Not Allowed**, if the user does not have editing rights over the metadata record. Returns 500 HTTP code

6.13.2 Batch start versioning (xml.metadata.batch.version)

For each metadata record in the selected set, **xml.metadata.batch.version** creates an initial version of the metadata record and its properties (categories, status, privileges) in the subversion repository.

Note: This service requires a previous call to the `xml.metadata.select` service (see *Select metadata records (xml.metadata.select)*) to select metadata records.

Note: Only those metadata records that the user running the service has editing rights over will be versioned. If a metadata record is already versioned then no action is taken.

Requires authentication: Yes

Request

Parameters:

None

Example request:

POST:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.metadata.batch.version

Mime-type:
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request/>
```

GET:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.metadata.batch.version
```

Response

If the request executed successfully then HTTP 200 status code is returned and an XML document with a summary of how the metadata records in the selected set have been processed. An example of such a response is shown below:

```
<response>
  <done>5</done>
  <notOwner>0</notOwner>
  <notFound>0</notFound>
</response>
```

The response fields are:

- **done** - number of metadata records successfully updated
- **notOwner** - number of metadata records skipped because the user running this service did not have ownership rights
- **notFound** - number of metadata records skipped because they were not found (may have been deleted)

If the request fails an HTTP 500 status code error is returned and the response is an XML document with the exception. An example of such a response is shown below:

```
<error id="service-not-allowed">
  <message>Service not allowed</message>
  .....
  <object>xml.metadata.batch.update.version</object>
  .....
</error>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or their profile has no rights to execute the service. Returns 500 HTTP code

6.14 Metadata Processing services

These services allow processing of one or more metadata records using an XSLT.

6.14.1 Rules for constructing a process XSLT for use with these services

Typically an XSLT used with this service will accept some parameters then filter the elements of the metadata record being processed, changing some elements and copying the rest. So the rules for constructing such an XSLT are:

- Accept parameters using `xsl:param` - values for these will be specified as part of the request. eg. `<xsl:param name="url"/>`
- Add templates that match and process the required metadata elements. For example:

```
<xsl:template match="gmd:identificationInfo/*">
       <!-- Do some work processing and/or copying contents of this element -->
</xsl:template>
```

- Include a template that matches all content, refers any specific matches to templates provided in the previous step or just copies the metadata elements (nodes) and attributes without changing them.

```
<xsl:template match="@*|node()">
  <xsl:copy>
    <xsl:apply-templates select="@*|node()"/>
  </xsl:copy>
</xsl:template>
```

- Put the process XSLT into the process directory of the relevant metadata schema plugin. eg. if your process XSLT applies to iso19139 metadata records then it should be in the process directory of the iso19139 schema (`GEONETWORK_DATA_DIR/config/schema_plugins/iso19139/process`).

6.14.2 Process a metadata record with an XSLT (xml.metadata.processing)

This service applies an XSLT to a metadata record specified by **id** or **uuid**.

Request

Parameters:

- **id** or **uuid**: Identifier of metadata to process.
- **save**: Set to '1' to save the processed metadata (default), '0' will not save the processed metadata and will return the processed metadata for inspection.
- **process**: Name of an XSLT in the process directory of a metadata schema in GeoNetwork. For example, `anonymizer.xsl` exists in the process directory of metadata schema iso19139 - to use this XSLT you would specific `anonymizer` as the process parameter value.
- Parameters of the process XSLT in order. Each parameter of the process XSLT needs to be specified with a value if no default exists in the process XSLT or with no value if the default is suitable. You will need to examine the process XSLT to determine which parameters to specify and what the default values are if any.

Example request for the anonymizer process XSLT:

POST:

```
Url:
http://localhost:8080/geonetwork/srv/en/xml.metadata.processing

Mime-type:
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <id>6</id>
  <save>0</save>
  <process>anonymizer</process>
  <email>john.p.bead@bonce.com<email/>
</request>
```

GET:

```
Url:
http://localhost:8080/geonetwork/srv/en/xml.metadata.processing?&id=6&save=0&process=anc
```

Response

If the processing specified in the request succeeded and the parameter `save` was set to '1' or left out, then the XML response contains the id of the metadata record that was processed. For example:

```
<response>
 <id>1</id>
</response>
```

If the processing specified in the request succeeded and the parameter `save` was set to '0', then the XML response contains the id of the metadata record and the processed metadata record. For example, if processing an iso19139 metadata record then the response would contain the processed iso19139 metadata record as follows:

```
<response>
 <id>2</id>
 <record>
   <gmd:MD_Metadata ...>
     .....
   </gmd:MD_Metadata>
 </record>
</response>
```

If the processing specified in the request failed, an XML error response is returned with the reason. For example, here is the response when processing was requested on a metadata record belonging to a metadata schema that does not have the specified processing XSLT:

```
<error id="bad-parameter">
  <message>Processing failed</message>
  .....
  <object>Not found:0, Not owner:0, No process found:1.</object>
  .....
</error>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or their profile has no rights to execute the service. Returns 500 HTTP code
- **Bad Parameter (error id: bad-parameter)**, when the processing (XSLT transform) returns an empty metadata record (explanation is returned in XML - see example response above). Returns 500 HTTP code

6.14.3 Batch process metadata records with an XSLT (xml.metadata.batch.processing)

The **xml.metadata.batch.processing** service applies an XSLT to each metadata record in a selected set of metadata records.

Note: This service requires a previous call to the `xml.metadata.select` service (see *Select metadata records (xml.metadata.select)*) to select metadata records.

Note: This service is only available to users with UserAdmin or Administrator profile.

Note: Only those metadata records for which the user running the service has editing rights on will be processed. If metadata versioning is on then any changes to the metadata records will be recorded in the version history.

Requires authentication: Yes

Request

Parameters:

- **save**: Set to '1' to save the processed metadata (default), '0' will not save the processed metadata.
- **process**: Name of an XSLT in the process directory of a metadata schema in GeoNetwork. For example, the anonymizer XSLT exists in the process directory of metadata schema iso19139 - to use this XSLT you would specific `anonymizer` as the process parameter value.
- Parameters of the process XSLT in order. Each parameter of the process XSLT needs to be specified with a value if no default exists in the process XSLT or with no value if the default is suitable. You will need to examine the process XSLT to determine which parameters to specify and what the default values are if any.

Example request for the anonymizer process XSLT:

POST:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.metadata.batch.processing

Mime-type:
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <save>0</save>
  <process>anonymizer</process>
  <email>john.p.bead@bonce.com<email/>
</request>
```

GET:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.metadata.batch.processing?&save=0&process=a
```

Response

If the request executed successfully then HTTP 200 status code is returned and an XML document with a summary of how the metadata records in the selected set have been processed. An example of such a response is shown below:

```
<response>
  <done>5</done>
  <notProcessFound>2</notProcessFound>
  <notOwner>0</notOwner>
  <notFound>0</notFound>
</response>
```

The response fields are:

- **done** - number of metadata records successfully updated
- **notProcessFound** - number of metadata records skipped because the process XSLT was not present in their metadata schema
- **notOwner** - number of metadata records skipped because the user running this service did not have ownership rights
- **notFound** - number of metadata records skipped because they were not found (may have been deleted)

If the request fails an HTTP 500 status code error is returned and the response is an XML document with the exception. An example of such a response is shown below:

```
<error id="service-not-allowed">
  <message>Service not allowed</message>
  .....
  <object>xml.metadata.batch.processing</object>
  .....
</error>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or their profile has no rights to execute the service. Returns 500 HTTP code

6.14.4 Batch update child records (xml.metadata.batch.update.children)

The **xml.metadata.batch.update.children** service copies metadata elements from the parent metadata record to all child metadata elements.

- This service works only for iso19139 (or profile) child metadata records ie. metadata records whose gmd:parentIdentifier is set to the uuid of a metadata record in the catalog.
- Any child metadata records that do not have the same metadata schema as the parent metadata record will be skipped.
- The service actually executes an XSLT in the metadata schema directory of the parent metadata record. The XSLT is called `update-child-from-parent-info.xsl`. It is run on each child metadata record and is passed parameters from the request as required. This design has been chosen to make customization of the service reasonably straight forward.

Note: If user of this service does not have edit privileges over a child metadata record then that record will be skipped.

Requires authentication: Yes

Request

Parameters:

- **id**: GeoNetwork internal integer id of parent metadata record.
- **parentUuid**: Uuid of parent metadata record.
- **schema**: Metadata schema name in GeoNetwork.
- **childrenIds**: GeoNetwork internal integer ids of child metadata records (comma separated)
- **updateMode**: 'replace' means replace content in the children with content from the parent. 'add' means add content from the parent to the child metadata records.
- Parameters for `update-child-from-parent-info.xsl`. Examine the relevant XSLT to determine which parameters to specify.

Example request:

POST:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.metadata.batch.update.children

Mime-type:
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request>
 <id>1</id>
 <parentUuid>da165110-88fd-11da-a88f-000d939bc5d8</parentUuid>
 <childrenIds>4,</childrenIds>
 <schema>iso19139</schema>
 <updateMode>replace</updateMode>
 <gmd-descriptiveKeywords>true</gmd-descriptiveKeywords>
 <gmd-contact>true</gmd-contact>
 <gmd-extent>true</gmd-extent>
 <gmd-pointOfContact>true</gmd-pointOfContact>
 <gmd-metadataMaintenance>true</gmd-metadataMaintenance>
</request>
```

GET:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.metadata.batch.update.children?&id=1&parent
```

Response

If the request executed successfully a HTTP 200 status code is returned and some XML describing what was processed. An example of such an XML response is:

```
<response>1 child/children updated for metadata da165110-88fd-11da-a88f-000d939bc5d8.</r
```

If the request fails an HTTP 500 status code error is returned and the response contains an XML document with details of the exception. An example of such a response is:

```
<error id="metadata-not-found">
  <message>Metadata not found</message>
  <class>MetadataNotFoundEx</class>
  .....
  <object>Could not find metadata parent record --> 1</object>
```

```
  .....
</error>
```

See *Exception handling* for more details.

Errors

- **Service not allowed (error id: service-not-allowed)**, when the user is not authenticated or their profile has no rights to execute the service. Returns 500 HTTP code
- **Metadata not found (error id: metadata-not-found)**, when the parent metadata record doesn't exist. Returns 500 HTTP code

6.15 Metadata Relation services

This section describes the services used to show, get, insert and delete relations between metadata records in GeoNetwork. If the metadata schema has elements that support relationships between metadata records (eg. ISO19115/19139), then the relationships are stored in the Lucene index with the metadata record. If a relationship concept does not exist in the metadata schema, then the relationship is stored in the Relations table as follows:

Field	Datatype	Description
id	foreign key to Metadata(id)	Source metadata
relatedId	foreign key to Metadata(id)	Metadata related to the source

6.15.1 Get all related records for a metadata record (xml.relation)

This service retrieves all the related records for a source metadata record specified by id in the parameters. The relationships can come from the Lucene index and/or the Relations table in the database.

Request

- **id (integer)**: This is the local GeoNetwork identifier of the metadata whose relations are requested.

Here is an example of POST/XML request:

```
<request>
    <id>10</id>
</request>
```

Response

If the request executed successfully then HTTP status code 200 is returned along with an XML document containing a relations root element and a relation child for each type of relation found. Example:

```
<relations>
  <relation type="parent">
     <id>3</id>
     <uuid>da165110-88fd-11da-a88f-000d939bc5d8</uuid>
```

```
      <title>....</title>
      <abstract>....</abstract>
  </relation>
</relations>
```

Each relation element has a type attribute indicating the type of relation with the metadata record id specified in the parameters. The XML elements returned vary for each type attribute as follows:

- **relation type=parent**: elements describe the parent metadata record of the specified metadata record:
- **id**: GeoNetwork internal id (integer)
- **uuid**: Metadata uuid
- **title**: Metadata title
- **abstract**: A brief explanation of the metadata
- **relation type=children**: **metadata** element describes the child metadata record of the specified metadata record
- **metadata**: container for child metadata record
  - **title**: Metadata title
  - **abstract: A brief explanation of the metadata**
    - *Other elements returned by a brief presentation of the child metadata record*
- **relation type=services**: multiple **metadata** elements describing the service metadata records that operate on the specified metadata record:
- **metadata**: container for a service metadata record
  - **title**: Metadata title
  - **abstract: A brief explanation of the metadata**
    - *Other elements returned by a brief presentation of the service metadata record*
- **relation type=fcats**: multiple **metadata** elements describing the feature catalog metadata records that are related to the specified metadata record
- **metadata**: container for feature catalog metadata record
  - **title**: Metadata title
  - **abstract: A brief explanation of the metadata**
    - *Other elements returned by a brief presentation of the feature catalog metadata record*
- **relation type=hasfeaturecat**: **metadata** element describing the metadata record that has a feature catalog relation to this feature catalog record
- **metadata**: container for metadata record
  - **title**: Metadata title
  - **abstract: A brief explanation of the metadata**
    - *Other elements returned by a brief presentation of the metadata record*

- **relation type=related**: elements describe a related metadata record to the specified metadata record:
- **id**: GeoNetwork internal id (integer)
- **uuid**: Metadata uuid
- **title**: Metadata title
- **abstract**: A brief explanation of the metadata

If the response did not execute successfully then an HTTP error code 500 is returned along with an XML document describing the exception/what went wrong. An example of such an error response is::

```
<error id="metadata-not-found">
  <message>Metadata not found</message>
  <class>MetadataNotFoundEx</class>
  .....
</error>
```

See *Exception handling* for more details.

6.15.2 Manage Relationships in the Relations table

These services manage relationships between metadata records that are held in the Relations table ie. they are relationships that cannot be held in a metadata record.

xml.relation.get

This service retrieves all the related records for a source metadata record specified by id in the parameters. The related records are those that are in the Relations table ie. they are those that cannot be held in a metadata record.

Request

- **id (integer)** or **uuid**: This is the local GeoNetwork identifier of the metadata or uuid of metadata whose relations are requested.
- **relation (string, 'normal')**: This optional parameter identifies the kind of relation that the client wants to be returned. It can be one of these values:
  - **normal**: The service performs a query into the id field and returns all relatedId records.
  - **reverse**: The service performs a query into the relatedId field and returns all id records.
  - **full**: Includes both normal and reverse queries (duplicated ids are removed).

Here is an example of POST/XML request:

```
<request>
    <id>10</id>
    <relation>full</relation>
</request>
```

Response

If the request executed successfully then HTTP status code 200 is returned along with an XML document containing a response root element and metadata children depending on the relations found. Example:

```
<response>
    <metadata>...</metadata>
    <metadata>...</metadata>
    ...
</response>
```

Each metadata element has the the structure returned by the brief template of the metadata schema presentation XSLT. Typical brief elements are:

- **title**: Metadata title
- **abstract**: A brief explanation of the metadata
- **keyword**: Keywords found inside the metadata
- **image**: Information about thumbnails
- **link**: A link to the source site
- **geoBox**: coordinates of the bounding box
- **geonet:info**: A container for GeoNetwork related information

Example of a brief metadata record presentation for *fgdc-std*:

```
<metadata>
    <title>Globally threatened species of the world</title>
    <abstract> Contains information on animals.</abstract>
    <keyword>biodiversity</keyword>
    <keyword>endangered animal species</keyword>
    <keyword>endangered plant species</keyword>
    <link type="url">http://www.mysite.org</link>
    <geoBox>
        <westBL>-180.0</westBL>
        <eastBL>180.0</eastBL>
        <southBL>-90.0</southBL>
        <northBL>90.0</northBL>
    </geoBox>
    <geonet:info>
        <id>11</id>
        <schema>fgdc-std</schema>
        <createDate>2005-03-31T19:13:31</createDate>
        <changeDate>2007-03-12T14:52:46</changeDate>
        <isTemplate>n</isTemplate>
        <title/>
        <source>38b75c1b-634b-443e-9c36-a12e89b4c866</source>
        <UUID>84b4190b-de43-4bd7-b25f-6ed47eb239ac</uuid>
        <isHarvested>n</isHarvested>
        <view>true</view>
        <admin>false</admin>
        <edit>false</edit>
        <notify>false</notify>
        <download>true</download>
        <dynamic>false</dynamic>
        <featured>false</featured>
```

```
    </geonet:info>
</metadata>
```

If the response did not execute successfully then an HTTP error code 500 is returned along with an XML document describing the exception/what went wrong. See *Exception handling* for more details.

Note: this service returns an empty response if the metadata record specified in the parameters doesn't exist.

---

xml.relation.insert

This service creates a relationship between a parent metadata record and a child metadata record. The relationship is held in the Relations table ie. relationships inserted using this service are those that cannot be held in a metadata record.

Request

- **parentId (integer)** or **parentUuid**: This is the identifier of the metadata which we are inserting a relationship for.
- **childId (integer)** or **parentUuid**: This is the identifier of the metadata which will be related to metadata record specified by **parentId** or **parentUuid**.

Here is an example of POST/XML request:

```
<request>
    <parentId>1</parentId>
    <childId>2</childId>
</request>
```

Response

Normally an HTTP status code 200 is returned along with an XML document containing a response root element with an attribute indicating whether the relationship already exists and the parentId and childId parameters from the request. Example:

```
<response alreadyExist="false">
  <parentId>1</parentId>
  <childId>2</childId>
</response>
```

Note: this service returns this response even if the metadata records specified in the parameters do not exist.

If the response did not execute successfully then an HTTP error code 500 is returned along with an XML document describing the exception/what went wrong. See *Exception handling* for more details.

xml.relation.delete

This service deletes a relationship between a parent metadata record and a child metadata record. The relationship is held in the Relations table ie. relationships removed using this service are those that cannot be held in a metadata record.

Request

- **parentId (integer)** or **parentUuid**: This is the identifier of the metadata which we are removing the relationship from.
- **childId (integer)** or **parentUuid**: This is the identifier of the metadata which is related to metadata record specified by **parentId** or **parentUuid**.

Here is an example of POST/XML request:

```
<request>
    <parentId>1</parentId>
    <childId>2</childId>
</request>
```

Response

Normally an HTTP status code 200 is returned along with an XML document with an empty response root element.

Note: this service returns an empty response regardless of whether the parent and/or child metadata metadata records specified in the id parameters exist or not.

If the response did not execute successfully then an HTTP error code 500 is returned along with an XML document describing the exception/what went wrong. See *Exception handling* for more details.

6.16 Metadata Validation services

These services are for validating metadata against the XML schema documents (XSDs) and schematrons specified as part of a GeoNetwork Metadata Schema. See *Schema Plugins* for more details.

6.16.1 Validate a metadata record (xml.metadata.validation)

This service can be used to validate a metadata record supplied as an XML parameter. The metadata record is first passed through the GeoNetwork schema detection rules (see *Schema Plugins*). After successful schema detection the metadata record is validated against the XML schema documents and schematrons (if any) specified in that schema.

Authentication required: No

Request

Parameters:

- **data**: Metadata record.

Example with an ISO19115/19139 metadata record:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.metadata.validation

Mime-type:
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <data>
    <gmd:MD_Metadata ....>
      .....
    </gmd:MD_Metadata>
  </data>
</request>
```

Response

If the validation is successful an HTTP 200 response code is returned along with an XML document giving details of the GeoNetwork metadata schema that the record matched and was successfully validated against.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
  <valid>y</valid>
  <schema>iso19139</schema>
</response>
```

If the validation was not successful then an HTTP 500 error response code is returned along with an XML document describing the validation problems. An example is:

