
# GeoNetwork XML services manual

**By the developers**

**V 2.4**

# Table of Contents

# Preface

**About this Project.** This document provides guidelines to use GeoNetwork opensource XML services to develop third party applications. The GeoNetwork project started out as a Spatial Data Catalogue System for the Food and Agriculture organisation of the United Nations (FAO)[1], the United Nations World Food Programme (WFP)[2] and the United Nations Environmental Programme (UNEP)[3]. At present the project is widely used as the basis of Spatial Data Infrastructures all around the world. The project is part of the Open Source Geospatial Foundation (*OSGeo*) and can be found at http://geonetwork-opensource.org.



The document is written in DocBook format for consistency and portability.

**Author Information.** This framework and documentation was written by the GeoNetwork opensource Developers. If you have questions, found a bug or have enhancements, please contact us through the GeoNetwork opensource Development Mailing list at `<geonetwork-devel@lists.sourceforge.net>`


[1] http://www.fao.org
[2] http://vam.wfp.org
[3] http://www.unep.org
[4] http://www.osgeo.org
[5] http://creativecommons.org/licenses/by-nd/3.0/

# Part I. XML Services

This part gives some insights of GeoNetwork opensource XML services avalaible to develop third party applications that use these services.

# 1. Introduction

## 1.1 XML services

GeoNetwork provides access to several internal structures through the use of XML services. These are much like HTML addresses but return XML instead. As an example, consider the xml.info service: you can use this service to get some system's information without fancy styles and graphics. In GeoNetwork, XML services have usually the xml. prefix in their address.

The avalaible services are defined in *WEB-INF\config.xml* file and can be invoqued by the users depending on their profiles. The rights to invoque the different services are defined in *WEB-INF\user-profiles.xml* file.

In nexts chapters is going to be described the API for principal services providing examples of the invocation and responses returned.

## 1.2 Calling specifications

### Request

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

GET The parameters are sent using 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. Figure 1.1, "A GET request to a XML service and its request encoding" shows an example of such request and the parameters encoded in XML. POST There are 3 variants of this method:

*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.

*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.

*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.

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

*Figure 1.1. A GET request to a XML service and its request encoding*

## 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 into the document's prologue. Anyway, all GeoNetwork services return documents in the UTF-8 encoding.

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

## 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 an identifier of the error from a common set. See Table 1.1, "Summary of error ids" for a list of all id 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 completes the id code.
  - *class*: The Java class of the raised error (name without package information).
  - *stack*: The server's stacktrace up to the point that generated the exception. It contains several at children, one for each nested level. Useful for debugging purposes.
    - *at*: Information about a nested level of called code. It has the following mandatory attributes:

      *class* Java class of the called method. *method* Java called method. *line* Line, inside the called method's source code where there the method call of the next nested level. *file* Source file where the class is defined.
  - *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 some useful information that can be needed to debug the service.
    - *language*: Language used when the service was called.
    - *service*: Name of the called service.

*Table 1.1. Summary of error ids*

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

Figure 1.2, "An example of generated exception" shows 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.

```
<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>
```

*Figure 1.2. An example of generated exception*

# 2. Login and logout services

## 2.1 Login services

### GeoNetwork standard login (xml.user.login)

The *xml.user.login* service is used to authenticate the user in GeoNetwork, allowing using the Xml services that require authentication. For example, the services to maintain group or user information.

#### Request

Parameters:

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

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>
```

*Figure 2.1. Login request example*

#### Response

When user authentication is succesful the next response is received:

```
OK

Date: Mon, 01 Feb 2010 09:29:43 GMT
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
Expires: -1
Transfer-Encoding: chunked
Server: Jetty(6.1.14)
```

*Figure 2.2. Login succesful raw response example*

The authentication process sets *JSESSIONID* cookie with the authentication token that should be send in the services that need authentication to be invoqued. Otherwise, a *Service not allowed* exception will be returned by these services.

### Errors

- *Missing parameter (error id: missing-parameter)*, when mandatory parameters are not send. Returned 400 HTTP code
- *bad-parameter XXXX*, when an empty username or password is provided. Returned 400 HTTP code
- *User login failed (error id: user-login)*, when login information is not valid. Returned 400 HTTP code

Example returning *User login failed* exception:

```
<?xml version="1.0" encoding="UTF-8"?>
<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="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" />
    <at class="javax.servlet.http.HttpServlet" file="HttpServlet.java" line="820"
method="service" />
    <at class="org.mortbay.jetty.servlet.ServletHolder" file="ServletHolder.java" line="502"
method="handle" />
  </stack>
  <object>admin2</object>
  <request>
    <language>en</language>
    <service>user.login</service>
  </request>
</error>
```

*Figure 2.3. Login error response example*

## Shibboleth login (shib.user.login)

The *shib.user.login* service process the creadentials of a Shibboleth login.

To use this service the user previously should be authenticated to Shibboleth. If the authentication is succesful, the HTTP headers will contain the user credentials.

When calling *shib.user.login* service in GeoNetwork, the Shibboleth credentials are then used to find or create (if don't exists) the user account in GeoNetwork.

GeoNetwork processes the next HTTP header parameters filled by Shibboleth authentication:

- system/shib/attrib/username
- system/shib/attrib/surname

- system/shib/attrib/firstname
- system/shib/attrib/profile: User profile. Values: Administrator, UserAdmin, Reviewer, Editor and Guest

GeoNetwork checks if exists a user with the specified *username* in the users table, creating it if not found.

## 2.2 Logout service

### Logout (xml.user.logout)

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

#### Request

Parameters:

- *None*:This request requires no parameters, just it's required sending the *JSESSIONID* cookie value.

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/>
```

*Figure 2.4. Logout request example*

#### Response

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

*Figure 2.5. Logout response example*

# 3. Group services

## 3.1 Groups retrieving

### Groups list (xml.group.list)

The *xml.group.list* service can be used to retrieve the user groups avalaible in GeoNetwork.

Requires authentication: No

#### Request

Parameters:

- *None*

Example:

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

Mime-type:
application/xml

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

*Figure 3.1. Group list request example*

#### Response

Here follows the structure of the response:

- *record*: This is the container for each group element returned
  - *id*: Group identifier
  - *name*: Human readable group name
  - *description*: Group description
  - *email*: Group email address
  - *label*: This is just a container to hold the group names translated in the languages supported by GeoNetwork. Each translated label it's enclosed in a tag that identifies the language code

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
  <record>
    <id>2</id>
    <name>sample</name>
    <description />
    <email />
    <referrer />
    <label>
      <en>Sample group</en>
      <fr>Sample group</fr>
      <es>Sample group</es>
      <de>Beispielgruppe</de>
      <nl>Voorbeeldgroep</nl>
    </label>
  </record>
  <record>
    <id>3</id>
    <name>RWS</name>
    <description />
    <email />
    <referrer />
    <label>
      <de>RWS</de>
      <fr>RWS</fr>
      <en>RWS</en>
      <es>RWS</es>
      <nl>RWS</nl>
    </label>
  </record>
</response>
```

*Figure 3.2. Group list response example*

### Group information (group.get)

Retrieves group information. **Non XML response.**

## 3.2 Groups maintenance

### Create/update a group (group.update)

The *group.update* service can be used to create new groups and update the information of 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 it's created with name, description and email parameters provided.
- *name*: (mandatory) Name of the group
- *description*: Group description
- *email*: Mail address for the group

Example:

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

Mime-type:
application/xml

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

*Figure 3.3. Group update request example*

### Response

If request it's executed succesfully HTTP 200 status code it's returned. If request fails an HTTP status code error it's returned and the response contains the XML document with the exception.

### 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. Returned 401 HTTP code
- *Missing parameter (error id: missing-parameter)*, when mandatory parameters are not provided. Returned 400 HTTP code
- *bad-parameter name*, when *name* it's empty. Returned 400 HTTP code
- *ERROR: duplicate key violates unique constraint "groups_name_key"*, when trying to create a new group using an existing group name. Returned 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 groups 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 it's enclosed in a tag that identifies the language code

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>
```

*Figure 3.4. Group label update request example*

### Response

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

*Figure 3.5. Group label update response example*

### 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. Returned 401 HTTP code
- *Missing parameter (error id: missing-parameter)*, when mandatory parameters are not provided. Returned 400 HTTP code

## Remove a group (group.remove)

The *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

Example:

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

Mime-type:
application/xml

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

*Figure 3.6. Group remove request example*

## Response

If request it's executed succesfully HTTP 200 status code it's returned. If request fails an HTTP status code error it's returned and the response contains the XML document with the exception.