```
<?xml version="1.0" encoding="UTF-8"?>
<error id="xsd-validation-error">
  <message>XSD Validation error(s):
    .....
  </message>
  <stack>
    .....
  </stack>
  <object>
   <xsderrors>
    <error>
      <typeOfError>WARNING</typeOfError>
      <errorNumber>1</errorNumber>
      <message>....</message>
      <xpath>.</xpath>
    </error>
```

```
   </xsderrors>
  </object>
  <request>
    <language>eng</language>
    <service>xml.metadata.validation</service>
  </request>
 </error>
```

Note: XML parseable description of the validation problems is in the object container.

Validation may also fail when schematrons are applied to the metadata record. An HTTP error response code is returned along with an XML document describing the validation problems. An example is:

```
<?xml version="1.0" encoding="UTF-8"?>
<error id="schematron-validation-error">^M
  <message>Schematron errors detected
    .....
  </message>
  <stack>
    .....
  </stack>
  <object>
  <geonet:schematronerrors xmlns:geonet="http://www.fao.org/geonetwork">
    <geonet:report geonet:rule="schematron-rules-iso">
      <svrl:schematron-output xmlns:svrl="http://purl.oclc.org/dsdl/svrl" xmlns:xlink="h
        <svrl:ns-prefix-in-attribute-values uri="http://www.opengis.net/gml" prefix="gml
        <svrl:ns-prefix-in-attribute-values uri="http://www.isotc211.org/2005/gmd" prefi
        <svrl:ns-prefix-in-attribute-values uri="http://www.isotc211.org/2005/srv" prefi
        <svrl:ns-prefix-in-attribute-values uri="http://www.isotc211.org/2005/gco" prefi
        <svrl:ns-prefix-in-attribute-values uri="http://www.fao.org/geonetwork" prefix="
        <svrl:ns-prefix-in-attribute-values uri="http://www.w3.org/1999/xlink" prefix="x
        <svrl:active-pattern document="" name="CharacterString must have content or its
        <svrl:fired-rule context="*[gco:CharacterString]" />
        <svrl:fired-rule context="*[gco:CharacterString]" />
        <svrl:fired-rule context="*[gco:CharacterString]" />
        <svrl:fired-rule context="*[gco:CharacterString]" />
        <svrl:fired-rule context="*[gco:CharacterString]" />
        <svrl:fired-rule context="*[gco:CharacterString]" />
        <svrl:fired-rule context="*[gco:CharacterString]" />
        <svrl:fired-rule context="*[gco:CharacterString]" />
        <svrl:fired-rule context="*[gco:CharacterString]" />
        <svrl:fired-rule context="*[gco:CharacterString]" />
        <svrl:active-pattern document="" name="CRS attributes constraints" />
        <svrl:active-pattern document="" name="[ISOFTDS19139:2005-TableA1-Row24] - A nam
        <svrl:fired-rule context="//*[gmd:CI_ResponsibleParty]" />
        <svrl:failed-assert ref="#_31" test="$count &gt; 0" location="/*[local-name()='M
          <svrl:text>
            <alert.M8>
              <div>
                You must specify one or more of individualName, organisationName or posi
                <span class="validationReportSuggestion">Suggestions: Check contact info
              </div>
            </alert.M8>
          </svrl:text>
        </svrl:failed-assert>
        <svrl:active-pattern document="" name="[ISOFTDS19139:2005-TableA1-Row07] - Other
```

```
      <svrl:active-pattern document="" name="[ISOFTDS19139:2005-TableA1-Row16] - Units
      <svrl:active-pattern document="" name="[ISOFTDS19139:2005-TableA1-Row13] - Descr
      <svrl:active-pattern document="" name="[ISOFTDS19139:2005-TableA1-Row10] - Conte
      <svrl:active-pattern document="" name="[ISOFTDS19139:2005-TableA1-Row11 Row12] -
      <svrl:active-pattern document="" name="[ISOFTDS19139:2005-TableA1-Row08] - Datas
      <svrl:active-pattern document="" name="[ISOFTDS19139:2005-TableA1-Row09] - Level
      <svrl:active-pattern document="" name="[ISOFTDS19139:2005-TableA1-Row17] - Units
      <svrl:active-pattern document="" name="[ISOFTDS19139:2005-TableA1-Row18] - Distr
      <svrl:active-pattern document="" name="[ISOFTDS19139:2005-TableA1-Row23] - Exter
      <svrl:active-pattern document="" name="[ISOFTDS19139:2005-TableA1-Row04] - Datas
      <svrl:fired-rule context="//gmd:MD_DataIdentification|//*[@gco:isoType='gmd:MD_D
      <svrl:successful-report ref="#_20" test="$extent = false()" location="/*[local-n
        <svrl:text>
          <report.M21>Extent defined for dataset.</report.M21>
        </svrl:text>
      </svrl:successful-report>
      <svrl:active-pattern document="" name="[ISOFTDS19139:2005-TableA1-Row05] - Datas
      <svrl:fired-rule context="//gmd:MD_DataIdentification|//*[@gco:isoType='gmd:MD_D
      <svrl:successful-report ref="#_20" test="$topic = false" location="/*[local-name
        <svrl:text>
          <report.M6>Topic category is:</report.M6>
          ""
        </svrl:text>
      </svrl:successful-report>
      <svrl:active-pattern document="" name="[ISOFTDS19139:2005-TableA1-Row06] - Eithe
      <svrl:active-pattern document="" name="[ISOFTDS19139:2005-TableA1-Row02] - Chara
      <svrl:fired-rule context="//gmd:MD_Metadata|//*[@gco:isoType='gmd:MD_Metadata']"
      <svrl:active-pattern document="" name="[ISOFTDS19139:2005-TableA1-Row19] - Detai
      <svrl:active-pattern document="" name="[ISOFTDS19139:2005-TableA1-Row20] - Condi
      <svrl:active-pattern document="" name="[ISOFTDS19139:2005-TableA1-Row21] - Domai
      <svrl:active-pattern document="" name="[ISOFTDS19139:2005-TableA1-Row22] - Short
      <svrl:active-pattern document="" name="[ISOFTDS19139:2005-TableA1-Row15] - Check
      <svrl:active-pattern document="" name="[ISOFTDS19139:2005-TableA1] - HierarchyLe
      <svrl:fired-rule context="//gmd:MD_Metadata/gmd:hierarchyLevel|//*[@gco:isoType=
      <svrl:successful-report ref="#_6" test="$hl = false()" location="/*[local-name()
        <svrl:text>
          <report.M61>Hierarchy level name is:</report.M61>
          "collectionSession"
        </svrl:text>
      </svrl:successful-report>
    </svrl:schematron-output>
  </geonet:report>
  <geonet:report geonet:rule="schematron-rules-geonetwork">
    <svrl:schematron-output xmlns:svrl="http://purl.oclc.org/dsdl/svrl" xmlns:xlink="h
      <svrl:ns-prefix-in-attribute-values uri="http://www.opengis.net/gml" prefix="gml
      <svrl:ns-prefix-in-attribute-values uri="http://www.isotc211.org/2005/gmd" prefi
      <svrl:ns-prefix-in-attribute-values uri="http://www.isotc211.org/2005/srv" prefi
      <svrl:ns-prefix-in-attribute-values uri="http://www.isotc211.org/2005/gco" prefi
      <svrl:ns-prefix-in-attribute-values uri="http://www.fao.org/geonetwork" prefix="
      <svrl:ns-prefix-in-attribute-values uri="http://www.w3.org/1999/xlink" prefix="x
      <svrl:active-pattern document="" name="[Language] - Metadata language is not def
      <svrl:fired-rule context="//gmd:MD_Metadata|//*[@gco:isoType='gmd:MD_Metadata']"
      <svrl:successful-report ref="#_1" test="$localeAndNoLanguage" location="/*[local
        <svrl:text>
          <report.M500>Main metadata language is:</report.M500>
          "eng"
        </svrl:text>
```

```
        </svrl:successful-report>
        <svrl:successful-report ref="#_1" test="$duplicateLanguage" location="/*[local-n
          <svrl:text>
            <report.M501>No duplicate languages found.</report.M501>
          </svrl:text>
        </svrl:successful-report>
      </svrl:schematron-output>
    </geonet:report>
  </geonet:schematronerrors>
  </object>
</error>
```

Note: XML parseable description of the schematron validation problems is in the object container. You should be looking for elements such as svrl:failed-assert.

See *Exception handling* for more details.

Errors

- **bad-parameter XXXX**, when a mandatory parameter is empty. Returns 500 HTTP code
- **XSD Validation Error (error id: xsd-validation-error)**, when validation against XSDs fails. Returns 500 HTTP code
- **Schematron Validation Error (error id: schematron-validation-error)**, when validation against schematrons fails. Returns 500 HTTP code
- **No Schema Matches (error id: no-schema-matches)**, when a matching GeoNetwork metadata schema cannot be found for the supplied metadata record. Returns 500 HTTP code

6.17 System configuration

6.17.1 Introduction

GeoNetwork configuration parameters can be changed to suit the needs of your site. There are two groups of parameters:

- parameters that can be changed through the web interface.
- parameters not accessible from the web interface and that must be changed when the system is not running

The first group of parameters can be queried or changed through the two services described in this section: `xml.config.get` and `xml.config.set`.

The second group of parameters must be changed manually by editing the `config-*.xml` files in `INSTALL_DIR/web/geonetwork/WEB-INF`.

The GAST tool can be used to help configure the database parameters (see the section on the GAST tool in the user manual).

6.17.2 xml.config.get

This service returns the system configuration parameters as an XML document.

Request

Parameters: *None*

Response

The response is an XML tree similar to the system hierarchy into the settings structure. The response has the following elements:

- **site**: A container for site information.
  - **name**: Site name.
  - **organisation**: Site organisation name.
  - **svnUuid**: Subversion Uuid (used for metadata versioning)
  - **siteId**: Uuid of site (used to uniquely identify site)
- **platform**: Details of development platform.
  - **version**: Version string of software.
  - **subVersion**: Additional version string.
- **server**: A container for server information.
  - **host**: Name of the host from which the site is reached.
  - **port**: Port number of the previous host.
  - **protocol**: http or https.
- **Intranet**: Information about the Intranet of the organisation.
  - **network**: IP address that specifies the intranet.
  - **netmask**: netmask used to identify intranet.
- **z3950**: Configuration of Z39.50 server.
  - **enable**: true means that the Z39.50 server component is running.
  - **port**: Port number to use to listen for incoming Z39.50 requests.
- **proxy**: Proxy configuration
  - **use**: true means proxy is used when connecting to external nodes.
  - **host**: Proxy server host.
  - **port**: Proxy server port.
  - **username**: Proxy credentials.
  - **password**: Proxy credentials.
- **feedback**: A container for feedback information

  - **email**: Feedback/Info email address
  - **mailServer**: Email server to use to send feedback emails
    * **host**: Email server address
    * **port**: Port number of email service on email server
- **removedMetadata**: A container for removed metadata information
  - **dir**: Folder used to store removed metadata in MEF format
- **ldap**: A container for LDAP parameters (see System Configuration in Users Manual for more information)
- **selectionmanager**: A container for selection manager configuration
  - **maxrecords**: Maximum number of records that can be selected
- **csw**: A container for csw server configuration
  - **enable**: CSW server is enabled if set to true.
  - **contactId**: Identifier of GeoNetwork user who is the contact for the CSW server.
  - **metadataPublic**: If set to true then metadata inserted through the CSW server will be made public immediately.
- **shib**: A container for Shibboleth parameters (see System Configuration in Users Manual for more information)
- **userSelfRegistration**: A container for user self-registration service configuration
  - **enable**: enabled if set to true.
- **clickablehyperlinks**: A container for configuration of clickable hyper-links in metadata content
  - **enable**: enabled if set to true. ie. hyperlinks in metadata content will be automatically turned into clickable HTML links
- **localrating**: A container for configuration of local rating versus remote rating
  - **enable**: local rating enabled if set to true.
- **downloadservice**: A container for configuration of file download interface on links built from ISO online resources with file download protocol
  - **leave**: don't build links or modify ISO online resources with file download protocol
  - **simple**: download file immediately when user clicks on link
  - **withdisclaimer**: when user clicks on link, display metadata resource restrictions and disclaimers before downloading file
- **xlinkResolver**: A container for configuration of XLink resolver service
  - **enable**: XLinks in metadata records will be resolved if set to true
- **autofixing**: A container for configuration of autofixing service
  - **enable**: Autofixing (ie. update-fixed-info.xsl) will be applied to metadata records when they are saved in the editor
- **searchStats**: A container for configuration of search statistics collection

  - **enable**: if true then search stats will collected on searches made through the GeoNetwork user interface

- **indexOptimzer**: A container to configure if and when Lucene index optimization will take place (likely to be deprecated in the next release of GeoNetwork)
  - **enable**: if true then enable optimization at the scheduled interval
- **oai**: A container to configure the Open Archives Initiative (OAI) server in GeoNetwork
  - **mdmode**: if '1' then OAI date searches uses the metadata temporal extent, if '2' then the modification date from the database is used
  - **tokentimeout**: time in seconds that a continuation token passed to a client can be used
  - **cachesize**: number of client sessions that the server can manage
- **inspire**: A container to configure the Inspire options in GeoNetwork
  - **enable**: if true then inspire indexing of ISO metadata will be enabled
  - **enableSearchPanel**: if true then inspire search panel will be shown in the search interface
- **harvester**: A container to configure harvesting options
  - **enableEditing**: if true then harvested records can be edited, false means editing will not be enabled
- **metadata**: A container to configure the different view/edit tabs shown to the user in the viewer/editor
  - **enableSimpleView**: simple (or default) mode means that only those elements present in the template/record will be displayed, new elements cannot be added - true means simple (or default) view is enabled, false means the tab will not be shown
  - **enableIsoView**: true means that tabs showing mandatory/core/all groupings of metadata elements will be present in the viewer/editor for ISO records
  - **enableInspireView**: true means that tabs showing inspire groupings of metadata elements will be present in the viewer/editor for ISO records
  - **enableXmlView**: true means that the tab showing the XML of the metadata record will be present in the viewer/editor
  - **defaultView**: 'simple', 'advanced', 'iso', 'xml' determines which tab will be the default view (ie. the view used when no previous view has been selected by the user in their current session).
- **threadedindexing**: A container to configure multi-threaded indexing
  - **maxThreads**: Number of threads to be used during multi-threaded indexing
- **autodetect**: A parameter to configure language detection in search terms
  - **enable**: if true then language detection is enabled
- **requestedLanguage**: A parameter to configure which indexes will be searched and which languages will be used to display results
  - **only**: 'off' - all languages ignored, 'prefer_locale' - prefer documents with translations to requested language, 'prefer_docLanguage' - prefer documents whose language is the requested language, 'only_locale' - translations in requested language, 'only_docLocale' - document language is the requested language

Example of xml.config.get response:

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
 <site>
   <name>My GeoNetwork catalogue</name>
   <organization>My organization</organization>
   <svnUuid>c7799284-e786-4425-a6cf-824bb07e478e</svnUuid>
   <siteId>b7ce20f2-888a-4139-8802-916730c4be06</siteId>
 </site>
 <platform>
   <version>2.8.0</version>
   <subVersion>RC2</subVersion>
 </platform>
 <server>
   <host>localhost</host>
   <port>8080</port>
   <protocol>http</protocol>
 </server>
 <intranet>
   <network>127.0.0.1</network>
   <netmask>255.0.0.0</netmask>
 </intranet>
 <z3950>
   <enable>true</enable>
   <port>2100</port>
 </z3950>
 <proxy>
   <use>false</use>
   <host/>
   <port/>
   <username/>
   <password/>
 </proxy>
 <feedback>
   <email/>
   <mailServer>
     <host/>
     <port>25</port>
   </mailServer>
 </feedback>
 <removedMetadata>
   <dir>WEB-INF/data/removed</dir>
 </removedMetadata>
 <ldap>
   <use>false</use>
   <host/>
   <port/>
   <defaultProfile>RegisteredUser</defaultProfile>
   <uidAttr>uid</uidAttr>
   <distinguishedNames>
     <base>dc=fao,dc=org</base>
     <users>ou=people</users>
   </distinguishedNames>
   <userAttribs>
     <name>cn</name>
     <profile>profile</profile>
     <group/>
```

```
    </userAttribs>
    <defaultGroup/>
  </ldap>
  <selectionmanager>
    <maxrecords>1000</maxrecords>
  </selectionmanager>
  <csw>
    <enable>true</enable>
    <contactId/>
    <metadataPublic>false</metadataPublic>
  </csw>
  <shib>
    <use>false</use>
    <path>/geonetwork/srv/en/shib.user.login</path>
    <attrib>
      <username>REMOTE_USER</username>
      <surname>Shib-Person-surname</surname>
      <firstname>Shib-InetOrgPerson-givenName</firstname>
      <profile>Shib-EP-Entitlement</profile>
      <group/>
      <organizationName/>
      <postalAddress/>
      <phone/>
      <email/>
      <fullName/>
    </attrib>
    <defaultGroup/>
  </shib>
  <userSelfRegistration>
    <enable>false</enable>
  </userSelfRegistration>
  <clickablehyperlinks>
    <enable>true</enable>
  </clickablehyperlinks>
  <localrating>
    <enable>false</enable>
  </localrating>
  <downloadservice>
    <leave>false</leave>
    <simple>true</simple>
    <withdisclaimer>false</withdisclaimer>
  </downloadservice>
  <xlinkResolver>
    <enable>false</enable>
  </xlinkResolver>
  <autofixing>
    <enable>true</enable>
  </autofixing>
  <searchStats>
    <enable>false</enable>
  </searchStats>
  <indexoptimizer>
    <enable>true</enable>
    <at>
      <hour>0</hour>
      <min>0</min>
      <sec>0</sec>
```

```
    </at>
    <interval>
      <day>0</day>
      <hour>24</hour>
      <min>0</min>
    </interval>
  </indexoptimizer>
  <oai>
    <mdmode>1</mdmode>
    <tokentimeout>3600</tokentimeout>
    <cachesize>60</cachesize>
  </oai>
  <inspire>
    <enable>false</enable>
    <enableSearchPanel>false</enableSearchPanel>
  </inspire>
  <harvester>
    <enableEditing>false</enableEditing>
  </harvester>
  <metadata>
    <enableSimpleView>true</enableSimpleView>
    <enableIsoView>true</enableIsoView>
    <enableInspireView>false</enableInspireView>
    <enableXmlView>true</enableXmlView>
    <defaultView>simple</defaultView>
  </metadata>
  <metadataprivs>
    <usergrouponly>false</usergrouponly>
  </metadataprivs>
  <threadedindexing>
    <maxthreads>1</maxthreads>
  </threadedindexing>
  <autodetect>
    <enable>false</enable>
  </autodetect>
  <requestedLanguage>
    <only>prefer_locale</only>
  </requestedLanguage>
</config>
```

6.17.3 xml.config.set

This service is used to update the system configuration. It is restricted to users with the *Administrator* profile.

Request

The request format is the same as the XML document produced by the `xml.config.get` service. To use the `xml.config.set` service in the simplest way:

1. Call `xml.config.get` to obtain an XML document describing the current configuration.
2. Update the content of the elements you want to change.
3. POST the modified XML document describing the new configuration to `xml.config.set`.

So a typical POST request would look like::

```
Url: http://localhost:8080/geonetwork/srv/eng/xml.config.set

<request>
  <config>
   .....
  </config>
</request>
```

Response

If the request executed successfully then HTTP status code 200 is returned along with an XML document confirming success. The success response is::

```
<response>ok</response>
```

If an exception occurred then an XML document with the details of the exception is returned.

See *Exception handling* for more details.

6.18 Site Information and Request Forwarding Services

Services in this section provide information about the site (eg. name, users, groups, schemas etc) and access to the site forwarding service which can be used by JavaScript clients.

6.18.1 Site Information (xml.info)

This service can be used to retrieve information about a GeoNetwork site. The information that can be requested includes: site name and id, users, groups, metadata schemas as well as lists of privileges, metadata status values, spatial regions, local metadata categories and so on.

Request

The XML request should contain at least one type parameter to indicate the kind of information to retrieve. Multiple type parameters can be specified. The set of allowed values is:

- **site**: Returns general information about the site like its name, id, etc...
- **users**: Depending upon the profile of the user making the call, information about users of the site will be returned. The rules are:
- Administrators can see all users
- User administrators can see all users they administer and all other user administrators in the same group set. The group set is defined by all groups visible to the user administrator (except for the All and Intranet groups).
- An authenticated user can only see their own information.
- A guest cannot see any user information at all.
- **groups**: Returns all user groups visible to the requesting user. Note: If the user is not authenticated, only the `Intranet` and `All` groups will be returned.

- **sources**: Returns all GeoNetwork sources (remote sites) that are known about at the site. This will include:
- Node name and siteId
- All source UUIDs and site names that have been discovered through harvesting
- All source UUIDs and site names from MEF files imported by the site
- **schemas**: Returns all registered metadata schemas for the site
- **categories**: Returns the metadata categories for the site
- **operations**: Returns all possible operations on metadata
- **regions**: Returns all geographical regions usable for spatial queries
- **status**: Returns all possible status values for metadata records

Request example:

```
<request>
    <type>site</type>
    <type>groups</type>
</request>
```

Response

Each type parameter produces an XML subtree in an info container element. An example response to a request for site, categories and groups information would look like the following:

```
<info>
    <site>...</site>
    <categories>...</categories>
    <groups>...</groups>
</info>
```

The structure of each possible subtree is as follows:

Site

- **site**: This is the container for site information
  - **name**: Human readable site name
  - **siteId**: Universal unique identifier (uuid) of the site
  - **platform**: Container for GeoNetwork development version information
    - **name**: Platform name. Always `geonetwork`.
    - **version**: Platform version, given in the X.Y.Z format
    - **subVersion**: Additional version notes, like 'alpha-1' or 'beta-2'.

Example site information:

```
<site>
    <name>My site</name>
    <organisation>FAO</organization>
    <siteId>0619cc50-708b-11da-8202-000d9335906e</siteId>
    <platform>
        <name>geonetwork</name>
        <version>2.2.0</version>
    </platform>
</site>
```

Users

- **users**: This is the container for user information
  - **user [0..n]**: A user of the system
    - **id**: The local identifier of the user
    - **username**: The login name
    - **surname**: The user's surname. Used for display purposes.
    - **name**: The user's name. Used for display purposes.
    - **profile**: User's profile. eg. Administrator, Editor, UserAdmin etc...
    - **address**: The user's address.
    - **state**: The user's state.
    - **zip**: The user's address zip/postal code.
    - **country**: The user's country.
    - **email**: The user's email address.
    - **organisation**: The user's organisation.
    - **kind**: The type of organisation (eg. NGO, Government)

Example response:

```
<users>
    <user>
        <id>3</id>
        <username>eddi</username>
        <surname>Smith</surname>
        <name>John</name>
        <profile>Editor</profile>
        <address/>
        <state/>
        <zip/>
        <country/>
        <email/>
        <organisation/>
        <kind>gov</kind>
    </user>
</users>
```

Groups

- **groups**: This is the container for groups
  - **group [2..n]**: This is a GeoNetwork group. There will always be at least two groups: the Internet and Intranet groups. This element has an id attribute which represents the local identifier for the group.
    - **name**: Group name
    - **description**: Group description
    - **referrer**: The user responsible for this group
    - **email**: The email address to notify when a data file uploaded with the metadata is downloaded
    - **label**: The localised labels used to show the group in the user interface. See *Localised entities*.

Example response:

```
<groups>
    <group id="1">
        <name>editors</name>
        <label>
            <eng>Editors</eng>
            <fre>Éditeurs</fre>
        </label>
    </group>
</groups>
```

Sources

- **sources**: This is the container for sources.
  - **source [0..n]**: A source known to the GeoNetwork node.
    - **name**: Source name
    - **UUID**: Source universal unique identifier

Example response for a source:

```
<sources>
    <source>
        <name>My Host</name>
        <UUID>0619cc50-708b-11da-8202-000d9335906e</uuid>
    </source>
</sources>
```

Schemas

- **schemas**: This is the container for the schema information
  - **schema [0..n]**: A metadata schema.

* **name** - the name of the schema - this is the name by which the schema is known to GeoNetwork. It is also the name of the directory in `GEONETWORK_DATA_DIR/config/schema_plugins` under which the schema can be found.
* **id** - A unique identifier assigned to the schema in the `schema-ident.xml` file.
* **version** - a version string assigned to the schema in the `schema-ident.xml` file.
* **namespaces** - namespaces used by the metadata schema and records that belong to that schema. This is a string suitable for use as a namespace definition in an XML file.
* **edit** - if true then records that use this schema can be edited by GeoNetwork, if false then they can't.
* **conversions** - information about the GeoNetwork services that can be called to convert metadata that use this schema into other XML formats. If there are valid conversions registered for this schema then this element will have a **converter** child for each one of these conversions. Each **converter** child has the following attributes which are intended to be used when searching for a particular format that may be produced by a conversion:
  - **name** - the name of the GeoNetwork service that invokes the converter
  - **nsUri** - the namespace URI of the XML produced by the conversion
  - **schemaLocation** - the schema location (URL) of the namespace URI
  - **xslt** - the name of the XSLT in the plugin schema convert subdirectory that is invoked by the GeoNetwork service to carry out the conversion.

Example response for schemas:

```
<schemas>
 <schema>
   <name>iso19139</name>
   <id>3f95190a-dde4-11df-8626-001c2346de4c</id>
   <version>1.0</version>
   <namespaces>xmlns:gts="http://www.isotc211.org/2005/gts" xmlns:gmx="http://www.isotc2
   <convertDirectory>/usr/local/src/git/geonetwork-2.8.x/web/src/main/webapp/WEB-INF/dat
   <edit>true</edit>
   <conversions>
     <converter name="xml_iso19139" nsUri="http://www.isotc211.org/2005/gmd" schemaLocat
     <converter name="xml_iso19139Tooai_dc" nsUri="http://www.openarchives.org/OAI/2.0/"
   </conversions>
 </schema>
 ...
</schemas>
```

Looking at the example schema (iso19139) above, there are two converters. The first is invoked by calling the GeoNetwork service `xml_iso19139` (eg. `http://somehost/geonetwork/srv/eng/xml_iso19139?uuid=<uuid of metadata>`). It produces an XML format with namespace URI `http://www.isotc211.org/gmd` with schemaLocation `http://www.isotc211.org/2005/gmd/gmd.xsd` and xslt name `xml_iso19139` because the xslt attribute is set to the empty string.

Categories

- **categories**: This is the container for categories.
  - **category [0..n]**: A single GeoNetwork category. This element has an id attribute which represents the local identifier for the category.
    - **name**: Category name
    - **label**: The localised labels used to show the category in the user interface. See *Localised entities*.

Example response:

```
<categories>
    <category id="1">
        <name>datasets</name>
        <label>
            <eng>Datasets</eng>
            <fre>Jeux de données</fre>
        </label>
    </category>
</categories>
```

Operations

- **operations**: This is the container for the operations
  - **operation [0..n]**: This is a possible operation on a metadata record. This element has an id attribute which represents the local identifier for the operation.
    - **name**: Short name for the operation.
    - **reserved**: Can be y or n and is used to distinguish between system reserved and user defined operations.
    - **label**: The localised labels used to show the operation in the user interface. See *Localised entities*.

Example response for operations:

```
<operations>
    <operation id="0">
        <name>view</name>
        <label>
            <eng>View</eng>
            <fre>Voir</fre>
        </label>
    </operation>
</operations>
```

Regions

- **regions**: This is the container for geographical regions
  - **region [0..n]**: This is a region container element. This element has an id attribute which represents the local identifier for the operation.

        * **north**: North coordinate of the bounding box.
        * **south**: South coordinate of the bounding box.
        * **west**: West coordinate of the bounding box.
        * **east**: east coordinate of the bounding box.
        * **label**: The localised labels used to show the region in the user interface. See *Localised entities*.

Example response for regions:

```
<regions>
    <region id="303">
        <north>82.99</north>
        <south>26.92</south>
        <west>-37.32</west>
        <east>39.24</east>
        <label>
            <eng>Western Europe</eng>
        </label>
    </region>
</regions>
```

Status

- **statusvalues**: This is the container for the metadata status value information.
  - **status [0..n]**: A metadata status value. This element has an id attribute which represents the local identifier of the status value.
    * **name**: The status value name
    * **reserved**: Can be y or n and is used to distinguish between system reserved and user defined status values.
    * **label**: The localised labels used to show the status value in the user interface. See *Localised entities*.

Example response for status:

```
<statusvalues>
  <status id="0">
    <name>unknown</name>
    <reserved>y</reserved>
    <label>
      <eng>Unknown</eng>
    </label>
  </status>
  ...
</statusvalues>
```

z3950repositories

- **z3950repositories**: This is the container for the Z3950 repositories that have been configured for this site.

  - **repository [0..n]**: A Z3950 Repository container.
    - **id**: The repository id. This should be used when referring to the repository in GeoNetwork services (eg. xml.harvest.* services - see *Harvesting services*).
    - **label**: The human readable name for the repository.

Example response for z3950repositories:

```
<z3950repositories>
  <repository>
    <id code="act" serverCode="cbb945ec-36ea-11df-9735-ebfc367b61a6">act</id>
    <label>ACT Geographic Data Directory</label>
  </repository>
  .....
</z3950repositories>
```

Localised entities

Localised entities in the responses from this service have a label element which contains localised strings in all supported languages. This element has a child for each supported language. Each child has a name reflecting the language code and content set to the localised text. Example:

```
<label>
    <eng>Editors</eng>
    <fre>Éditeurs</fre>
    <esp>Editores</esp>
</label>
```

6.18.2 Request Forwarding (xml.forward)

This is a request forwarding service. It can be used by JavaScript code to connect to a remote host because a JavaScript program cannot access any machine other than its server (the same origin policy, see http://en.wikipedia.org/wiki/Same_origin_policy). For example, it is used by the harvesting web interface to query a remote host and retrieve the list of site ids.

Request

The details of the request:

```
<request>
    <site>
        <url>...</url>
        <type>...</type>
        <account>
            <username>...</username>
            <password>...</password>
        </account>
    </site>
    <params>...</params>
</request>
```

Where:

- **site**: A container for site information where the request will be forwarded.

- **url**: Refers to the remote URL to connect to. Usually it points to a GeoNetwork XML service but it can point to any XML service.
- **type**: If set to GeoNetwork then use GeoNetwork authentication schema. Any other value, or if the element is missing, refers to a generic node.
- **account**: This element is optional. If present, the provided credentials will be used to authenticate to the remote site.
- **params**: Container for the request parameters.

Request for info from a remote server:

```
<request>
    <site>
        <url>http://mynode.org:8080/geonetwork/srv/en/xml.info</url>
    </site>
    <params>
        <request>
            <type>site<type>
        </request>
    </params>
</request>
```

Note: This service uses the proxy configuration. See `System Configuration` section of the user manual.

Response

Response from the remote service.

6.19 File download services

This section provides a detailed explanation of GeoNetwork file download services. These are the services you would use if you want to download a file attached to a metadata record as 'Data for Download' (usually in onlineResources section of an ISO record) or perhaps as a gmx:FileName (where allowed).

The two services, used together, can be used to create a simple click through licensing scheme for file resources attached to metadata records in GeoNetwork.

6.19.1 xml.file.disclaimer

Retrieves information from the metadata about constraints or restrictions on the resources attached to the metadata record. The information is xml and an xhtml presentation of the constraints and restrictions.

Note: only users that have download rights over the record will be able to use this service. To obtain these rights your application will need to use `xml.user.login` - see *GeoNetwork standard login (xml.user.login)*.

Request

Called with a metadata id or uuid, one or more file names (if more than one file is attached to the metadata record as 'data for download') and access (which is almost always private). Example:

```
<request>
    <uuid>d8c8ca11-ecc8-45dc-b424-171a9e212220</uuid>
    <fname>roam-rsf-aus-bathy-topo-contours.sff</fname>
    <fname>mse09_M8.nc</fname>
    <access>private</access>
</request>
```

Response

The service returns a copy of the request parameters, a copy of the metadata record xml and an HTML version of the license annex generated from the metadata record by the XSL `metadata-license-annex.xsl` (in the `INSTALL_DIR/web/geonetwork/xsl` directory).

Example of an `xml.file.disclaimer` response for a GeoNetwork node (Note: the <metadata> and <license> elements are not shown in full as they are too big):

```
<response>
    <id>22</id>
    <uuid>d8c8ca11-ecc8-45dc-b424-171a9e212220</uuid>
    <fname>roam-rsf-aus-bathy-topo-contours.sff</fname>
    <fname>mse09_M8.nc</fname>
    <access>private</access>
    <metadata>
        <gmd:MD_Metadata xmlns:gmd="http://www.isotc211.org/2005/gmd" xmlns:xsi="http://
            <!--.........-->
        </gmd:MD_Metadata>
    </metadata>
    <license>
        <html>
            <head>
                <link href="http://localhost:8080/geonetwork/favicon.ico" rel="shortcut
                <link href="http://localhost:8080/geonetwork/favicon.ico" rel="icon" typ
                <link rel="stylesheet" type="text/css" href="http://localhost:8080/geone
                <link rel="stylesheet" type="text/css" href="http://localhost:8080/geone
            </head>
            <body>
                <!--.........-->
            </body>
        </html>
    </license>
</response>
```

The idea behind this service is that you will receive an HTML presentation of the constraints/restrictions on the resource that you can show to a user for an accept/decline response.

The HTML presentation is controlled by the server so together with the `xml.file.download` service, this is the way that GeoNetwork can be used to provide a simple click-through licensing system for file resources attached to metadata records.

To signify acceptance of the license and download the resources you should use the `xml.file.download` service.

If an exception occurred then the service returns HTTP status code 500 and an XML document describing what went wrong. An example of such a response is:

```
<error id="metadata-not-found">
  <message>Metadata not found</message>
  <class>MetadataNotFoundEx</class>
  .......
</error>
```

See *Exception handling* for more details.

Errors

- **IllegalArgumentException**: Request must contain a UUID or an ID parameter.
- **MetadataNotFoundException**: Metadata not found.
- **OperationNowAllowedException**: you don't have download permission over this record.

6.19.2 xml.file.download

After your application has received any license conditions that go with the file resources attached to the metadata record from `xml.file.disclaimer`, you can use this service to download the resources.

Note: only users that have download rights over the record will be able to use this service. To obtain these rights your application will need to use `xml.user.login` - see *GeoNetwork standard login (xml.user.login)*.

Note: this service is protected against users and/or applications that do not go through the `xml.file.disclaimer` service first.

Request

Called with a metadata id or uuid, one or more file names (if more than one file is attached to the metadata record as 'data for download'), access (which is almost always private) and details of the user who has accepted the license and wants to download the files. Example:

```
<request>
    <uuid>d8c8ca11-ecc8-45dc-b424-171a9e212220</uuid>
    <fname>roam-rsf-aus-bathy-topo-contours.sff</fname>
    <fname>mse09_M8.nc</fname>
    <access>private</access>
    <name>Aloyisus Wankania</name>
    <org>Allens Butter Factory</org>
    <email>A.Wankania@allens.org</email>
    <comments>Gimme the data buddy</comments>
</request>
```

Response

The service returns HTTP status code 200 along with a zip archive containing:

- the file resources requested in the **fname** parameter(s)
- a copy of the metadata record (as a mef) - called `metadata.zip`
- a copy of the html license generated (as provided by the `xml.file.disclaimer` service) - called `license-annex.html`

If an exception occurs or the `xml.file.disclaimer` service has not been executed by the same user, then a zero-length file will be returned. Unlike other GeoNetwork services, no other indication of an exception is given.

6.20 Harvesting services

This section describes the services used to create, update and manage GeoNetwork harvesters. These services allow complete control over harvester behaviour. Authentication is required for all services described in this section. In addition, these services can only be run by users with the **Administrator** profile.

6.20.1 Get harvester definitions (xml.harvesting.get)

Retrieves information about one or all configured harvesters.

Request

Called without parameters, this service returns all harvesters. Example:

```
<request/>
```

Otherwise, an **id** parameter can be specified to request the definition of a specific harvester instance:

```
<request>
    <id>123</id>
</request>
```

Response

When called without parameters the service returns HTTP status code 200 along with an XML document with all harvester instances. The XML document has a root element called `nodes` with a `node` child for each harvester.

Example of an xml.harvesting.get response for a GeoNetwork harvester:

```
<nodes>
    <node id="125" type="geonetwork">
        <site>
            <name>test 1</name>
            <uuid>0619cc50-708b-11da-8202-000d9335aaae</uuid>
            <account>
                <use>false</use>
```

```
                <username />
                <password />
            </account>
            <host>http://www.fao.org/geonetwork</host>
            <createRemoteCategory>true</createRemoteCategory>
            <mefFormatFull>true</mefFormatFull>
            <xslfilter/>
        </site>
        <content>
            <validate>true</validate>
            <importxslt>none</importxslt>
        </content>
        <options>
            <every>0 0 0/3 ? * *</every>
            <oneRunOnly>false</oneRunOnly>
            <status>inactive</status>
        </options>
        <searches>
            <search>
                <freeText />
                <title />
                <abstract />
                <keywords />
                <digital>false</digital>
                <hardcopy>false</hardcopy>
                <source>
                    <UUID>0619cc50-708b-11da-8202-000d9335906e</uuid>
                    <name>Food and Agriculture organisation</name>
                </source>
            </search>
        </searches>
        <groupsCopyPolicy>
            <group name="all" policy="copy"/>
            <group name="mygroup" policy="createAndCopy"/>
        </groupsCopyPolicy>
        <categories>
            <category id="4"/>
        </categories>
        <info>
            <lastRun />
            <running>false</running>
        </info>
    </node>
</nodes>
```

If you specify a harvester **id** parameter in the request, then the XML document returned has a `node` root element that describes the harvester.

Example of an xml.harvesting.get response for a WebDAV harvester:

```
<node id="165" type="webdav">
    <site>
        <name>test 1</name>
        <UUID>0619cc50-708b-11da-8202-000d9335aaae</uuid>
        <url>http://www.mynode.org/metadata</url>
        <icon>default.gif</icon>
        <account>
            <use>true</use>
```

```
            <username>admin</username>
            <password>admin</password>
        </account>
    </site>
    <options>
        <every>0 0 0/3 ? * *</every>
        <oneRunOnly>false</oneRunOnly>
        <recurse>false</recurse>
        <validate>true</validate>
        <status>inactive</status>
    </options>
    <privileges>
        <group id="0">
            <operation name="view" />
        </group>
        <group id="14">
            <operation name="download" />
        </group>
    </privileges>
    <categories>
        <category id="2"/>
    </categories>
    <info>
        <lastRun />
        <running>false</running>
    </info>
</node>
```

Each harvester has some common XML elements, plus additional elements that are specific to each harvesting type.