## 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. Returned 401 HTTP code
- *Missing parameter (error id: missing-parameter)*, when mandatory parameters are not provided. Returned 400 HTTP code
- *bad-parameter id*, when *id* parameter it's empty. Returned 400 HTTP code

# 4. User services

## 4.1 Users retrieving

### Users list (xml.user.list)

The *xml.user.list* service can be used to retrieve the users defined in GeoNetwork.

Requires authentication: Yes

#### Request

Parameters:

- *None*

Example:

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

Mime-type:
application/xml

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

*Figure 4.1. User list request example*

#### Response

Here follows the structure of the response:

- *record*: This is the container for each user element returned
  - *id*: User identifier
  - *username*: Login name for the user
  - *password*: Password encoded in md5
  - *surname*: User surname
  - *name*: User name
  - *profile*: User profile. The profiles defined in GeoNetwork are: Administrator, User administrator, Content Reviewer, Editor, Registered user
  - *address*: User physical address
  - *city*: User address city
  - *state*: User address state
  - *zip*: User address zip

- *country*: User address country
- *email*: User email address
- *organisation*: User organisation/department
- *kind*: Kind of organisation

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
  <record>
    <id>1</id>
    <username>admin</username>
    <password>d033e22ae348aeb566fc214aec3585c4da997</password>
    <surname>admin</surname>
    <name>admin</name>
    <profile>Administrator</profile>
    <address />
    <city />
    <state />
    <zip />
    <country />
    <email />
    <organisation />
    <kind />
  </record>
  <record>
    <id>2</id>
    <username>editor</username>
    <password>ab41949825606da179db7c89ddcedcc167b64847</password>
    <surname>Smith</surname>
    <name>John</name>
    <profile>Editor</profile>
    <address />
    <city>Amsterdam</city>
    <state />
    <zip />
    <country>nl</country>
    <email>john.smith@mail.com</email>
    <organisation />
    <kind>gov</kind>
  </record>
</response>
```

*Figure 4.2. User list response example*

Exceptions:

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

## User groups list (xml.usergroups.list)

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

Requires authentication: Yes

### Request

Parameters:

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

Example:

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

Mime-type:
application/xml

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

*Figure 4.3. User groups list request example*

### Response

Here follows the structure of the response:

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

```
<?xml version="1.0" encoding="UTF-8"?>
<groups>
  <group>
    <id>3</id>
    <name>RWS</name>
    <description />
  </group>
</groups>
```

*Figure 4.4. User groups list response example*

Exceptions:

- *Service not allowed (error id: service-not-allowed)*, when the user is not authenticated or his profile has no rights to execute the service
- *User XXXX doesn't exist*, if no exists a user with provided *id* value

## User information (user.get)

Retrieves user information. **Non XML response.**

## 4.2 Users maintenance

### Create a user (user.update)

The *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 where 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

Example:

```
Url:
http://localhost:8080/geonetwork/srv/en/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>
```

*Figure 4.5. User create request example*

### Response

If request it's executed succesfully HTTP 200 status code it's returned. If request fails an HTTP status code error it's returned and the response contains the XML document with the exception.

### 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. Returned 401 HTTP code
- *Missing parameter (error id: missing-parameter)*, when mandatory parameters are not provided
- *bad-parameter*, when a mandatory fields is empty
- *Unknow profile XXXX (error id: error)*, when the profile is not valid
- *ERROR: duplicate key violates unique constraint "users_username_key"*, when trying to create a new user using an existing username
- *ERROR: insert or update on table "usergroups" violates foreign key constraint "usergroups_groupid_fkey"*, when group identifier is not an existing group identifier
- *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 the *UserAdmin* user is not allowed to manage
- *ERROR: you don't have rights to do this*, when the authenticated user has a profile that is not *Administrator* or *UserAdmin*

## Update user information (user.update)

The *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

*Remarks*: If an optional parameter it's not provided the value it's updated in the database with an empty string.

Example:

```
Url:
http://localhost:8080/geonetwork/srv/en/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>
```

*Figure 4.6. Update user information request example*

### Response

If request it's executed succesfully HTTP 200 status code it's returned. If request fails an HTTP status code error it's returned and the response contains the XML document with the exception.

### 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. Returned 401 HTTP code
- *Missing parameter (error id: missing-parameter)*, when the mandatory parameters are not provided. Returned 400 HTTP code
- *bad-parameter*, when a mandatory field is empty. Returned 400 HTTP code
- *Unknow profile XXXX (error id: error)*, when the profile is not valid. Returned 500 HTTP code
- *ERROR: duplicate key violates unique constraint "users_username_key"*, when trying to create a new user using an existing username. Returned 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. Returned 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 the *UserAdmin* user is not allowed to manage. Returned 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*. Returned 500 HTTP code

## Reset user password (user.update)

The *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

Example:

```
Url:
http://localhost:8080/geonetwork/srv/en/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>
```

*Figure 4.7. Reset user password request example*

## Response

If request it's executed succesfully HTTP 200 status code it's returned. If request fails an HTTP status code error it's returned and the response contains the XML document with the exception.

## 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. Returned 401 HTTP code
- *Missing parameter (error id: missing-parameter)*, when the mandatory parameters are not provided. Returned 400 HTTP code
- *bad-parameter*, when a mandatory field is empty. Returned 400 HTTP code
- *Unknow profile XXXX (error id: error)*, when the profile is not valid. Returned 500 HTTP code
- *ERROR: you don't have rights to do this*, when the authenticated user has a profile that it's not *Administrator* or *UserAdmin*. Returned 500 HTTP code

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

The *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

*Remarks*: If an optional parameter it's not provided the value it's updated in the database with an empty string.

Example:

```
Url:
http://localhost:8080/geonetwork/srv/en/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>
```

*Figure 4.8. Current user info update request example*

### Response

If request it's executed succesfully HTTP 200 status code it's returned. If request fails an HTTP status code error it's returned and the response contains the XML document with the exception.

### Errors

- *Service not allowed (error id: service-not-allowed)*, when the user is not authenticated. Returned 401 HTTP code

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

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

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>
```

*Figure 4.9. User update password request example*

### Response

If request it's executed succesfully HTTP 200 status code it's returned. If request fails an HTTP status code error it's returned and the response contains the XML document with the exception.

### Errors

- *Service not allowed (error id: service-not-allowed)*, when the user is not authenticated. Returned 401 HTTP code
- *Old password is not correct*. Returned 500 HTTP code
- *Bad parameter (newPassword)*, when an empty password is provided. Returned 400 HTTP code

## Remove a user (user.remove)

The *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 himself), while users with profile *UserAdmin* can delete users only in the groups where they belong (except himself).

Requires authentification: Yes

## Request

Parameters:

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

Example:

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

Mime-type:
application/xml

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

*Figure 4.10. User remove request example*

## Response

If request it's executed succesfully HTTP 200 status code it's returned. If request fails an HTTP status code error it's returned and the response contains the XML document with the exception.

## 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. Returned 401 HTTP code
- *Missing parameter (error id: missing-parameter)*, when the *id* parameter is not provided. Returned 400 HTTP code
- *You cannot delete yourself from the user database (error id: error)*, when trying to delete the authenticated user himself. Returned 500 HTTP code
- *You don't have rights to delete this user (error id: error)*, when trying to delete using an authenticated user that don't belongs to *Administrator* or *User administrator* profiles. Returned 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 of the authenticated user (belonging the authenticated user to profile *User administrator*). Returned 500 HTTP code

# 5. Metadata services

## 5.1 Retrieve metadata services

### Search metadata (xml.search)

The *xml.search* service can be used to retrieve the metadata stored in 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

Examples:

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

Mime-type:
application/xml

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

*Figure 5.1. Request to search for all metadata example*

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

Mime-type:
application/xml

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

*Figure 5.2. Request with free text search example*

```
Url:
http://localhost:8080/geonetwork/srv/en/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>
```

*Figure 5.3. Request with a geographic search example*

```
Url:
http://localhost:8080/geonetwork/srv/en/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>
```

*Figure 5.4. Request to search using dates and keywords example*

## Response

The response is the metadata record with additional *geonet:info* section. The main fields for *geonet:info* 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
  - *metadata*: Container for metadata records found. Each record contains an *geonet:info* element with the following information:
    - *title*: RSS channel title
    - *description*: RSS channel description
    - *item*: Metadata RSS item (one item for each metadata retrieved)
      - *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

```
<?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>
```