The common XML elements are described at *Harvesting nodes*.

If an error occurred then HTTP status code 500 is returned along with an XML document which contains details of what went wrong. An example of such an error response is:

```
<error id="object-not-found">
  <message>Object not found</message>
  <class>ObjectNotFoundEx</class>
  .....
</error>
```

See *Exception handling* for more details.

Errors

- **ObjectNotFoundEx** If a harvester definition with the specified **id** cannot be found.

6.20.2 Create harvester instance (xml.harvesting.add)

Create a new harvester. The harvester can be of any type supported by GeoNetwork (see *Harvesting nodes* for a list). When a new harvester instance is created, its status is set to inactive. A call to the `xml.harvesting.start` service is required to set the status to active and run the harvester at the scheduled time.

Request

The service requires an XML tree with all information about the harvesting node to be added. The common XML elements that must be in the tree are described at *Harvesting nodes*. Settings and example requests for each type of harvester in GeoNetwork are as follows:

- *Harvesting node geonetwork*
- *Harvesting node webdav*
- *Harvesting node csw*
- *Harvesting node z3950*
- *Harvesting node oaipmh*
- *Harvesting node thredds*
- *Harvesting node wfsfeatures*
- *Harvesting node filesystem*
- *Harvesting node arcsde*
- *Harvesting node ogcwxs*
- *Harvesting node geoPREST*

Summary of features of the supported harvesting types

Harvesting type	Authentication	Privileges	Categories
GeoNetwork	native	through policies	yes
WebDAV	HTTP digest	yes	yes
CSW	HTTP Basic	yes	yes

Response

If the request succeeds and the harvester instance is created, then HTTP status code 200 is returned along with an XML document containing the definition of the harvester as is described in the response section of the `xml.harvesting.get` service above.

If an error occurred then HTTP status code 500 is returned along with an XML document which contains details of what went wrong. An example of such an error response is:

```
<error id="object-not-found">
  <message>Object not found</message>
  <class>ObjectNotFoundEx</class>
  .....
</error>
```

See *Exception handling* for more details.

6.20.3 Get information for Harvester definition (xml.harvesting.info)

This service can be used to obtain information from the server that is relevant to defining a harvester eg. harvester icons, stylesheets etc.

Request and Response

All requests must have a **type** parameter which defines the type of information required. The requests and responses for each value of the **type** parameter are:

icons

Return the list of icons that can be used when creating a harvester instance. Icons are usually set in **site/icon** harvester setting.

POST Request Example:

```
<request>
  <type>icons<type>
</request>

URL:
http://localhost:8080/geonetwork/srv/eng/xml.harvesting.info
```

Response Example:

```
<root>
  <icons>
    <icon>wfp.gif</icon>
    <icon>unep.gif</icon>
    <icon>webdav.gif</icon>
    <icon>gn20.gif</icon>
    <icon>thredds.gif</icon>
    <icon>wfs.gif</icon>
    <icon>csw.gif</icon>
    <icon>filesystem.gif</icon>
    <icon>fao.gif</icon>
    <icon>default.gif</icon>
    <icon>Z3950.gif</icon>
    <icon>oai-mhp.gif</icon>
    <icon>esri.gif</icon>
  </icons>
</root>
```

importStylesheets

Return the list of stylesheets that can be used when creating a harvester instance. The `id` element in the response can be used in the **content/importxslt** harvester setting for those harvesters that support it.

POST Request Example:

```
<request>
  <type>icons<type>
</request>

URL:
http://localhost:8080/geonetwork/srv/eng/xml.harvesting.info
```

Response Example:

```
<root>
  <stylesheets>
    <record>
      <id>ArcCatalog8_to_ISO19115.xsl</id>
      <name>ArcCatalog8_to_ISO19115</name>
    </record>
    <record>
      <id>CDMCoords-to-ISO19139Keywords.xsl</id>
      <name>CDMCoords-to-ISO19139Keywords</name>
    </record>
    .....
  </stylesheets>
</root>
```

oaiPmhServer

Request information about the sets and prefixes of an OAIPMH server. This request requires an additional url attribute on the type parameter specifying the name of the OAIPMH server to query.

POST Request Example:

```
<request>
  <type url="http://localhost:8080/geonetwork/srv/eng/oaipmh">oaiPmhServer</type>
</request>

URL:
http://localhost:8080/geonetwork/srv/eng/xml.harvesting.info
```

Response Example:

```
<root>
 <oaiPmhServer>
   <formats>
     <format>iso19115</format>
     <format>fgdc-std</format>
     <format>iso19139</format>
     <format>csw-record</format>
     <format>iso19110</format>
     <format>dublin-core</format>
     <format>oai_dc</format>
   </formats>
   <sets>
     <set>
       <name>maps</name>
       <label>Maps &amp;amp; graphics</label>
     </set>
     <set>
       <name>datasets</name>
       <label>Datasets</label>
     </set>
     ......
   </sets>
 </oaiPmhServer>
</root>
```

wfsFragmentSchemas

Return list of schemas that have WFS Fragment conversion stylesheets. These stylesheets are stored in the `WFSToFragments` directory in the `convert` directory of a metadata schema. eg. for schema iso19139 this directory would be `GEONETWORK_DATA_DIR/config/schema_plugins/iso19139/convert/WFSToFragments`.

POST Request Example:

```
<request>
 <type>wfsFragmentSchemas</type>
</request>

URL:
http://localhost:8080/geonetwork/srv/eng/xml.harvesting.info
```

Response Example:

```
<root>
 <schemas>
   <record>
     <id>iso19139</id>
     <name>iso19139</name>
   </record>
 </schemas>
</root>
```

wfsFragmentStylesheets

Return WFS Fragment conversion stylesheets for a schema previously returned by the request type `wfsFragmentSchemas` described above. These stylesheets are stored in the `WFSToFragments` directory in the `convert` directory of a metadata schema. eg. for schema iso19139 this directory would be `GEONETWORK_DATA_DIR/config/schema_plugins/iso19139/convert/WFSToFragments`.

POST Request Example:

```
<request>
  <schema>iso19139</schema>
  <type>wfsFragmentStylesheets</type>
</request>

URL:
http://localhost:8080/geonetwork/srv/eng/xml.harvesting.info
```

Response Example:

```
<root>
 <stylesheets>
   <record>
     <id>deegree22_philosopher_fragments.xsl</id>
     <name>deegree22_philosopher_fragments</name>
     <schema>iso19139</schema>
   </record>
   <record>
     <id>geoserver_boundary_fragments.xsl</id>
```

```
      <name>geoserver_boundary_fragments</name>
      <schema>iso19139</schema>
    </record>
 </stylesheets>
</root>
```

threddsFragmentSchemas

Return list of schemas that have THREDDS Fragment conversion stylesheets. These stylesheets are stored in the `ThreddsToFragments` directory in the `convert` directory of a metadata schema. eg. for schema iso19139 this directory would be `GEONETWORK_DATA_DIR/config/schema_plugins/iso19139/convert/ThreddsToFragments`.

POST Request Example:

```
<request>
 <type>threddsFragmentSchemas</type>
</request>
```

```
URL:
http://localhost:8080/geonetwork/srv/eng/xml.harvesting.info
```

Response Example:

```
<root>
 <schemas>
   <record>
     <id>iso19139</id>
     <name>iso19139</name>
   </record>
 </schemas>
</root>
```

threddsFragmentStylesheets

Return WFS Fragment conversion stylesheets for a schema previously returned by the request type `threddsFragmentSchemas` described above. These stylesheets are stored in the `ThreddsToFragments` directory in the `convert` directory of a metadata schema. eg. for schema iso19139 this directory would be `GEONETWORK_DATA_DIR/config/schema_plugins/iso19139/convert/ThreddsToFragments`.

POST Request Example:

```
<request>
  <schema>iso19139</schema>
  <type>threddsFragmentStylesheets</type>
</request>
```

```
URL:
http://localhost:8080/geonetwork/srv/eng/xml.harvesting.info
```

Response Example:

```
<root>
 <stylesheets>
   <record>
     <id>netcdf-attributes.xsl</id>
     <name>netcdf-attributes</name>
     <schema>iso19139</schema>
   </record>
   <record>
     <id>thredds-metadata.xsl</id>
     <name>thredds-metadata</name>
     <schema>iso19139</schema>
   </record>
 </stylesheets>
</root>
```

ogcwxsOutputSchemas

Return list of schemas that have GetCapabilities conversion stylesheets for a particular three letter OGC service type code. These stylesheets are stored in the `OGCWxSGetCapabilitiesto19119` directory in the `convert` directory of a metadata schema. eg. for schema iso19139:

- the directory for these stylesheets would be `GEONETWORK_DATA_DIR/config/schema_plugins/iso19`
- if a conversion from the GetCapabilities statement of a particular OGC service to a metadata record of this schema exists, then a stylesheet for that serviceType will be present in the directory eg. for schema iso19139 and serviceType `WFS`, the conversion stylesheet name would be `OGCWFSGetCapabilities-to-ISO19119_ISO19139.xsl`

POST Request Example:

```
<request>
 <type>ogcwxsOutputSchemas</type>
 <serviceType>WFS</serviceType>
</request>

URL:
http://localhost:8080/geonetwork/srv/eng/xml.harvesting.info
```

Response Example:

```
<root>
 <schemas>
   <record>
     <id>iso19139</id>
     <name>iso19139</name>
   </record>
 </schemas>
</root>
```

Errors

If an error occurred then HTTP status code 500 is returned along with an XML document which contains details of what went wrong. An example of such an error response is:

```
<error id="bad-parameter">
  <message>type</message>
  <class>BadParameterEx</class>
  .....
</error>
```

See *Exception handling* for more details.

6.20.4 Update a Harvester Instance (xml.harvesting.update)

This service can be used to change the parameters of a harvester instance.

Note: You cannot change the harvester type.

Request

The simplest way to use this service is to:

1. use the `xml.harvesting.get` service to obtain the XML definition of the harvester that you want to update.
2. modify the parameters as required.
3. call this service with the modified XML definition of the harvester as the request.

The XML request is the same as that used in `xml.harvesting.add`.

Response

If the update succeeded then HTTP status code 200 is returned along with an XML document containing the harvester definition as supplied in the request.

If an error occurred then HTTP status code 500 is returned along with an XML document which contains details of what went wrong. An example of such an error response is:

```
<error id="object-not-found">
  <message>Object not found</message>
  <class>ObjectNotFoundEx</class>
  .....
</error>
```

See *Exception handling* for more details.

6.20.5 Control or Remove a Harvester Instance (xml.harvesting.remove, xml.harvesting.start, xml.harvesting.stop, xml.harvesting.run)

These services are described in on section because they share a common request interface. Their purpose is to remove, start, stop or run a harvester:

1. **remove**: Remove a harvester. Deletes the harvester instance.
2. **start**: When created, a harvester is in the inactive state. This operation makes it active which means it will be run at the enxt scheduled time.

3. **stop**: Makes a harvester inactive - it will no longer be executed at the scheduled time. Note this will *not* stop a harvester that is already performing a harvest.
4. **run**: Start the harvester now. Used to test the harvesting.

Request

A set of ids to operate on. Example:

```
<request>
    <id>123</id>
    <id>456</id>
    <id>789</id>
</request>
```

Response

Similar to the request but every id has a status attribute indicating the success or failure of the operation. For example, the response to the previous request could be:

```
<response>
    <id status="ok">123</id>
    <id status="not-found">456</id>
    <id status="inactive">789</id>
</response>
```

The table below summarises, for each service, the possible status values.

Status value	remove	start	stop	run
ok	✓	✓	✓	✓
not-found	✓	✓	✓	✓
inactive				✓
already-inactive			✓	
already-active		✓		
already-running				✓

If the request has no id parameters, an empty response is returned.

Most errors relating to a harvester specified in the request (eg. harvester id not found) are returned as status attributes in the response. However, exceptions can still occur, in which case HTTP status code 500 is returned along with an XML document which contains details of what went wrong. An example of such an error response is:

```
<error id="service-not-allowed">
  <message>Service not allowed</message>
  <class>ServiceNotAllowedEx</class>
  .....
</error>
```

See *Exception handling* for more details.

6.20.6 Retrieve Harvesting History (xml.harvesting.history)

This service can be used to retrive the history of harvest sessions for a specified harvester instance or all harvester instances. The harvester history information is stored in the GeoNetwork database in the HarvestHistory table.

Request

Called without an **id** parameter, this service returns the harvest history of all harvesters. The response can be sorted by harvest *date* or by harvester *type*. The sort order is specified in the parameter **sort**. Example:

```
<request>
  <sort>date</sort>
</request>
```

Otherwise, an **id** parameter can be specified to request the harvest history of a specific harvester instance. In this case the sort order is by *date* of harvest:

```
<request>
  <id>123</id>
</request>
```

Response

If the update succeeded then HTTP status code 200 is returned along with an XML document containing the harvest history. The response for both types of requests is the same except that the response to a request for the history of a specific harvester will only have history details for that harvester. An example of the response is:

```
<response>
 <response>
   <record>
     <id>1</id>
     <harvestdate>2013-01-01T19:24:54</harvestdate>
     <harvesteruuid>b6a11fc3-3f6f-494b-a8f3-35eaadced575</harvesteruuid>
     <harvestername>test plaja</harvestername>
     <harvestertype>geonetwork</harvestertype>
     <deleted>n</deleted>
     <info>
       <result>
         <total>5</total>
         <added>5</added>
         <updated>0</updated>
         <unchanged>0</unchanged>
         <unknownSchema>0</unknownSchema>
         <removed>0</removed>
         <unretrievable>0</unretrievable>
         <doesNotValidate>0</doesNotValidate>
       </result>
     </info>
     <params>
       .....
     </params>
   </record>
```

```
 </response>
 <nodes>
   <node id="955" type="geonetwork">
     .....
   </node>
   .....
 </nodes>
 <sort>date</sort>
</response>
```

Each **record** element in the embedded **response** element contains the details of a harvest session. The elements are:

- **id** - harvest history record id in harvesthistory table
- **harvestdate** - date of harvest
- **harvesteruuid** - uuid of harvester that ran
- **harvestername** - name of harvester (Site/Name parameter) that ran
- **harvestertype** - type of harvester that ran
- **deleted** - has the harvester that ran been deleted? 'y' - yes, 'n' - no
- **info** - results of the harvest. May contain one of the following elements:
- **result** - details of the successful harvest (a harvester dependent list of results from the harvest)
- **error** - an exception from an unsuccessful harvest - see *Exception handling* for content details of this element
- **params** - the parameters that the harvester had been configured with for the harvest

After the embedded **response** element, the currently configured harvesters are returned as **node** children of a **nodes** element - see *Create harvester instance (xml.harvesting.add)* for references to each of the harvester types that can be returned here.

If an error occurred then HTTP status code 500 is returned along with an XML document which contains details of what went wrong. An example of such an error response is:

```
<error id="object-not-found">
  <message>Object not found</message>
  <class>ObjectNotFoundEx</class>
  .....
</error>
```

See *Exception handling* for more details.

6.20.7 Delete Harvesting History Entries (xml.harvesting.history.delete)

This service can be used to delete harvester history entries from the harvesthistory table in the GeoNetwork database.

Request

One or more **id** parameters can be specified to request deletion of the harvest history entries in the harvesthistory table. The **id** element values can be obtained from *Retrieve Harvesting History*

(xml.harvesting.history):

```
<request>
  <id>1</id>
  <id>2</id>
</request>
```

Response

If successful then HTTP status code 200 is returned along with an XML document with details of how many harvest history records were successfully deleted. An example of this response is:

```
<response>2</response>
```

Note: If records with the id specified in the parameters are not present, they will be quietly ignored.

If an error occurred then HTTP status code 500 is returned along with an XML document which contains details of what went wrong. An example of such an error response is:

```
<error id="service-not-allowed">
  <message>Service not allowed</message>
  <class>ServiceNotAllowedEx</class>
  .....
</error>
```

See *Exception handling* for more details.

6.21 Schema Services

Metadata schemas can be plugged into GeoNetwork - see *Schema Plugins*. Any application that needs to:

- find information (eg. names, versions, namespaces, converters) about metadata schemas that are plugged into GeoNetwork should use the *Site Information (xml.info)* service
- find information about the schema elements and codelists should use the `xml.schema.info` service described in this section of the manual
- add, delete, update schema plugins in GeoNetwork should use the `xml.metadata.schema.add`, `xml.metadata.schema.delete` and `xml.metadata.schema.update` services described in this section of the manual

6.21.1 Metadata Schema Information (xml.info?type=schemas)

See *Site Information (xml.info)* for more details.

6.21.2 Schema Element and Codelist Information (xml.schema.info)

This service returns information about a set of schema elements or codelists. The returned information consists of a localised label, a description, conditions that the element must satisfy etc.

Request

Requests to this service can only be made using the HTTP POST binding with application/XML content type. Requests can ask for information on multiple elements and codelists from different schemas. Description of the request fields is as follows:

- **element**: Must contain a **schema** and a **name** attribute. The schema attribute must be the name of a schema currently registered in GeoNetwork (see *Site Information (xml.info)* for more details). The name attribute must be the qualified name of the metadata schema element on which information is required. Other optional attributes can be specified to help determine the appropriate context for the metadata element. These optional attributes are:
  - **context**: The qualified name of the metadata schema element that is the parent of the element specified in the **name** attribute.
  - **fullContext**: The qualified xpath of the element specified in the **name** attribute.
  - **isoType**: For profiles of ISO19115/19139 only. The qualified name of the element in the base ISO19115/19139 schema that the element specified in the **name** attribute is a substitute for. eg. in the Marine Community Profile of ISO19115/19139, `mcp:MD_Metadata` is a substitute for `gmd:MD_Metadata` in the base ISO19115/19139 metadata schema.
- **codelist**: Accepts a **schema** and **name** attribute as for **element** but information on any codelist associated with the qualified name of the metadata schema element in the **name** attribute will be returned instead.

```
<request>
    <element schema="iso19139" name="gmd:constraintLanguage" />
    <codelist schema="iso19115" name="DateTypCd" />
</request>
```

Note: The text returned is localised into the language specified in the the service call. eg. A call to /geonetwork/srv/eng/xml.schema.info will return text in the English (eng) language.

Response

If the request executed successfully then HTTP status code 200 will be returned along with an XML document containing the response. The root field of the response will be populated with information about the element/codelist specified in the request. The fields of the response are:

- **element**: A container for information about a schema element. It has a name attribute which contains the qualified name of the element.
  - **label**: The human readable name of the element, localised into the language specified in the request.
  - **description**: A generic description of the element.
  - **condition [0..1]**: This field is optional and indicates if the element must satisfy a condition eg. mandatory. The condition text is displayed with the element name so it intended to be human readable.
- **codelist**: A container for information about a codelist. It has a name attribute which contains the qualified name of the codelist.
  - **entry [1..n]**: A container for a codelist entry. There can be more than one entry.

        * **code**: The entry code. This is the value that will be used in the metadata.
        * **label**: This is a human readable name for the code, used to show the code in the user interface. It is localised.
        * **description**: A localised description/meaning of the code. The description is shown in the user interface with the label.

```
<response>
    <element name="gmd:constraintLanguage">
        <label>Constraint language</label>
        <description>language used in Application Schema</description>
        <condition>mandatory</condition>
    </element>
    <codelist name="DateTypCd">
        <entry>
            <code>creation</code>
            <label>Creation</label>
            <description>date when the resource was brought into existence</description>
        </entry>
        <entry>
            <code>publication</code>
            <label>Publication</label>
            <description>date when the resource was issued</description>
        </entry>
        <entry>
            <code>revision</code>
            <label>Revision</label>
            <description>date identifies when the resource was examined
            or re-examined and improved or amended</description>
        </entry>
    </codelist>
</response>
```

Error management

If an exception occurs during the processing of the request, then an HTTP 500 status code is returned along with an XML document describing the exception. See *Exception handling* for more details.

Apart from exceptions, the service can encounter errors trying to retrieve an element/codelist information eg. if the requested element is not present. If such an error is encountered, then the object is copied from the response and an error attribute is added describing the error. An example of such a response is:

```
<response>
    <element schema="iso19139" name="blablabla" error="not-found"/>
</response>
```

Possible errors returned by xml.schema.info service:

Error code	Description
unknown-schema	The specified schema is not supported
unknown-namespace	The namespace of the specified prefix was not found
not-found	The requested element / codelist was not found

6.21.3 Add a metadata schema (xml.metadata.schema.add)

The **xml.metadata.schema.add** service can be used to add a metadata schema to GeoNetwork. The details of what the schema should contain are covered in the *Schema Plugins* section of this manual.

Only **Administrator** users can run this service.

Requires authentification: Yes

Request

Parameters:

- **schema**: (mandatory) Name of the schema to add.

One of the following parameters:

- **fname**: Server file name (full path) to metadata schema zip archive.
- **url**: Http URL of metadata schema zip archive.
- **uuid**: Uuid of metadata record in current catalog that has a metadata schema zip archive uploaded and stored with it.

Schema add request example:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.metadata.schema.add

Mime-type:
application/xml

Post request:
<request>
  <schema>iso19139.mcp</schema>
  <fname>/usr/local/src/git/schemaPlugins-2.8.x/dist/iso19139.mcp.zip</fname>
</request>
```

Response

If the request executed succesfully then an HTTP 200 status code is returned and an XML document confirming success is returned. An example response is::

```
<response status="ok" message="Schema iso19139.mcp has been added/updated"/>
```

If the request fails then an HTTP 500 status code error is returned and the response contains an XML document with the details of the exception/what went wrong. An example error response is::

```
<error id="operation-aborted">
  <message>Schema already exists</message>
  <class>OperationAbortedEx</class>
  .....
</error>
```

See *Exception handling* for more details.

6.21.4 Update a metadata schema (xml.metadata.schema.update)

The **xml.metadata.schema.update** service can be used to update a metadata schema in GeoNetwork. The details of what the schema should contain are covered in the *Schema Plugins* section of this manual.

Only **Administrator** users can run this service.

Requires authentification: Yes

Request

Parameters:

- **schema**: (mandatory) Name of the schema to update. Must be the name of a currently registered metadata schema in GeoNetwork.

One of the following parameters:

- **fname**: Server file name (full path) to metadata schema zip archive.
- **url**: Http URL of metadata schema zip archive.
- **uuid**: Uuid of metadata record in current catalog that has a metadata schema zip archive uploaded and stored with it.

Schema update request example:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.metadata.schema.update

Mime-type:
application/xml

Post request:
<request>
  <schema>iso19139.mcp</schema>
  <fname>/usr/local/src/git/schemaPlugins-2.8.x/dist/iso19139.mcp.zip</fname>
</request>
```

Response

If the request executed succesfully then an HTTP 200 status code is returned and an XML document confirming success is returned. An example response is::

```
<response status="ok" message="Schema iso19139.mcp has been added/updated"/>
```

If the request fails then an HTTP 500 status code error is returned and the response contains an XML document with the details of the exception/what went wrong. An example error response is::

```
<error id="operation-aborted">
  <message>Schema doesn't exist</message>
  <class>OperationAbortedEx</class>
  .....
</error>
```

See *Exception handling* for more details.

6.21.5 Delete a metadata schema (xml.metadata.schema.delete)

The **xml.metadata.schema.delete** service can be used to delete a metadata schema in GeoNetwork. A metadata schema can only be deleted if:

- there are no metadata records in the catalog that use it
- no other metadata schema is dependent on it

Only **Administrator** users can run this service.

Requires authentification: Yes

Request

Parameters:

- **schema**: (mandatory) Name of the schema to delete. Must be the name of a currently registered metadata schema in GeoNetwork.

Schema delete request example:

```
Url:
http://localhost:8080/geonetwork/srv/eng/xml.metadata.schema.delete

Mime-type:
application/xml

Post request:
<request>
  <schema>iso19139.mcp</schema>
</request>
```

Response

If the request executed without an exception then an HTTP 200 status code is returned and an XML document giving status is returned. An example response is::

```
<response status="ok" message="Schema iso19139.mcp has been deleted"/>
```

Other responses may describe errors, in which case the status is set to "error". An example error response is::

```
<response status="error" message="Cannot remove schema iso19139 because there are record
```

If the request fails due to an exception in the service then an HTTP 500 status code error is returned and the response contains an XML document with the details of the exception/what went wrong. An example error response is::

```
<error id="operation-aborted">
  <message>Schema doesn't exist</message>
  <class>OperationAbortedEx</class>
  .....
</error>
```

See *Exception handling* for more details.

6.22 MEF services

This section describes the services related to the Metadata Exchange Format. These services allow import/export metadata using the MEF (Metadata Exchange Format) format.

Note: before using these services please review the section on the MEF format at *Metadata Exchange Format*

6.22.1 mef.export

This service exports GeoNetwork metadata using the MEF file format. The metadata record can be specified using a uuid or the currently selected set of metadata records can be used - see *Select metadata records (xml.metadata.select)* for more details on how to select a set of metadata records.

This service is public but metadata access rules apply. For a partial export, the view privilege is enough but for a full export the download privilege is also required. Without a login step, only partial exports on public metadata are allowed.

This service uses the system temporary directory to build the MEF file. By default the tmp directory is `INSTALL_DIR/web/geonetwork/data`. You will need to ensure that sufficient disk space is available on that filesystem for full exports (ie. those that include data files uploaded with the metadata record) to be successful. Alternatively, you can specify a different directory by configuring the `uploadDir` parameter in the `general` section of the `INSTALL_DIR/web/geonetwork/WEB-INF/config.xml` file.

Request

This service accepts requests in GET/POST and XML form. The input parameters are all optional and are as follows:

- **uuid** the universal unique identifier of the metadata to be exported. If this parameter is optional then the selected set of metadata will be exported. To select a set of metadata see *Select metadata records (xml.metadata.select)*.
- **format** which MEF format to use. Can be one of: *simple*, *partial*, *full*. Default is *full* - which means thumbnails and data files uploaded with the metadata record are included in the MEF.
- **skipUuid** (*true*|*false*) If set to *true*, the metadata record UUIDs will not be exported into the MEF `info.xml` file. Without a UUID (which is a unique key inside the database) the metadata records in a MEF can be repeatedly imported as they will receive a new UUID on import. The default value is *false*.
- **version** (*true*|*false*) If set to *true*, MEF Version 2.0 is used, otherwise MEF Version 1.0 is used. This parameter needs to be present if related records are being included in the MEF - see the **relation** parameter below.
- **resolveXlink** (*true*|*false*) If set to *true*, then any XLinks in the metadata records are resolved before the records are exported. If *false*, the metadata records are exported with unresolved XLinks. Default is *true*.
- **removeXlinkAttributes** (*true*|*false*) If set to *true*, then any XLink attributes in the metadata records are removed before the records are exported. Default is *true*.

- **relation** (*true|false*) If set to *true and* **version** is also set to *true*, then related records (eg.parent, feature catalog) are exported into the MEF with the metadata record. Default is *true*.

Example POST request:

```
<request>
 <uuid>f80bca2e-ff75-4107-8999-4c1864cb1b1b</uuid>
 <format>full</format>
 <skipUuid>true</skipUuid>
 <version>true</version>
 <relation>true</relation>
</request>

URL:
http://localhost:8080/geonetwork/srv/eng/mef.export
```

Response

If successful the service returns HTTP status code 200 and the response is a MEF file with name as follows:

- prefix is `export-`
- MEF format: *simple*, *partial*, *full*
- current time in milliseconds since 1970
- `zip` extension

eg. `export-full-1357949862822.zip`

If an exception occurred then service returns HTTP status code 500 and an XML document describing what went wrong. An example of such a response is:

```
<error id="metadata-not-found">
  <message>Metadata not found</message>
  <class>MetadataNotFoundEx</class>
  .......
</error>
```

See *Exception handling* for more details.

6.22.2 mef.import

This service is used to import a metadata record in the MEF format.

Only users with the **Administrator** profile can run this service.

Request

The service accepts a multipart/form-data POST request with a single **mefFile** parameter that must contain the MEF information.

Response

If the import is successful, the service returns HTTP status code 200 along with an XML document containing the GeoNetwork integer id of the created metadata. Example:

```
<ok>123</ok>
```

If an exception occurred then service returns HTTP status code 500 and an XML document describing what went wrong. An example of such a response is:

```
<error id="metadata-not-found">
  <message>Metadata not found</message>
  <class>MetadataNotFoundEx</class>
  .......
</error>
```

See *Exception handling* for more details.

Notes

- Version 1.0 of the MEF format does not capture the metadata owner (the creator) and the group owner. During import, the user that is performing this operation will become the metadata owner and the group owner will be set to null.

6.23 CSW service

GeoNetwork opensource catalog publishes metadata using the OGC CSW (Catalog Services for the Web) protocol supporting HTTP binding to invoke the operations.

The protocol operations are described in the document **OpenGIS® Catalogue Services Specification**:

http://portal.opengeospatial.org/files/?artifact_id=20555

GeoNetwork is compliant with the 2.0.2 version of the specification, supporting the following CSW operations:

- *GetCapabilities*
- *DescribeRecord*
- *GetRecordById*
- *GetRecords*
- *Harvest*
- *Transaction*

This chapter briefly describes the different operations supported in GeoNetwork and gives some usage examples. To get a complete reference of the operations and parameters of each CSW operation refer to the document **OpenGIS® Catalogue Services Specification**.

The invocation of the operations from a Java client is analogous as described in the chapter for XML services.

6.23.1 CSW operations

The CSW operations are divided in 2 types: Discovery and Publication. The Discovery operations are used to query the server about its capacities and to search and retrieve metadata from it. The Publication opertions (Harvest and Transaction) are used to insert metadata into the catalog.

The CSW operations can be accesed using POST, GET methods and SOAP encoding.

The GeoNetwork opensource catalog CSW Discovery service operations are accessible through the url:

http://localhost:8080/geonetwork/srv/en/csw

GetCapabilities

GetCapabilities operation allows CSW clients to retrieve service metadata from a server. The response to a **GetCapabilities** request is an XML document containing service metadata about the server.

Request examples

GET request:

```
http://localhost:8080/geonetwork/srv/en/csw?request=GetCapabilities&service=CSW&acceptVe
```

POST request:

```
Url:
http://localhost:8080/geonetwork/srv/en/csw

Content-type:
application/xml

Post data:
<?xml version="1.0" encoding="UTF-8"?>
<csw:GetCapabilities xmlns:csw="http://www.opengis.net/cat/csw/2.0.2" service="CSW">
<ows:AcceptVersions xmlns:ows="http://www.opengis.net/ows">
<ows:Version>2.0.2</ows:Version>
</ows:AcceptVersions>
<ows:AcceptFormats xmlns:ows="http://www.opengis.net/ows">
<ows:OutputFormat>application/xml</ows:OutputFormat>
</ows:AcceptFormats>
</csw:GetCapabilities>
```

SOAP request:

```
Url:
http://localhost:8080/geonetwork/srv/en/csw

Content-type:
application/soap+xml

Post data:
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
<env:Body>
<csw:GetCapabilities xmlns:csw="http://www.opengis.net/cat/csw/2.0.2"
service="CSW">
```

```
<ows:AcceptVersions xmlns:ows="http://www.opengis.net/ows">
<ows:Version>2.0.2</ows:Version>
</ows:AcceptVersions>
<ows:AcceptFormats xmlns:ows="http://www.opengis.net/ows">
<ows:OutputFormat>application/xml</ows:OutputFormat>
</ows:AcceptFormats>
</csw:GetCapabilities>
</env:Body>
</env:Envelope>
```

DescribeRecord

DescribeRecord operation allows a client to discover elements of the information model supported by the target catalogue service. The operation allows some or all of the information model to be described.

Request examples

GET request:

```
http://localhost:8080/geonetwork/srv/en/csw?request=DescribeRecord&service=CSW&version=2
```

POST request:

```
Url:
http://localhost:8080/geonetwork/srv/en/csw

Content-type:
application/xml

Post data:
<?xml version="1.0" encoding="UTF-8"?>
<csw:DescribeRecord xmlns:csw="http://www.opengis.net/cat/csw/2.0.2" service="CSW" versi
```

SOAP request:

```
Url:
http://localhost:8080/geonetwork/srv/en/csw

Content-type:
application/soap+xml

Post data:
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Body>
    <csw:DescribeRecord xmlns:csw="http://www.opengis.net/cat/csw/2.0.2" service="CSW" v
  </env:Body>
</env:Envelope>
```

GetRecordById

GetRecordById request retrieves the default representation of catalogue metadata records using their identifier.

To retrieve non public metadata a previous **xml.user.login** service invocation is required. See *login service*.

Request examples

GET request:

```
http://localhost:8080/geonetwork/srv/en/csw?request=GetRecordById&service=CSW&version=2.
```

POST request:

```
Url:
http://localhost:8080/geonetwork/srv/en/csw

Content-type:
application/xml

Post data:
<?xml version="1.0" encoding="UTF-8"?>
  <csw:GetRecordById xmlns:csw="http://www.opengis.net/cat/csw/2.0.2" service="CSW" vers
  <csw:Id>5df54bf0-3a7d-44bf-9abf-84d772da8df1</csw:Id>
  <csw:ElementSetName>full</csw:ElementSetName>
</csw:GetRecordById>
```

SOAP request:

```
Url:
http://localhost:8080/geonetwork/srv/en/csw

Content-type:
application/soap+xml

Post data:
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Body>
    <csw:GetRecordById xmlns:csw="http://www.opengis.net/cat/csw/2.0.2" service="CSW" ve
      <csw:Id>5df54bf0-3a7d-44bf-9abf-84d772da8df1</csw:Id>
      <csw:ElementSetName>full</csw:ElementSetName>
    </csw:GetRecordById>
  </env:Body>
</env:Envelope>
```

GetRecords

GetRecords request allows to query the catalogue metadata records specifying a query in OCG Filter or CQL languages.

To retrieve non public metadata a previous **xml.user.login** service invocation is required. See *login service*.

Request examples

GET request (using CQL language):

```
Url:
http://localhost:8080/geonetwork/srv/en/csw?request=GetRecords&service=CSW&version=2.0.2
```

POST request:

```
Url:
http://localhost:8080/geonetwork/srv/en/csw

Content-type:
application/xml

Post data:
<?xml version="1.0" encoding="UTF-8"?>
<csw:GetRecords xmlns:csw="http://www.opengis.net/cat/csw/2.0.2" service="CSW" version="
  <csw:Query typeNames="csw:Record">
    <csw:Constraint version="1.1.0">
      <Filter xmlns="http://www.opengis.net/ogc" xmlns:gml="http://www.opengis.net/gml">
        <PropertyIsLike wildCard="%" singleChar="_" escape="\\">
          <PropertyName>AnyText</PropertyName>
          <Literal>%africa%</Literal>
        </PropertyIsLike>
      </Filter>
    </csw:Constraint>
  </csw:Query>
</csw:GetRecords>
```

SOAP request:

```
Url:
http://localhost:8080/geonetwork/srv/en/csw

Content-type:
application/soap+xml

Post data:
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Body>
    <csw:GetRecords xmlns:csw="http://www.opengis.net/cat/csw/2.0.2" service="CSW" versi
      <csw:Query typeNames="csw:Record">
        <csw:Constraint version="1.1.0">
          <Filter xmlns="http://www.opengis.net/ogc" xmlns:gml="http://www.opengis.net/g
            <PropertyIsLike wildCard="%" singleChar="_" escape="\\">
              <PropertyName>AnyText</PropertyName>
              <Literal>%africa%</Literal>
            </PropertyIsLike>
          </Filter>
        </csw:Constraint>
      </csw:Query>
    </csw:GetRecords>
  </env:Body>
</env:Envelope>
```

The GeoNetwork opensource catalog CSW Publication service operations are accessible through the url:

http://localhost:8080/geonetwork/srv/en/csw-publication

Harvest

The **Harvest** operation defines an interface for indirectly creating, modifying and deleting catalogue records by invoking a CSW client harvesting run from the server to a specified target. It can be run in either synchronous or asynchronous mode and the harvesting run can be executed just once or periodically. This operation requires user authentification to be invoked.

Synchronous one-run Harvest example

POST request:

```
Url:
http://localhost:8080/geonetwork/srv/en/csw-publication

Content-type:
application/xml

Post data:
  <?xml version="1.0" encoding="UTF-8"?>
  <csw:Harvest xmlns:csw="http://www.opengis.net/cat/csw/2.0.2" xmlns:gmd="http://www.is
      <csw:Source>http://[ URL to the target CSW server ]?request=GetCapabilities&amp;se
      <csw:ResourceType>http://www.isotc211.org/schemas/2005/gmd/</csw:ResourceType>
  </csw:Harvest>
```

GET request:

```
Url:
  http://localhost:8080/geonetwork/srv/en/csw-publication?request=Harvest&service=CSW&ve
```

Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<csw:HarvestResponse xmlns:csw="http://www.opengis.net/cat/csw/2.0.2">
    <csw:TransactionResponse>
        <csw:TransactionSummary>
            <csw:totalInserted>22</csw:totalInserted>
            <csw:totalUpdated>0</csw:totalUpdated>
            <csw:totalDeleted>0</csw:totalDeleted>
        </csw:TransactionSummary>
    </csw:TransactionResponse>
</csw:HarvestResponse>
```

Aynchronous one-run Harvest example

POST request:

```
Url:
http://localhost:8080/geonetwork/srv/en/csw-publication

Content-type:
application/xml

Post data:
  <?xml version="1.0" encoding="UTF-8"?>
  <csw:Harvest xmlns:csw="http://www.opengis.net/cat/csw/2.0.2" xmlns:gmd="http://www.is
```

```
      <csw:Source>http://[ URL to the target CSW server ]?request=GetCapabilities&amp;se
      <csw:ResourceType>http://www.isotc211.org/schemas/2005/gmd/</csw:ResourceType>
      <csw:ResponseHandler>[ URI or email address of response handler ]</csw:ResponseHar
  </csw:Harvest>
```

GET request:

```
Url:
  http://localhost:8080/geonetwork/srv/en/csw-publication?request=Harvest&service=CSW&ve
```

Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<csw:HarvestResponse xmlns:csw="http://www.opengis.net/cat/csw/2.0.2">
  <csw:Acknowledgement timeStamp="2011-12-05T15:13:59">
    <csw:EchoedRequest>
        <csw:Harvest xmlns:csw="http://www.opengis.net/cat/csw/2.0.2" xmlns:gmd="http://
            <csw:Source>http://[ URL to the target CSW server ]?request=GetCapabilities&
            <csw:ResourceType>http://www.isotc211.org/schemas/2005/gmd/</csw:ResourceTyp
            <csw:ResponseHandler>[ URI or email address of response handler ]</csw:Respo
        </csw:Harvest>
    </csw:EchoedRequest>
    <csw:RequestId>e7684bec-1fa9-4053-814f-7ae970d7a4a1</csw:RequestId>
  </csw:Acknowledgement>
</csw:HarvestResponse>
```

Transaction

The **Transaction** operation defines an interface for creating, modifying and deleting catalogue records. This operation requires user authentification to be invoqued.