*Figure 5.5. Metadata search response example*

## 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

Parameters (one of them mandatory):

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

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>
```

*Figure 5.6. Get metadata request example*

## Response

The response is the 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

```
<?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>
```

*Figure 5.7. Get metadata response example*

Exceptions:

- *Request must contain a UUID or an ID*, when no uuid or id parameter is provided

### Errors

- *Operation not allowed (error id: operation-not-allowed)*, when the user is not allowed to show the metadata record. Returned 403 HTTP code

## 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* file setting the next 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

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>
```

*Figure 5.8. RSS search request example*

### Response

Here follows 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
    - georrs:point: Bounding box in georss simplepoint format

```
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/georss"
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</link>
<link href="http://geonetwork3.fao.org/ows/296?SERVICE=wms$amp;VERSION=1.1.1&REQUEST=GetMap
&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</guid>
<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 simplepoint format (default) (http://georss.org)-->
<georss:point>16.9 1.8</georss:point>
</item>
</channel>
</rss>
```

*Figure 5.9. RSS latest response example*

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

The *rss.latest* service can be used to retrieve the latest added metadata records in RSS format. This service can be configured in *WEB-INF\config.xml* file setting the next parameters:

- *maxItems*: Maximum number of RSS records to retrieve (default = 20)
- *timeBetweenUpdates*: Minutes to query database for new metadata (default = 60)

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

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>
```

*Figure 5.10. RSS latest request example*

## Response

Here follows 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
    - georrs:where: Bounding box with the metadata extent

```
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/georss"
  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</link>
    <link href="http://geonetwork3.fao.org/ows/296?SERVICE=wms$amp;VERSION=1.1.1&REQUEST=GetMap
      &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</guid>
    <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>
```

*Figure 5.11. RSS latest response example*

## 5.2 Metadata administration services

### Update operations allowed for a metadata (metadata.admin)

The *metadata.admin* service updates the operations allowed for a metadata with the list of operations allowed send in the parameters, **deleting all the operations allowed assigned previously**.

Requires authentication: Yes

#### Request to metadata.admin service

Parameters:

- id: Identifier of metadata to update

- *_G_O*: *(can be multiple elements)*
  - G: Group identifier
  - O: Operation identifier

Operation identifiers:

- 0: view
- 1: download
- 2: editing
- 3: notify
- 4: dynamic
- 5: featured

Example:

```
POST

Url:
http://localhost:8080/geonetwork/srv/en/metadata.admin

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/en/metadata.admin?id=6&_1_2&_1_1
```

*Figure 5.12. Request metadata update operations allowed example*

## Response to metadata.admin service

The response contains the identifier of the metadata updated.

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

*Figure 5.13. Response metadata update operations allowed example*

## 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. Returned 401 HTTP code

- Metadata not found (error id: metadata-not-found) if not exists a metadata record with the identifier provided
- ERROR: insert or update on table "operationallowed" violates foreign key 'operationallowed_operationid_fkey », if an operation identifier provided is not valid
- ERROR: insert or update on table "operationallowed" violates foreign key 'operationallowed_groupid_fkey », if a group identifier provided is not valid

## Massive update privilegies (metadata.massive.update.privileges)

The *metadata.massive.update.privileges* service updates the operations allowed for a selected metadata with the list of operations allowed send in the parameters, **deleting all the operations allowed assigned previously**.

This service requires a previous call to *metadata.select* service to select the metadata records to update.

Requires authentication: Yes

### Request to metadata.select service

Parameters:

- id: Identifier of metadata to select
- *selected*: Selection state. Values: add, add-all, remove, remove-all

Example:

```
Select all metadata allowed

Url:
http://localhost:8080/geonetwork/srv/en/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

Url:
http://localhost:8080/geonetwork/srv/en/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
Url:
http://localhost:8080/geonetwork/srv/en/metadata.select

Mime-type:
application/xml

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

*Figure 5.14. Select metadata request examples*

## Response to metadata.select service

The response contains the number of metadata selected.