Insert operation example

POST request:

```
Url:
http://localhost:8080/geonetwork/srv/en/csw-publication

Content-type:
application/xml

Post data:
<?xml version="1.0" encoding="UTF-8"?>
<csw:Transaction xmlns:csw="http://www.opengis.net/cat/csw/2.0.2" version="2.0.2" servic
  <csw:Insert>
    <gmd:MD_Metadata xmlns:gmd="http://www.isotc211.org/2005/gmd" xmlns:xsi="http://www.
    ...
    </gmd:MD_Metadata>
  </csw:Insert>
</csw:Transaction>
```

Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<csw:TransactionResponse xmlns:csw="http://www.opengis.net/cat/csw/2.0.2">
```

```
  <csw:TransactionSummary>
    <csw:totalInserted>1</csw:totalInserted>
    <csw:totalUpdated>0</csw:totalUpdated>
    <csw:totalDeleted>0</csw:totalDeleted>
  </csw:TransactionSummary>
</csw:TransactionResponse>
```

Update operation example

POST request:

```
Url:
http://localhost:8080/geonetwork/srv/en/csw

Content-type:
application/xml

Post data:
<?xml version="1.0" encoding="UTF-8"?>
<csw:Transaction xmlns:csw="http://www.opengis.net/cat/csw/2.0.2" version="2.0.2" servic
  <csw:Update>
    <gmd:MD_Metadata xmlns:gmd="http://www.isotc211.org/2005/gmd" xmlns:xsi="http://www.
    ...
    </gmd:MD_Metadata>
    <csw:Constraint version="1.1.0">
      <ogc:Filter>
        <ogc:PropertyIsEqualTo>
          <ogc:PropertyName>title</ogc:PropertyName>
          <ogc:Literal>Eurasia</ogc:Literal>
        </ogc:PropertyIsEqualTo>
      </ogc:Filter>
    </csw:Constraint>
  </csw:Update>
</csw:Transaction>
```

Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<csw:TransactionResponse xmlns:csw="http://www.opengis.net/cat/csw/2.0.2">
  <csw:TransactionSummary>
    <csw:totalInserted>0</csw:totalInserted>
    <csw:totalUpdated>1</csw:totalUpdated>
    <csw:totalDeleted>0</csw:totalDeleted>
  </csw:TransactionSummary>
</csw:TransactionResponse>
```

Delete operation example

POST request:

```
Url:
http://localhost:8080/geonetwork/srv/en/csw

Content-type:
application/xml
```

```
Post data:
<?xml version="1.0" encoding="UTF-8"?>
<csw:Transaction xmlns:csw="http://www.opengis.net/cat/csw/2.0.2" xmlns:ogc="http://www.
  <csw:Delete>
    <csw:Constraint version="1.1.0">
      <ogc:Filter>
        <ogc:PropertyIsEqualTo>
          <ogc:PropertyName>title</ogc:PropertyName>
          <ogc:Literal>africa</ogc:Literal>
        </ogc:PropertyIsEqualTo>
      </ogc:Filter>
    </csw:Constraint>
  </csw:Delete>
</csw:Transaction>
```

Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<csw:TransactionResponse xmlns:csw="http://www.opengis.net/cat/csw/2.0.2">
  <csw:TransactionSummary>
    <csw:totalInserted>0</csw:totalInserted>
    <csw:totalUpdated>0</csw:totalUpdated>
    <csw:totalDeleted>1</csw:totalDeleted>
  </csw:TransactionSummary>
</csw:TransactionResponse>
```

Errors

- User is not authenticated:

```
<?xml version="1.0" encoding="UTF-8"?>
<ows:ExceptionReport xmlns:ows="http://www.opengis.net/ows" xmlns:xsi="http://www.w
  <ows:Exception exceptionCode="NoApplicableCode">
    <ows:ExceptionText>Cannot process transaction: User not authenticated.</ows:Exc
  </ows:Exception>
</ows:ExceptionReport>
```

6.24 Java development with XML services

In this chapter are shown some examples to access GeoNetwork XML services in Java. Apache http commons library is used to send the requests and retrieve the results.

6.24.1 Retrieve groups list

This example shows a simple request, without requiring authentication, to retrieve the GeoNetwork groups.

Source

```
package org.geonetwork.xmlservices.client;

import org.apache.commons.httpclient.HttpClient;
import org.apache.commons.httpclient.methods.PostMethod;
import org.apache.commons.httpclient.methods.StringRequestEntity;
import org.jdom.Document;
import org.jdom.Element;

public class GetGroupsClient {

  public static void main(String args[]) {
    **// Create request xml**
    Element request = new Element("request");
    **// Create PostMethod specifying service url**
    String serviceUrl = "http://localhost:8080/geonetwork/srv/en/xml.group.list";
    PostMethod post = new PostMethod(serviceUrl);

    try {
      String postData = Xml.getString(new Document(request));

      **// Set post data, mime-type and encoding**
      post.setRequestEntity(new StringRequestEntity(postData, "application/xml", "UTF8")

      **// Send request**
      HttpClient httpclient = new HttpClient();
      int result = httpclient.executeMethod(post);

      **// Display status code**
      System.out.println("Response status code: " + result);

      **// Display response**
      System.out.println("Response body: ");
      System.out.println(post.getResponseBodyAsString());

    } catch (Exception ex) {
      ex.printStackTrace();

    } finally {
      **// Release current connection to the connection pool
      // once you are done**
      post.releaseConnection();
    }
  }
}
```

Output

```
Response status code: 200

Response body:
<?xml version="1.0" encoding="UTF-8"?>
  <response>
  <record>
    <id>2</id>
```

```
    <name>sample</name>
    <description>Demo group</description>
    <email>group@mail.net</email>
    <referrer />
    <label>
      <en>Sample group</en>
      <fr>Sample group</fr>
      <es>Sample group</es>
      <de>Beispielgruppe</de>
      <nl>Voorbeeldgroep</nl>
    </label>
  </record>
</response>
```

6.24.2 Create a new user (exception management)

This example show a request to create a new user, that requires authentication to complete succesfully. The request is executed without authentication to capture the exception returned by GeoNetwork.

Source

```
package org.geonetwork.xmlservices.client;

import org.apache.commons.httpclient.HttpClient;
import org.apache.commons.httpclient.HttpStatus;
import org.apache.commons.httpclient.methods.PostMethod;
import org.apache.commons.httpclient.methods.StringRequestEntity;
import org.jdom.Document;
import org.jdom.Element;

public class CreateUserClient {
  public static void main(String args[]) {

  **// Create request** xml
  Element request = new Element("request")
  .addContent(new Element("operation").setText("newuser"))
  .addContent(new Element("username").setText("samantha"))
  .addContent(new Element("password").setText("editor2"))
  .addContent(new Element("profile").setText("Editor"))
  .addContent(new Element("name").setText("Samantha"))
  .addContent(new Element("city").setText("Amsterdam"))
  .addContent(new Element("country").setText("Netherlands"))
  .addContent(new Element("email").setText("samantha@mail.net"));

  **// Create PostMethod specifying service url**
  String serviceUrl = "http://localhost:8080/geonetwork/srv/en/user.update";
  PostMethod post = new PostMethod(serviceUrl);

  try {
    String postData = Xml.getString(new Document(request));

    **// Set post data, mime-type and encoding**
    post.setRequestEntity(new StringRequestEntity(postData, "application/xml", "UTF8"));
```

```
    **// Send request**
    HttpClient httpclient = new HttpClient();
    int result = httpclient.executeMethod(post);

    **// Display status code**
    System.out.println("Response status code: " + result);

    **// Display response**
    System.out.println("Response body: ");
    String responseBody = post.getResponseBodyAsString();
    System.out.println(responseBody);

    if (result != HttpStatus.SC_OK) {
      **// Process exception**
      Element response = Xml.loadString(responseBody, false);
      System.out.println("Error code: " +
      response.getAttribute("id").getValue());
      System.out.println("Error message: " +
      response.getChildText("message"));
    }

    } catch (Exception ex) {
      ex.printStackTrace();

    } finally {
      // Release current connection to the connection pool
      // once you are done
      post.releaseConnection();
    }
  }
}
```

Output

```
Response status code: 401

Response body:
<?xml version="1.0" encoding="UTF-8"?>
<error id="service-not-allowed">
  <message>Service not allowed</message>
  <class>ServiceNotAllowedEx</class>
  <stack>
    <at class="jeeves.server.dispatchers.ServiceManager" file="ServiceManager.java" line
    <at class="jeeves.server.JeevesEngine" file="JeevesEngine.java" line="621" method="d
    <at class="jeeves.server.sources.http.JeevesServlet" file="JeevesServlet.java" line=
    <at class="jeeves.server.sources.http.JeevesServlet" file="JeevesServlet.java" line=
    <at class="javax.servlet.http.HttpServlet" file="HttpServlet.java" line="727" method
    <at class="javax.servlet.http.HttpServlet" file="HttpServlet.java" line="820" method
    <at class="org.mortbay.jetty.servlet.ServletHolder" file="ServletHolder.java" line="
    <at class="org.mortbay.jetty.servlet.ServletHandler" file="ServletHandler.java" line
    <at class="org.mortbay.jetty.security.SecurityHandler" file="SecurityHandler.java" l
    <at class="org.mortbay.jetty.servlet.SessionHandler" file="SessionHandler.java" line
  </stack>
  <object>user.update</object>
  <request>
    <language>en</language>
```

```
    <service>user.update</service>
  </request>
</error>
```

Error code: service-not-allowed Error message: Service not allowed

6.24.3 Create a new user (sending credentials)

This example show a request to create a new user, that requires authentication to complete succesfully.

In this example **httpClient** it's used first to send a login request to GeoNetwork, getting with **JSESSIONID** cookie. Nexts requests send to GeoNetwork using **httpClient** send the **JSESSIONID** cookie, and are managed as authenticated requests.

Source

```
package org.geonetwork.xmlservices.client;

import org.apache.commons.httpclient.Credentials;
import org.apache.commons.httpclient.HttpClient;
import org.apache.commons.httpclient.HttpStatus;
import org.apache.commons.httpclient.UsernamePasswordCredentials;
import org.apache.commons.httpclient.auth.AuthScope;
import org.apache.commons.httpclient.methods.PostMethod;
import org.apache.commons.httpclient.methods.StringRequestEntity;
import org.jdom.Document;
import org.jdom.Element;

public class CreateUserClientAuth {
  private HttpClient httpclient;

  CreateUserClientAuth() {
    httpclient = new HttpClient();
  }

  **/\**
  * Authenticates the user in GeoNetwork and send a request
  * that needs authentication to create a new user
  *
  \*/**
  public void sendRequest() {
    **// Authenticate user**
    if (!login()) System.exit(-1);

    **// Create request XML**
    Element request = new Element("request")
    .addContent(new Element("operation").setText("newuser"))
    .addContent(new Element("username").setText("samantha"))
    .addContent(new Element("password").setText("editor2"))
    .addContent(new Element("profile").setText("Editor"))
    .addContent(new Element("name").setText("Samantha"))
    .addContent(new Element("city").setText("Amsterdam"))
    .addContent(new Element("country").setText("Netherlands"))
    .addContent(new Element("email").setText("samantha@mail.net"));
```

```
  **// Create PostMethod specifying service url**
  String serviceUrl = "http://localhost:8080/geonetwork/srv/en/user.update";
  PostMethod post = new PostMethod(serviceUrl);

  try {
    String postData = Xml.getString(new Document(request));

    **// Set post data, mime-type and encoding**
    post.setRequestEntity(new StringRequestEntity(postData, "application/xml", "UTF8")

    **// Send request**
    **(httpClient has been set in
    // login request with JSESSIONID cookie)**
    int result = httpclient.executeMethod(post);

    **// Display status code**
    System.out.println("Create user response status code: " + result);

    if (result != HttpStatus.SC_OK) {
      **// Process exception**
      String responseBody = post.getResponseBodyAsString();
      Element response = Xml.loadString(responseBody, false);
      System.out.println("Error code: " +
      response.getAttribute("id").getValue());
      System.out.println("Error message: " +
      response.getChildText("message"));
    }

  } catch (Exception ex) {
    ex.printStackTrace();

  } finally {
    **// Release current connection to the connection pool
    // once you are done**
    post.releaseConnection();
  }
}

**/\**
* Logins a user in GeoNetwork
*
* After login **httpClient** gets with JSSESIONID cookie. Using it
* for nexts requests, these are managed as "authenticated requests"
*
* @return  True if login it's ok, false otherwise
\*/**
private boolean login() {
  **// Create request XML**
  Element request = new Element("request")
  .addContent(new Element("username").setText("admin"))
  .addContent(new Element("password").setText("admin"));

  **// Create PostMethod specifying login service url**
  String loginUrl =
  "http://localhost:8080/geonetwork/srv/en/xml.user.login";
  PostMethod post = new PostMethod(loginUrl);
```

```
    try {
      String postData = Xml.getString(new Document(request));

      **// Set post data, mime-type and encoding**
      post.setRequestEntity(new StringRequestEntity(postData,
      "application/xml", "UTF8"));

      **// Send login request**
      int result = httpclient.executeMethod(post);

      **// Display status code and authentication session cookie**
      System.out.println("Login response status code: " + result);
      System.out.println("Authentication session cookie: " +
      httpclient.getState().getCookies()[0]);

      return (result == HttpStatus.SC_OK);

    } catch (Exception ex) {
      ex.printStackTrace();
      return false;

    } finally {
      // Release current connection to the connection pool
      // once you are done
      post.releaseConnection();
    }

  }

  public static void main(String args[]) {
    CreateUserClientAuth request = new CreateUserClientAuth();

    request.sendRequest();
  }
}
```

Output

```
Login response status code: 200
Authentication session cookie: JSESSIONID=ozj8iyva0agv
Create user response status code: 200
```

Trying to run again the program, as the user it's just created we get an exception:

```
Login response status code: 200
Authentication session cookie: JSESSIONID=1q09kwg0r6fqe
Create user response status code: 500
```

Error response:

```
<?xml version="1.0" encoding="UTF-8"?>
<error id="error">
  <message>ERROR: duplicate key violates unique constraint "users_username_key"</message
  <class>PSQLException</class>
  <stack>
    <at class="org.postgresql.core.v3.QueryExecutorImpl" file="QueryExecutorImpl.java" l
```

```
    <at class="org.postgresql.core.v3.QueryExecutorImpl" file="QueryExecutorImpl.java" l
    <at class="org.postgresql.core.v3.QueryExecutorImpl" file="QueryExecutorImpl.java" l
    <at class="org.postgresql.jdbc2.AbstractJdbc2Statement" file="AbstractJdbc2Statement
    <at class="org.postgresql.jdbc2.AbstractJdbc2Statement" file="AbstractJdbc2Statement
    method="executeWithFlags" />
    <at class="org.postgresql.jdbc2.AbstractJdbc2Statement" file="AbstractJdbc2Statement
    method="executeUpdate" />
    <at class="jeeves.resources.dbms.Dbms" file="Dbms.java" line="261" method="execute"
    <at class="org.fao.geonet.services.user.Update" file="Update.java" line="134" metho
    <at class="jeeves.server.dispatchers.ServiceInfo" file="ServiceInfo.java" line="238'
    <at class="jeeves.server.dispatchers.ServiceInfo" file="ServiceInfo.java" line="141'
  </stack>
  <request>
    <language>en</language>
    <service>user.update</service>
  </request>
</error>
```

Error code: error Error message: ERROR: duplicate key violates unique constraint "users_username_key"

CHAPTER 7

Settings hierarchy

7.1 Introduction

GeoNetwork stores configuration settings in the Settings table. The settings are organized into a 'tree' structure where the leaf nodes of the tree are key/value pairs.

A key is limited to 64 characters whilst a value is effectively unlimited in most databases supported by GeoNetwork.

The root of the tree contains two nodes:

- **system**: contains nodes that describe system configuration settings
- **harvesting**: contains nodes that describe the settings used by harvesters

In the following sections:

- the indentation is used to show hierarchy
- names in **bold** represent keys
- the datatype of the value is shown in parenthesis after the key.
- an *italic* font is used to indicate types: string, integer, boolean, enumerated (values separated by a '|' represent the set of allowed values).
- a missing datatype means that the value of the node is not used.
- square brackets indicate cardinality. If they are missing, a cardinality of [1..1] should be inferred.

7.2 System node

Some examples of system configuration settings nodes contained in the system node are shown in the hierarchy below.

- **system**: information on system configuration settings
    - **site**: Contains information about the site
    - **name** (*string*): Name used to present this site to other sites. Used to fill comboboxes or lists.

- **organisation** (*string*): Name of the organization/company/institute that is running GeoNetwork
- **siteId** (*string*): A UUID that uniquely identifies the site. It is generated by the installer.
- **platform**: Contains information about the current software version
- **version** (*string*): GeoNetwork's version in the X.Y.Z format
- **subVersion** (*string*): A small extra description about the version, like 'alpha-1', 'beta' etc...
- **server**: Used when it is necessary to build absolute URLs to the GeoNetwork server. This is the case, for example, when creating links inside a metadata record or when providing CSW capabilities.
- **host** (*string*): HTTP server address
- **port** (*integer*): HTTP server port (can be empty which means port 80)
- **Intranet**: specify the network address details of the Intranet
- **network** (*string*): Intranet address
- **netmask** (*string*): Intranet netmask
- **z3950**: A container for Z39.50 server parameters
- **enable** (*boolean*): If true, GeoNetwork will start the Z30.50 server
- **port** (*integer*): The port opened by GeoNetwork to listen to Z39.50 requests. Usually 2100.
- **proxy**: This container specifies the proxy configuration for GeoNetwork to use when making outgoing connections
- **use** (*boolean*): If true, GeoNetwork will use the given proxy for outgoing connections
- **host** (*string*): Proxy host name
- **port** (*integer*): Proxy host port
- **username** (*string*): Proxy credentials.
- **password** (*string*): Proxy credentials.
- **feedback**: email messages sent via web feedback form or when downloading a resource.
- **email** (*string*): email address of person who will receive feedback
- **mailServer**: This container configures the mail server that will be used to send email
  - **host** (*string*): Address of the SMTP server to use
  - **port** (*string*): SMTP port on server to use
- **removedMetadata**: This container contains settings about removed metadata.
- **dir**: This folder will contain removed metadata in MEF format. It gets populated when the user deletes a metadata using the web interface.
- **LDAP**: Parameters for LDAP authentication by GeoNetwork

- **use** (*boolean*): If true, GeoNetwork will use LDAP authentication for usernames and passwords
- **host** (*string*): LDAP host
- **port** (*integer*): Port number on LDAP host
- **defaultProfile** (*string*): Default GeoNetwork profile to use when the profile user attribute does not exist in the LDAP attributes.
- **login** - optional credentials used to obtain access to the LDAP service
  - **userDN** (*string*)
  - **password** (*string*)
- **distinguishedNames** - optional LDAP info used by GeoNetwork to locate user credentials
  - **base** (*string*)
  - **users** (*string*)
- **userAttribs**: A container for user attributes present into the LDAP directory that must be retrieved and used to create the user in GeoNetwork.
  - **name** (*string*)
  - **password** (*string*)
  - **profile** (*string*)

7.3 Harvesting nodes

All harvesters share a common set of nodes, which are retrieved by the harvesting engine for all harvesters. These common nodes are described in the hierarchy below.

- **harvesting**
- **node** [0..n] (*geonetwork|csw|arcsde|filesystem|geonetwork20|oaipmh|ogcwxs|thredds|webdav|wfsfeatures|z3950|z3950Config|geoPREST*): Type of harvesting node
  - **site**: A container for site information.
    - **name** (*string*): Name of harvest instance (shown in the harvester user interface).
    - **uuid** (*string*): A unique identifier assigned by the system when the harvesting node is created.
    - **useAccount** (*boolean*): Indicates if the harvester has to authenticate to access the data.
      - **username** (*string*): credentials to be used for authentication
      - **password** (*string*): credentials to be used for authentication
  - **options**:
    - **every** (*string*): Harvesting schedule - similar to the Unix CRON format
    - **oneRunOnly** (*boolean*): If true, the harvester will harvest just once and then it will set the status to inactive.

        * **status** (*active|inactive*): Indicates if the harvesting from this node is stopped (inactive) or if the harvester is waiting for the next scheduled harvest (active).

    - **content**: A container for options relating to processing content after harvesting but before storage in the GeoNetwork database

        * **importxslt** (*string*): Name of XSLT to apply to metadata records before being stored and indexed in GeoNetwork (**note:** not every harvester supports this option - see notes section for each harvester in the harvester settings descriptions below). A list of the possible XSLTs that can be used for this parameter can be obtained from the `xml.harvesting.info` service - see *importStylesheets*.

        * **validate** (*boolean*): If true, the harvester will validate the metadata record against the matching metadata schema in GeoNetwork. (**note:** not every harvester supports this option - see notes section for each harvester in the harvester settings descriptions below)

    - **privileges** [0..1]: This is a container for privileges to assign to each harvested metadata record.

        * **group** (*integer*) [0..n]: A local group. The value is the local identifier of the group. There can be several group nodes each with its own set of privileges.

            · **operation** (*integer*) [0..n]: Privilege to assign to the group. The value is the numeric id of the operation like 0=view, 1=download, 2=edit etc...

    - **categories** [0..1]: This is a container for categories to assign to each harvested metadata record.

        * **category** (*integer*) [0..n]: A local category. The value is the local identifier of the category.

    - **info**: Container for status information about harvesting from this node.

        * **lastRun** (*string*): When the harvester was last run. The value is the current time in milliseconds since 1 January, 1970. If empty then the harvester has not yet been run.

Settings for each of the different harvesters that show the additional elements specific to those harvesters are in the following sections.

7.3.1 Harvesting node geonetwork

This is the native harvesting supported by GeoNetwork 2.1 and above.

- **node** (*string*): geonetwork

    - **site**: Contains host and account information

        * **host** (*string*) eg. http://localhost:8080/geonetwork

        * **createRemoteCategory** (*boolean*) True: If local category exists with same name as the remote category, then assign harvested metadata to that category. False: Ignore categories.

        * **mefFormatFull** (*boolean*) True: harvest remote metadata as a **full** MEF file. ie. metadata plus thumbnails and data files uploaded with metadata.

        * **importXslt** (*string*) Name and parameters of a metadata processing XSLT - see *Metadata Processing services* for more details.

  - **search** [0..n]: Contains the search parameters. If this element is missing, an unconstrained search will be performed.
    * **freeText** (*string*) - free text search on remote instance
    * **title** (*string*) - search title on remote instance
    * **abstract** (*string*) - search abstract on remote instance
    * **keywords** (*string*) - search keywords on remote instance
    * **digital** (*boolean*) - search for records marked 'digital' on remote instance
    * **hardcopy** (*boolean*) - search for records marked 'hardcopy' on remote instance
    * **sourceUuid** (*string*) - search for records that come from source with this uuid on the remote instance
    * **sourceName** (*string*) - search for records that come from source with this name on the remote instance
  - **groupsCopyPolicy** [0..n]: Represents a copy policy for a remote group. It is used to maintain remote privileges on harvested metadata.
    * **name** (*string*): Internal name (not localised) of a remote group.
    * **policy** (*string*): Copy policy. For the group all, policies are: copy, copyToIntranet. For all other groups, policies are: copy, createAndCopy. The Intranet group is not considered.

XML Example

Harvester settings in XML are used by the harvester services - see *Harvesting services*.

Example of GeoNetwork harvester settings in XML::

```
<node id="954" type="geonetwork">
 <site>
   <name>test</name>
   <uuid>683bfc02-73e2-4100-a601-369936b6f82a</uuid>
   <account>
     <use>true</use>
     <username>admin</username>
     <password>admin</password>
   </account>
   <host>http://localhost:8080/geonetwork</host>
   <createRemoteCategory>true</createRemoteCategory>
   <mefFormatFull>true</mefFormatFull>
   <xslfilter />
 </site>
 <content>
   <validate>true</validate>
   <importxslt>none</importxslt>
 </content>
 <options>
   <every>0 0 0/3 ? * *</every>
   <oneRunOnly>false</oneRunOnly>
   <status>inactive</status>
 </options>
 <searches>
```

```
    <search>
      <freeText>Maps</freeText>
      <title></title>
      <abstract></abstract>
      <keywords></keywords>
      <digital>false</digital>
      <hardcopy>true</hardcopy>
      <source>
        <uuid>b7ce20f2-888a-4139-8802-916730c4be06</uuid>
        <name>My GeoNetwork catalogue</name>
      </source>
    </search>
  </searches>
  <categories>
    <category id="5" />
  </categories>
  <groupsCopyPolicy />
  <info>
    <lastRun />
    <running>false</running>
  </info>
</node>
```

Notes

- this harvester does not use the content/importXslt setting - instead a more sophisticated approach using the metadata processing services can be applied through the **importXslt** setting - for more details and an example see *Metadata Processing services* for more details.

7.3.2 Harvesting node webdav

This harvester type is capable of connecting to a web server which is WebDAV enabled (WebDAV is WEB Distributed Authoring and Versioning) or WAF (Web Accessible Folder) enabled.

- **node** (*string*): webdav
  - **site**: Contains the URL to connect to and account information
    - **url** (*string*): URL to connect to. Must be well formed, starting with `http://`, `file://` or a supported protocol.
    - **icon** (*string*): This is the icon that will be used as the metadata source logo. The image is taken from the `images/harvesting` folder and copied to the `images/logos` folder. A list of the possible values that can be used for this parameter can be obtained from the `xml.harvesting.info` service - see *icons*.
  - **options**
    - **recurse** (*boolean*): Indicates if the remote folder must be recursively scanned for metadata.
    - **validate** (*boolean*): If set, the harvester will validate the metadata against its schema and the metadata will be harvested only if it is valid.
    - **subtype** (*waf|webdav*): Indicates the type of server being harvested

XML Example

Harvester settings in XML are used by the harvester services - see *Harvesting services*.

Example of WEBDAV/WAF harvester settings in XML::

```
<node id="1039" type="webdav">
 <site>
   <name>burbble</name>
   <uuid>b64e006a-a26c-4268-85ec-0bf0450be966</uuid>
   <account>
     <use>true</use>
     <username>apples</username>
     <password>oranges</password>
   </account>
   <url>http://davtest.com/webdav</url>
   <icon>webdav.gif</icon>
 </site>
 <content>
   <validate>true</validate>
   <importxslt>none</importxslt>
 </content>
 <options>
   <every>0 0 0 ? * *</every>
   <oneRunOnly>false</oneRunOnly>
   <status>inactive</status>
   <validate>true</validate>
   <recurse>true</recurse>
   <subtype>webdav</subtype>
 </options>
 <privileges>
   <group id="1">
     <operation name="view" />
   </group>
 </privileges>
 <categories>
   <category id="2" />
 </categories>
 <info>
   <lastRun />
   <running>false</running>
 </info>
</node>
```

Notes

- this harvester does not use the content/importXslt setting

7.3.3 Harvesting node csw

This harvester type is capable of querying an OGC Catalogue Services for the Web (CSW) server and harvesting metadata.

- **node** (*string*): csw
  - **site**

        * **capabUrl** (*string*): URL of the GetCapabilities statement. This will be used to retrieve the operations and server address.
        * **icon** (*string*): This is the icon that will be used as the metadata source logo. The image is taken from the `images/harvesting` folder and copied to the `images/logos` folder. A list of the possible values that can be used for this parameter can be obtained from the `xml.harvesting.info` service - see *icons*.
    - **search** [0..n]: Contains search parameters. If this element is missing, an unconstrained search will be performed.
        * **freeText** (*string*)
        * **title** (*string*)
        * **abstract** (*string*)
        * **subject** (*string*)
        * **minscale** (*integer*) - minimum scale denominator
        * **maxscale** (*integer*) - maximum scale denominator

XML Example

Harvester settings in XML are used by the harvester services - see *Harvesting services*. *Example of CSW harvester settings in XML:*:

```
<node id="1122" type="csw">
 <site>
   <name>blonks</name>
   <uuid>723953dc-1308-4905-abc0-9869f18af132</uuid>
   <account>
     <use>true</use>
     <username>frogmouth</username>
     <password>tawny</password>
   </account>
   <capabilitiesUrl>http://cswserver.com/services/csw?request=GetCapabilities&amp;servic
   <icon>csw.gif</icon>
 </site>
 <content>
   <validate>false</validate>
   <importxslt>none</importxslt>
 </content>
 <options>
   <every>0 0 0 ? * *</every>
   <oneRunOnly>false</oneRunOnly>
   <status>inactive</status>
 </options>
 <searches>
   <search>
     <freeText>cobblers</freeText>
     <title />
     <abstract />
     <subject />
     <minscale>25000</minscale>
     <maxscale>400000</maxscale>
   </search>
```

```
 </searches>
 <privileges>
   <group id="1">
     <operation name="view" />
   </group>
 </privileges>
 <categories>
   <category id="2" />
 </categories>
 <info>
   <lastRun />
   <running>false</running>
 </info>
</node>
```

Notes

- this harvester does not use the content/importXslt setting

7.3.4 Harvesting node z3950

This harvester type is capable of querying one or more Z3950 servers and harvesting metadata.

- **node** (*string*): z3950
  - **site**
    - **query** (*string*): Z3950 query in Prefix Query Format (mandatory). See the discussion on PQF in the Z3950 Harvester section of the Users Manual and/or http://www.indexdata.com/yaz/doc/tools.html#PQF.
    - **icon** (*string*): This is the icon that will be used as the metadata source logo. The image is taken from the `images/harvesting` folder and copied to the `images/logos` folder. A list of the possible values that can be used for this parameter can be obtained from the `xml.harvesting.info` service - see *icons*.
    - **repositories**
      - **repository** [0..n]: Contains the Z3950 repository ids that will be harvested. Z3950 repository ids in GeoNetwork can be obtained through the *Site Information (xml.info)* service.

XML Example

Harvester settings in XML are used by the harvester services - see *Harvesting services*. *Example of Z3950 harvester settings in XML:*:

```
<node id="1090" type="z3950">
 <site>
   <name>testz3950</name>
   <uuid>8554ec2b-2b80-4f9e-9d9d-028e2407ee89</uuid>
   <account>
     <use>true</use>
     <username />
```

```
      <password />
    </account>
    <query>@attrset geo @attr 1=1016 @attr 4=6 @attr 2=3 "water"</query>
    <icon>Z3950.gif</icon>
    <repositories>
      <repository id="act" />
      <repository id="geonetwork-auscope" />
      <repository id="product" />
      <repository id="publication" />
      <repository id="source" />
      <repository id="geonetwork-aims" />
    </repositories>
  </site>
  <content>
    <validate>false</validate>
    <importxslt>none</importxslt>
  </content>
  <options>
    <every>0 0 0 ? * *</every>
    <oneRunOnly>false</oneRunOnly>
    <status>inactive</status>
  </options>
  <privileges>
    <group id="1">
      <operation name="view" />
      <operation name="dynamic" />
      <operation name="featured" />
    </group>
  </privileges>
  <categories>
    <category id="2" />
  </categories>
  <info>
    <lastRun />
    <running>false</running>
  </info>
</node>
```

7.3.5 Harvesting node oaipmh

This harvester type is capable of querying an OAIPMH (Open Archives Initiative Protocol for Metadata Harvesting version 2.0) server and harvesting metadata.

- **node** (*string*): oaipmh
  - **site**
    - **url** (*string*): OAIPMH server URL
    - **icon** (*string*): This is the icon that will be used as the metadata source logo. The image is taken from the `images/harvesting` folder and copied to the `images/logos` folder. A list of the possible values that can be used for this parameter can be obtained from the `xml.harvesting.info` service - see *icons*.
  - **options**

        * **validate** (*boolean*): Validate metadata before saving to database. This option will be deprecated in favour of content/validate.

    - **search** [0..n]: Contains search parameters which because they are constrained to follow the OAIPMH version 2.0 standard, are curiously limited to a date range, prefix and metadata set name. If this element is missing, an unconstrained search will be performed.

        * **from** (*string*) from date search
        * **until** (*string*) to date search
        * **set** (*string*) metadata set name on OAIPMH server (possible values can be retrieved using http://your_oaipmhservername?verb=ListSets)
        * **prefix** (*string*) metadata prefix name on OAIPMH server (possible values can be retrieved using http://your_oaipmhservername?verb=ListMetadataFormats)

XML Example

Harvester settings in XML are used by the harvester services - see *Harvesting services*. *Example of OAIPMH harvester settings in XML:*:

```
<node id="1152" type="oaipmh">
 <site>
   <name>oaitest</name>
   <uuid>0a3e4a4a-4c4c-4f82-860f-551b6cf12341</uuid>
   <account>
     <use>true</use>
     <username />
     <password />
   </account>
   <url>http://localhost:6060/joai</url>
   <icon>oai-mhp.gif</icon>
 </site>
 <content>
   <validate>false</validate>
   <importxslt>{IMPORTXSLT}</importxslt>
 </content>
 <options>
   <every>0 0 0 ? * *</every>
   <oneRunOnly>false</oneRunOnly>
   <status>inactive</status>
   <validate>false</validate>
 </options>
 <searches>
   <search>
     <from>2013-01-08</from>
     <until>2013-01-30</until>
     <set>iso19139</set>
     <prefix>gmd</prefix>
     <stylesheet />
   </search>
 </searches>
 <privileges>
   <group id="1">
     <operation name="view" />
   </group>
```

```
 </privileges>
 <categories>
   <category id="2" />
 </categories>
 <info>
   <lastRun />
   <running>false</running>
 </info>
</node>
```

Notes

- this harvester does not use the content/importXslt setting
- this harvester does not use the content/validate setting - this will change

7.3.6 Harvesting node thredds

This harvester type is capable of harvesting metadata from a THREDDS (Thematic Real-time Environmental Data Directory Service). The metadata fragments are:

- transformed into ISO19115/19139 (or profile) metadata fragments
- copied or linked into an ISO19115/19139 (or profile) template

The harvester supports metadata fragments following the Unidata Data Discovery Conventions (http://www.unidata.ucar.edu/software/netcdf-java/formats/DataDiscoveryAttConvention.html) from two types of THREDDS datasets:

- collections: essentially datasets that contain other datasets (like directories in a filesystem tree)
- atomics: datasets that do not contain other datasets (like files in a filesystem tree)

The settings for a THREDDS harvester type are:

- **node** (*string*): thredds
  - **site**
    - **url** (*string*): THREDDS catalog server URL - should always be the URL of an XML THREDDS catalog eg. http://motherlode.ucar.edu:8080/thredds/catalog/satellite/3.9/WEST-CONUS_4km/catalog.xml - HTML URLs will **not** work!
    - **icon** (*string*): This is the icon that will be used as the metadata source logo. The image is taken from the `images/harvesting` folder and copied to the `images/logos` folder. A list of the possible values that can be used for this parameter can be obtained from the `xml.harvesting.info` service - see *icons*.
  - **options**
    - **lang** (*string*) - three letter language code describing language of metadata being harvested (almost always **eng**) - this value is used to set the metadata language in the records harvested from THREDDS
    - **topic** (*string*) - ISO topic category describing the harvested metadata

* **createThumbnails** (*boolean*) - If true and the THREDDS catalog delivers WMS images for a dataset whose metadata is being harvested then a thumbnail will be created from the WMS for the harvested metadata
* **createServiceMd** (*boolean*) - If true then a skeleton ISO19119 service metadata record will be created for the services specified in the THREDDS catalog.
* **createCollectionDatasetMd** (*boolean*) - If true then metadata records will be created using harvested metadata fragments from collection datasets in the THREDDS catalog.
* **collectionGeneration** (*fragments|default*) - Use *fragments*. *default* refers to DIF metadata extraction which will be deprecated.
* **outputSchemaOnCollectionsFragments** (*string*) Metadata schema of ISO records that will be created from THREDDS catalog collection metadata. A list of possible values for this parameter can be obtained from the `xml.harvesting.info` service, see *threddsFragmentSchemas*.
* **collectionMetadataTemplate** (*integer*) Template record that harvested fragments will copied or linked into to create metadata records. Use http://your_geonetwork/geonetwork/srv/eng/xml.info?type=templates to get a list of template metadata record ids that can be used for this setting.
* **createCollectionSubtemplates** (*boolean*) - If true then subtemplates are created from the metadata fragments harvested from the THREDDS catalog datasets. The subtemplates are stored in the GeoNetwork database and linked into the **collectionMetadataTemplate** template record to create the harvested metadata records. If false, then subtemplates are not created, instead the harvested fragments are copied into the **collectionMetadataTemplate** record to create normal metadata records.
* **collectionFragmentStylesheet** (*string*) Stylesheet that turns collection metadata fragments in the THREDDS catalog into ISO metadata fragments. A list of possible values for this parameter can be obtained from the `xml.harvesting.info` service, see *threddsFragmentStylesheets*.
* **createAtomicDatasetMd** (*boolean*) - If true then metadata records will be created using harvested metadata fragments from atom datasets in the THREDDS catalog.
* **atomicGeneration** (*fragments|default*) - Use *fragments*. *default* refers to DIF metadata extraction which will be deprecated.
* **outputSchemaOnAtomicFragments** (*string*) Metadata schema of ISO records that will be created from THREDDS catalog atomic metadata. A list of possible values for this parameter can be obtained from the `xml.harvesting.info` service, see *threddsFragmentSchemas*.
* **atomicMetadataTemplate** (*integer*) Template record that harvested fragments will copied or linked into to create metadata records. Use http://your_geonetwork/geonetwork/srv/eng/xml.info?type=templates to get a list of template metadata record ids that can be used for this setting.
* **createAtomicSubtemplates** (*boolean*) - If true then subtemplates are created from the metadata fragments harvested from the THREDDS catalog atoms. The subtemplates are stored in the GeoNetwork database and linked into the **atomicMetadataTemplate** template record to create the harvested metadata records. If false, then subtemplates are not created, instead the harvested fragments are copied into the **atomicMetadataTemplate** record to create normal metadata records.

* **atomicFragmentStylesheet** (*string*) Stylesheet that turns atom metadata fragments in the THREDDS catalog into ISO metadata fragments. A list of possible values for this parameter can be obtained from the `xml.harvesting.info` service, see *thredds-FragmentStylesheets*.

XML Example

Harvester settings in XML are used by the harvester services - see *Harvesting services*. *Example of THREDDS harvester settings in XML:*