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

*Figure 5.15. Response select metadata example*

## Request to metadata.massive.update.privileges

Parameters:

- *_G_O*: *(can be multiple elements)*
  - G: Group identifier
  - O: Operation identifier

Operation identifiers:

- 0: view
- 1: download
- 2: editing
- 3: notify
- 4: dynamic
- 5: featured

Example:

```
POST

Url:
http://localhost:8080/geonetwork/srv/en/metadata.massive.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/en/metadata.massive.update.privileges?_1_2&_1_1
```

*Figure 5.16. Request metadata massive update privilegies example*

## Response to metadata.massive.update.privileges

If request is executed succesfully HTTP 200 status code is returned. If request fails an HTTP status code error is returned and the response contains the XML document with the exception.

## 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. Returned 401 HTTP code
- Metadata not found (error id: metadata-not-found) if not exists a metadata record with the identifier provided

- ERROR: insert or update on table "operationallowed" violates foreign key 'operationallowed_operationid_fkey », if an operation identifier provided is not valid
- ERROR: insert or update on table "operationallowed" violates foreign key 'operationallowed_groupid_fkey », if a group identifier provided is not valid

## 5.3 Metadata ownership services

This services allow to manage the metadata ownership (the user who created the metadata), for example to get information about the users who created metadata records or transfer the ownership of metadata records to another user. Only users with *Administrator* and *UserAdmin* profiles can execute these services.

### Massive new owner (metadata.massive.newowner)

The *metadata.massive.newowner* service allows to change the owner of a group of metadata. This service requires a previous call to *metadata.select* service to select the metadata records to update.

Requires authentication: Yes

#### Request to metadata.select service

Parameters:

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

Example:

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

Mime-type:
application/xml

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

*Figure 5.17. Select metadata request example*

#### Response to metadata.select service

The response contains the number of metadata selected.

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

*Figure 5.18. Select metadata response example*

### Request to metadata.massive.newowner

Once the metadata records have been selected can be *metadata.massive.newowner* invoked with the next parameters:

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

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

Mime-type:
application/xml

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

*Figure 5.19. Transfer ownership request example*

### Response to metadata.massive.newowner

If request is executed succesfully HTTP 200 status code is returned. If request fails an HTTP status code error is returned and the response contains the XML document with the exception.

## Transfer ownership (xml.ownership.transfer)

The *xml.ownership.transfer* service can be used to transfer ownership and privileges of metadata owned by a user (in a group) to another user (in a group). This service should be used with data retrieved from previous invocations to the services xml.ownership.editors and xml.ownership.groups, described below.

Requires authentication: Yes

### Request

Parameters:

- *sourceUser*: (mandatory) Identifier of the user to transfer the ownership of her metadata
- *sourceGroup*: (mandatory) Identifier of source group of the metadata to transfer ownership
- *targetUser*: (mandatory) Identifier of the user to get the set the new metadata ownership
- *targetGroup*: (mandatory) Identifier of target group of the transferred ownership metadata

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)

```
Url:
http://localhost:8080/geonetwork/srv/en/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>
```

*Figure 5.20. Transfer ownership request example*

### Response

Here follows the structure of the response:

- *response*: This is the container for the response
  - *privileges*: Transferred privileges
  - *metadata*: Transferred metadata records

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

*Figure 5.21. Transfer ownership response example*

### 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. Returned 401 HTTP code
- *Missing parameter (error id: missing-parameter)*, when mandatory parameters are not provided
- *bad-parameter XXXX*, when a mandatory parameter is empty

## Retrieve metadata owners (xml.ownership.editors)

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

Requires authentication: Yes

### Request

Parameters:

- *None*

Example:

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

Mime-type:
application/xml

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

*Figure 5.22. Retrieve metadata owners request example*

## Response

Here follows the structure of the response:

- *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

```
<?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>
```

*Figure 5.23. Retrieve metadata editors response example*

## 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. Returned 401 HTTP code

## Retrieve groups/users allowed to transfer metadata ownership from a user (xml.ownership.groups)

The *xml.ownership.groups* service can be used to retrieve the groups/users to which can be transferred the metadata ownership/privilegies from the specified user.

### Request

Parameters:

- *id*: (mandatory) User identifier of the user to check to which groups/users can be transferred the ownership/privilegies of her metadata

```
Url:
http://localhost:8080/geonetwork/srv/en/xml.ownership.groups

Mime-type:
application/xml

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

*Figure 5.24. Retrieve ownership groups request example*

### Response

Here follows the structure of the response:

- *response*: This is the container for the response
  - *targetGroup*: Allowed target group to transfer ownership of user metadata (can be multiple *targetGroup* elements)
    - *id, name, description, email, referrer, label*: Group information
    - *editor*: Users of the group that own metadata (can be multiple *editor* elements)
      - *id,surname, name*: Metadata user owner information

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
  <targetGroup>
    <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>
    <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>
      <de>RWS</de>
      <fr>RWS</fr>
      <en>RWS</en>
      <es>RWS</es>
      <nl>RWS</nl>
    </label>
    <editor>
      <id>7</id>
      <surname />
      <name>Samantha</name>
    </editor>
  </targetGroup>

  ...

</response>
```

*Figure 5.25. Retrieve ownership groups response example*

### 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. Returned 401 HTTP code

## 5.4 Metadata editing

This services allow to maintaining the metadata in the catalog.

## Insert metadata (metadata.insert)

The *metadata.insert* service allows to create a new metadata record in the catalog.

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_" value to don't apply any stylesheet
- *validate*: Indicates if the metadata should be validated before inserting in the catalog. Values: on, off (default)

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>
```

*Figure 5.26. Insert metadata request example*

### Response

If request is executed succesfully HTTP 200 status code is returned. If request fails an HTTP status code error is returned and the response contains the XML document with the exception.

If validate parameter is set to "on" and the provided metadata is not valid confirming the xsd schema an exception report is returned.

```
<?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 not a valid value for 'dat
        <xpath>gmd:identificationInfo/gmd:MD_DataIdentification/gmd:citation/gmd:CI_Citation/gmd:date/gmd:CI_Date/g
      </error>
      <error>
        <message>ERROR(2) org.xml.sax.SAXParseException: cvc-type.3.1.3: The value '' of element 'gco:DateTime' is
        <xpath>gmd:identificationInfo/gmd:MD_DataIdentification/gmd:citation/gmd:CI_Citation/gmd:date/gmd:CI_Date/g
      </error>
      <error>
        <message>ERROR(3) org.xml.sax.SAXParseException: cvc-datatype-valid.1.2.1: '' is not a valid value for 'int
        <xpath>gmd:identificationInfo/gmd:MD_DataIdentification/gmd:spatialResolution/gmd:MD_Resolution/gmd:equival
      </error>
      <error>
        <message>ERROR(4) org.xml.sax.SAXParseException: cvc-type.3.1.3: The value '' of element 'gco:Integer' is n
        <xpath>gmd:identificationInfo/gmd:MD_DataIdentification/gmd:spatialResolution/gmd:MD_Resolution/gmd:equival
      </error>
    </xsderrors>
  </object>
  <request>
    <language>en</language>
    <service>metadata.insert</service>
  </request>
</error>
```

*Figure 5.27. Validation metadata report*

## 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. Returned 401 HTTP code
- *Missing parameter (error id: missing-parameter)*, when mandatory parameters are not provided. Returned 400 HTTP code
- *bad-parameter XXXX*, when a mandatory parameter is empty. Returned 400 HTTP code

- *ERROR: duplicate key violates unique constraint "metadata_uuid_key"*, if exists another metadata record in catalog with the same uuid of the metadata provided to insert

## Update metadata (metadata.update)

The metadata.update service allows to update the content of a metadata record in the catalog.

Requires authentication: Yes

### Request

Parameters:

- *id*: (mandatory) Identifier of the metadata to update
- *version*: (mandatory) This parameter is used to check if another user has updated the metadata after we retrieved it and before involking the update metadata service. **CHECK how to provide value to the user**
- *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.
- *title*: Metadata title (for templates)
- *data* (mandatory) Contains the metadata record