```
<node id="977" type="thredds">
 <site>
   <name>test thredds with motherlode</name>
   <uuid>b3307257-6b2a-48e7-80f5-74acadeff66f</uuid>
   <account>
     <use>true</use>
     <username />
     <password />
   </account>
   <url>http://motherlode.ucar.edu:8080/thredds/catalog/satellite/3.9/WEST-CONUS_4km/cat
   <icon>thredds.gif</icon>
 </site>
 <content>
   <validate>false</validate>
   <importxslt>{IMPORTXSLT}</importxslt>
 </content>
 <options>
   <every>0 0 0 ? * *</every>
   <oneRunOnly>false</oneRunOnly>
   <status>inactive</status>
   <lang>eng</lang>
   <topic>environment</topic>
   <createThumbnails>true</createThumbnails>
   <createServiceMd>true</createServiceMd>
   <createCollectionDatasetMd>true</createCollectionDatasetMd>
   <createAtomicDatasetMd>true</createAtomicDatasetMd>
   <ignoreHarvestOnAtomics>true</ignoreHarvestOnAtomics>
   <atomicGeneration>fragments</atomicGeneration>
   <modifiedOnly>true</modifiedOnly>
   <atomicFragmentStylesheet>thredds-metadata.xsl</atomicFragmentStylesheet>
   <atomicMetadataTemplate>7</atomicMetadataTemplate>
   <createAtomicSubtemplates>true</createAtomicSubtemplates>
   <outputSchemaOnAtomicsDIF />
   <outputSchemaOnAtomicsFragments>iso19139</outputSchemaOnAtomicsFragments>
   <ignoreHarvestOnCollections>true</ignoreHarvestOnCollections>
   <collectionGeneration>fragments</collectionGeneration>
   <collectionFragmentStylesheet>thredds-metadata.xsl</collectionFragmentStylesheet>
   <collectionMetadataTemplate>7</collectionMetadataTemplate>
   <createCollectionSubtemplates>true</createCollectionSubtemplates>
   <outputSchemaOnCollectionsDIF />
   <outputSchemaOnCollectionsFragments>iso19139</outputSchemaOnCollectionsFragments>
   <datasetCategory>2</datasetCategory>
 </options>
 <privileges>
   <group id="1">
     <operation name="view" />
```

```
    </group>
  </privileges>
  <categories>
    <category id="3" />
  </categories>
  <info>
    <lastRun />
    <running>false</running>
  </info>
</node>
```

Notes

- this harvester does not use the content/importXslt setting
- this harvester does not use the content/validate setting

7.3.7 Harvesting node wfsfeatures

This harvester type is capable of querying and harvesting metadata from the GetFeature response from an OGC Web Feature Service (WFS). The metadata fragments are:

- transformed into ISO19115/19139 (or profile) metadata fragments
- copied or linked into an ISO19115/19139 (or profile) template

The settings for a WFS Features harvester type are:

- **node** (*string*): wfsfeatures
  - **site**
    - **url** (*string*): OGC WFS service URL. eg. http://localhost:7070/deegree/services - the harvester adds the necessary parameters to call the GetFeature service with the query provided from the **options**.
    - **icon** (*string*): This is the icon that will be used as the metadata source logo. The image is taken from the `images/harvesting` folder and copied to the `images/logos` folder. A list of the possible values that can be used for this parameter can be obtained from the `xml.harvesting.info` service - see *icons*.
  - **options**
    - **lang** (*string*) - three letter language code describing language of metadata being harvested (almost always **eng**) - this value is used to set the metadata language in the records harvested from THREDDS
    - **query** (*string*) - An OGC WFS GetFeature query
    - **outputSchema** (*string*) Metadata schema of ISO records that will be created from WFS GetFeature harvester. A list of possible values for this parameter can be obtained from the `xml.harvesting.info` service, see *wfsFragmentSchemas*.
    - **stylesheet** (*string*) Stylesheet that turns WFS Features into ISO metadata fragments. A list of possible values for this parameter can be obtained from the `xml.harvesting.info` service, see *wfsFragmentStylesheets*.

* **streamFeatures** (*boolean*) - If true then responses will be saved to disk and features will be extracted using the streaming XML parser. Use for large WFS GetFeatures responses.
* **createSubtemplates** (*boolean*) - If true then subtemplates are created from the metadata fragments harvested from the WFS GetFeatures response. The subtemplates are stored in the GeoNetwork database and linked into the **templateId** template record to create the harvested metadata records. If false, then subtemplates are not created, instead the harvested fragments are copied into the **templateId** record to create normal metadata records.
* **templateId** (*integer*) Template record that harvested fragments will copied or linked into to create metadata records. Use http://your_geonetwork/geonetwork/srv/eng/xml.info?type=templates to get a list of template metadata record ids that can be used for this setting.
* **recordsCategory** (*integer*) Category id of GeoNetwork category that will be assigned to the metadata records created by the harvester. Use http://your_geonetwork/geonetwork/srv/eng/xml.info?type=categories to get a list of category ids that can be used for this setting.

XML Example

Harvester settings in XML are used by the harvester services - see *Harvesting services*. *Example of WFS Features harvester settings in XML:*

```
<node id="1201" type="wfsfeatures">
 <site>
   <name>test wfs</name>
   <uuid>588ea4fa-a105-40fd-9697-8082d23cc967</uuid>
   <account>
     <use>true</use>
     <username />
     <password />
   </account>
   <url>http://localhost:6060/deegree/services</url>
   <icon>wfs.gif</icon>
 </site>
 <content>
   <validate>false</validate>
   <importxslt>{IMPORTXSLT}</importxslt>
 </content>
 <options>
   <every>0 0 0 ? * *</every>
   <oneRunOnly>false</oneRunOnly>
   <status>inactive</status>
   <lang>eng</lang>
   <query>&lt;wfs:GetFeature service="WFS" version="1.1.0"
       xmlns:wfs="http://www.opengis.net/wfs"&gt;
    &lt;wfs:Query typeName="gboundaries"/&gt;&lt;/wfs:GetFeature&gt;</query>
   <outputSchema>iso19139</outputSchema>
   <stylesheet>geoserver_boundary_fragments.xsl</stylesheet>
   <streamFeatures>false</streamFeatures>
   <createSubtemplates>true</createSubtemplates>
   <templateId>2</templateId>
   <recordsCategory>2</recordsCategory>
```

```
 </options>
 <privileges>
   <group id="1">
     <operation name="view" />
     <operation name="dynamic" />
     <operation name="featured" />
   </group>
 </privileges>
 <categories>
   <category id="2" />
 </categories>
 <info>
   <lastRun />
   <running>false</running>
 </info>
</node>
```

Notes

- this harvester does not use the content/importXslt setting
- this harvester does not use the content/validate setting

7.3.8 Harvesting node filesystem

This harvester type is capable of querying and harvesting metadata (filename *.xml) from a directory tree on a filesystem accessible to the GeoNetwork server.

The settings for the filesystem harvester type are:

- **node** (*string*): filesystem
  - **site**
    - **directory** (*string*): This is the filesystem directory (on the server) from which metadata files (*.xml) will be harvested.
    - **recurse** (*boolean*): If true, then any directories in the fileystem directory will be checked for *.xml files (and any directories in those directories etc etc).
    - **nodelete** (*boolean*): If true, then metadata from previous executions of the filesystem harvester will be left in the catalog (ie. they will not be deleted)
    - **icon** (*string*): This is the icon that will be used as the metadata source logo. The image is taken from the `images/harvesting` folder and copied to the `images/logos` folder. A list of the possible values that can be used for this parameter can be obtained from the `xml.harvesting.info` service - see *icons*.

XML Example

Harvester settings in XML are used by the harvester services - see *Harvesting services*. *Example of Filesystem harvester settings in XML:*

```
<node id="1234" type="filesystem">
 <site>
   <name>/Users filesystem harvester</name>
   <uuid>c7b7660a-337a-4aa1-843b-648280ad8d86</uuid>
   <account>
     <use>false</use>
     <username />
     <password />
   </account>
   <directory>/Users</directory>
   <recurse>true</recurse>
   <nodelete>true</nodelete>
   <icon>filesystem.gif</icon>
 </site>
 <content>
   <validate>true</validate>
   <importxslt>DIF-to-ISO19139.xsl</importxslt>
 </content>
 <options>
   <every>0 0 0 ? * *</every>
   <oneRunOnly>false</oneRunOnly>
   <status>inactive</status>
 </options>
 <searches />
 <privileges>
   <group id="1">
     <operation name="view" />
   </group>
 </privileges>
 <categories>
   <category id="2" />
 </categories>
 <info>
   <lastRun />
   <running>false</running>
 </info>
</node>
```

7.3.9 Harvesting node arcsde

This harvester type is capable of harvesting metadata from an arcsde server.

The settings for the arcsde harvester type are:

- **node** (*string*): arcsde
  - **site**
    - **server** (*string*): This is the name of the arcde server.
    - **port** (*integer*): Port number of arcsde server.
    - **username** (*string*): arcsde username
    - **password** (*string*): arcsde password
    - **database** (*string*): Name of arcsde database

* **icon** (*string*): This is the icon that will be used as the metadata source logo. The image is taken from the `images/harvesting` folder and copied to the `images/logos` folder. A list of the possible values that can be used for this parameter can be obtained from the `xml.harvesting.info` service - see *icons*.

XML Example

Harvester settings in XML are used by the harvester services - see *Harvesting services*. *Example of arcsde harvester settings in XML:*

```
<node id="1259" type="arcsde">
 <site>
   <name>test arcsde harvester</name>
   <uuid>5b5070d3-464a-484e-941a-e56812e46431</uuid>
   <account>
     <use>false</use>
     <username />
     <password />
   </account>
   <server>arcsde_server.esri.com</server>
   <port>5151</port>
   <username>sde_user</username>
   <password>sde_password</password>
   <database>esri_sde</database>
   <icon>esri.gif</icon>
 </site>
 <content>
   <validate>true</validate>
   <importxslt>none</importxslt>
 </content>
 <options>
   <every>0 0 0 ? * *</every>
   <oneRunOnly>false</oneRunOnly>
   <status>inactive</status>
 </options>
 <searches />
 <categories>
   <category id="4" />
 </categories>
 <info>
   <lastRun />
   <running>false</running>
 </info>
</node>
```

Notes

- this harvester does not use the content/importXslt setting
- ArcSDE java API libraries need to be installed by the user in GeoNetwork (folder `INSTALL_DIR/web/geonetwork/WEB-INF/lib`), as these are proprietary libraries not distributed with GeoNetwork. The following jars are required:
- jpe_sdk.jar
- jsde_sdk.jar

Note: dummy-api-XXX.jar must be removed from `INSTALL_DIR/web/geonetwork/WEB-INF/lib`

7.3.10 Harvesting node ogcwxs

This harvester type is capable of harvesting metadata from the GetCapabilities response of OGC services.

The settings for an OGCWxS harvester type are:

- **node** (*string*): ogcwxs
  - **site**
    - **url** (*string*): OGC WxS service URL entry point. eg. http://localhost:7070/deegree/services - the harvester adds the necessary parameters to call the GetCapabilities service.
    - **ogctype** (*WMS1.0.0|WMS1.1.1|WMS1.3.0|WFS1.0.0|WFS1.1.0|*
      |WCS1.0.0|WPS0.4.0|WPS1.0.0|CSW2.0.2|SOS1.0.0): Type and version of OGC service being harvested.
    - **icon** (*string*): This is the icon that will be used as the metadata source logo. The image is taken from the `images/harvesting` folder and copied to the `images/logos` folder. A list of the possible values that can be used for this parameter can be obtained from the `xml.harvesting.info` service - see *icons*.
  - **options**
    - **lang** (*string*) - three letter language code describing language of metadata being harvested - this value is used to set the metadata language in the records created by this harvester. Use http://your_geonetwork/geonetwork/srv/eng/xml.info?type=isolanguages to get a list of language codes that can be used for this setting.
    - **topic** (*string*) - ISO Topic Category. List of values comes from codelist of ISO topic category element (gmd:topicCategory).
    - **createThumbnails** (*boolean*) - if true and service is a WMS then thumbnails will be created for dataset metadata, if false then thumbnails will not be created
    - **useLayer** (*boolean*) - Create metadata for layer elements using GetCapabilities information
    - **useLayerMd** (*boolean*) - Create metadata for layer elements using remote metadata provided in the MetadataURL attribute (if the remote metadata doesn't exist/isn't recognized then GetCapabilities info will be used).
    - **datasetCategory** - (*integer*) Category id of GeoNetwork category that will be assigned to the dataset metadata records created by the harvester. Use http://your_geonetwork/geonetwork/srv/eng/xml.info?type=categories to get a list of category ids that can be used for this setting.
    - **outputSchema** (*string*) Metadata schema of ISO records that will be created from OGCWxS GetCapabilities harvester. A list of possible values for this parameter can be obtained from the `xml.harvesting.info` service, see *ogcwxsOutputSchemas*.

XML Example

Harvester settings in XML are used by the harvester services - see *Harvesting services*. *Example of OGCWxS GetCapabilities harvester settings in XML:*

```
<node id="1138" type="ogcwxs">
 <site>
   <name>test ogcwxs</name>
   <uuid>87852766-536a-40b7-875a-3ab5ff24fa78</uuid>
   <account>
     <use>false</use>
     <username />
     <password />
   </account>
   <url>http://localhost:6060/deegree2/services</url>
   <ogctype>WFS1.1.0</ogctype>
   <icon>default.gif</icon>
 </site>
 <content>
   <validate>false</validate>
   <importxslt>none</importxslt>
 </content>
 <options>
   <every>0 0 0 ? * *</every>
   <oneRunOnly>true</oneRunOnly>
   <status>inactive</status>
   <lang>eng</lang>
   <topic>elevation</topic>
   <createThumbnails>true</createThumbnails>
   <useLayer>true</useLayer>
   <useLayerMd>true</useLayerMd>
   <datasetCategory>2</datasetCategory>
   <outputSchema>iso19139</outputSchema>
 </options>
 <privileges>
   <group id="1">
     <operation name="view" />
     <operation name="dynamic" />
     <operation name="featured" />
   </group>
 </privileges>
 <categories>
   <category id="3" />
 </categories>
 <info>
   <lastRun />
   <running>false</running>
 </info>
</node>
```

7.3.11 Harvesting node geoPREST

This harvester type is capable of querying a GeoPortal 9.3.x site and harvesting metadata using the GeoPortal REST API.

- **node** (*string*): geoPREST

  - **site**
    - **baseUrl** (*string*): Base URL of the GeoPortal server site. eg. http://yoursite.com/geoportal. REST URLs will be built from this URL.
    - **icon** (*string*): This is the icon that will be used as the metadata source logo. The image is taken from the `images/harvesting` folder and copied to the `images/logos` folder. A list of the possible values that can be used for this parameter can be obtained from the `xml.harvesting.info` service - see *icons*.
  - **search** [0..n]: Contains search parameters. If this element is missing, no results will be returned. You can use the Lucene query syntax documented at http://webhelp.esri.com/geoportal_extension/9.3.1/index.htm#srch_lucene.htm.
    - **freeText** (*string*)

XML Example

Harvester settings in XML are used by the harvester services - see *Harvesting services*. *Example of geoPREST harvester settings in XML:*:

```
<node id="978" type="geoPREST">
 <site>
   <name>qldtest</name>
   <uuid>4b9de966-1d5f-4133-97da-f4232fb7761a</uuid>
   <account>
     <use>true</use>
     <username />
     <password />
   </account>
   <baseUrl>http://qspatial.information.qld.gov.au/geoportal</baseUrl>
   <icon>default.gif</icon>
 </site>
 <content>
   <validate>false</validate>
   <importxslt>none</importxslt>
 </content>
 <options>
   <every>0 0 0 ? * *</every>
   <oneRunOnly>true</oneRunOnly>
   <status>inactive</status>
 </options>
 <searches>
   <search>
     <freeText>Barrier Reef</freeText>
   </search>
   <search>
     <freeText>Cadastral</freeText>
   </search>
 </searches>
 <privileges>
   <group id="1">
     <operation name="view" />
     <operation name="dynamic" />
     <operation name="featured" />
   </group>
 </privileges>
```

```
 <categories>
   <category id="2" />
 </categories>
 <info>
   <lastRun />
   <running>false</running>
   <result>
     <total>119</total>
     <added>87</added>
     <updated>32</updated>
     <unchanged>0</unchanged>
     <unknownSchema>0</unknownSchema>
     <removed>0</removed>
     <unretrievable>0</unretrievable>
     <badFormat>0</badFormat>
     <doesNotValidate>0</doesNotValidate>
   </result>
 </info>
</node>
```

Notes

- this harvester uses two REST services from the GeoPortal API:
- `rest/find/document` with searchText parameter to return an RSS listing of metadata records that meet the search criteria
- `rest/document` with id parameter from each result returned in the RSS listing
- this harvester has been tested with GeoPortal 9.3.x. It should be used for that version of GeoPortal in preference to the CSW harvester.

Index

E

Exception Handling, 58

M

mef.export, 176
mef.import, 177

R

rss.latest, 92
rss.search, 90

X

xml.category.create.update, 79
xml.category.get, 81
xml.category.remove, 82
xml.category.update, 80
xml.config.get, 139
xml.config.set, 145
xml.file.disclaimer, 154
xml.file.download, 156
xml.forward, 153
xml.group.create.update, 62
xml.group.get, 65
xml.group.remove, 66
xml.group.update, 64
xml.harvesting.add, 159
xml.harvesting.get, 157
xml.harvesting.history, 167
xml.harvesting.history.delete, 169
xml.harvesting.info, 160
xml.harvesting.remove, 166
xml.harvesting.run, 166
xml.harvesting.start, 166
xml.harvesting.stop, 166
xml.info, 146
xml.metadata.batch.delete, 100
xml.metadata.batch.newowner, 106
xml.metadata.batch.processing, 126
xml.metadata.batch.update.categories, 120
xml.metadata.batch.update.children, 128
xml.metadata.batch.update.privileges, 104
xml.metadata.batch.update.status, 115
xml.metadata.batch.version, 123
xml.metadata.category, 118
xml.metadata.delete, 99
xml.metadata.get, 87
xml.metadata.insert, 94
xml.metadata.privileges, 103
xml.metadata.processing, 125
xml.metadata.schema.add, 172
xml.metadata.schema.delete, 174
xml.metadata.schema.update, 173
xml.metadata.select, 101
xml.metadata.status, 113
xml.metadata.status.get, 117
xml.metadata.update, 97
xml.metadata.validation, 135
xml.metadata.version, 121
xml.ownership.editors, 109
xml.ownership.groups, 111
xml.ownership.transfer, 108
xml.relation, 130
xml.relation.delete, 134
xml.relation.get, 132
xml.relation.insert, 134
xml.schema.info, 170
xml.search, 84
xml.user.get, 68
xml.user.infoupdate, 75
xml.user.login, 60
xml.user.logout, 62
xml.user.metadata, 89
xml.user.pwupdate, 77
xml.user.remove, 78

xml.user.update, 70
xml.usergrpups.list, 67