```
Url:
http://localhost:8080/geonetwork/srv/en/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>
```

*Figure 5.28. Update metadata request example*

### Response

If request is executed succesfully HTTP 200 status code is returned. If request fails an HTTP status code error is returned and the response contains the XML document with the exception.

### 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. Returned 401 HTTP code
- *Missing parameter (error id: missing-parameter)*, when mandatory parameters are not provided. Returned 400 HTTP code
- *bad-parameter XXXX*, when a mandatory parameter is empty. Returned 400 HTTP code
- *Concurrent update (error id: client)*, when the version number provided is different from actual version number for metatada. Returned 400 HTTP code

## Delete metadata (metadata.delete)

The *metadata.delete* service allows to remove a metadata record from the catalog. The metadata content is backup in MEF format by default in data\removed folder. This folder can be configured in geonetwork\WEB-INF\config.xml.

Requires authentication: Yes

### Request

Parameters:

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

Example:

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

Mime-type:
application/xml

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

*Figure 5.29. Delete metadata request example*

### Response

If request is executed succesfully HTTP 200 status code is returned. If request fails an HTTP status code error is returned and the response contains the XML document with the exception.

### 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. Returned 401 HTTP code
- *Metadata not found (error id: error)*, if the identifier provided don't correspond to an existing metadata. Returned 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:
  - 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

## 5.5 Harvesting services

### Introduction

This chapter provides a detailed explanation of the GeoNetwork's harvesting services. These services allow a complete control over the harvesting behaviour. They are used by the web interface and can be used by any other client.

### xml.harvesting.get

Retrieves information about one or all configured harvesting nodes.

#### Request

Called with no parameters returns all nodes. Example:

```
<request/>
```

Otherwise, an id parameter can be specified:

```
<request>
    <id>123</id>
</request>
```

#### Response

When called with no parameters the service provide its output inside a nodes container. You get as many node elements as are configured. Figure 5.30, "Example of an xml.harvesting.get response for a GeoNetwork node" shows an example of output.

```
<nodes>
    <node id="125" type="geonetwork">
        <site>
            <name>test 1</name>
            <UUID>0619cc50-708b-11da-8202-000d9335aaae</uuid>
            <host>localhost</host>
            <port>8080</port>
            <servlet>geonetwork</servlet>
            <account>
                <use>false</use>
                <username />
                <password />
            </account>
        </site>
        <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>
        <options>
            <every>90</every>
            <oneRunOnly>false</oneRunOnly>
            <status>inactive</status>
        </options>
        <info>
            <lastRun />
            <running>false</running>
        </info>
        <groupsCopyPolicy>
            <group name="all" policy="copy"/>
            <group name="mygroup" policy="createAndCopy"/>
        </groupsCopyPolicy>
        <categories>
            <category id="4"/>
        </categories>
    </node>
</nodes>
```

*Figure 5.30. Example of an xml.harvesting.get response for a GeoNetwork node*

If you specify an id, you get a response like that one in Figure 5.31, “Example of an xml.harvesting.get response for a WebDAV node” (for a WebDAV node).

```
<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>90</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>
```

*Figure 5.31. Example of an xml.harvesting.get response for a WebDAV node*

The node's structure has a common XML format, plus some additional information provided by the harvesting types. In the following structure, each element has a cardinality specified using the [x..y] notation, where x and y denote the minimum and the maximum values. The cardinality [1..1] is omitted for clarity.

- *node*: The root element. It has a mandatory *id* attribute that represents the internal identifier and a mandatory type attribute which indicates the harvesting type.
  - *site*: A container for site information.
    - *name (string)*: The node's name used to describe the harvesting.
    - *UUID (string)*: This is a system generated unique identifier associated to the harvesting node. This is used as the source field into the Metadata table to group all metadata from the remote node.
    - account: A container for account information.
      - *use (boolean)*: true means that the harvester will use the provided username and password to authenticate itself. The authentication mechanism depends on the harvesting type.
      - *username (string)*: Username on the remote node.

    - *password (string)*: Password on the remote node.

- options: A container for generic options.
    - *every (integer):* Harvesting interval in minutes.
    - *oneRunOnly (boolean)*: After the first run, the entry's status will be set to inactive.
    - *status (string)*: Indicates if the harvesting from this node is stopped (inactive) or if the harvester is waiting for the timeout (active).
- *privileges [0..1]*: A container for privileges that must be associated to the harvested metadata. This optional element is present only if the harvesting type supports it.
    - *group [0..n]*: A container for allowed operations associated to this group. It has the id attribute which value is the identifier of a GeoNetwork group.
        - *operation [0..n]*: Specifies an operation to associate to the containing group. It has a name attribute which value is one of the supported operation names. The only supported operations are: *view*, *dynamic*, *featured*.
- *categories [0..1]*: This is a container for categories to assign to each imported metadata. This optional element is present if the harvesting type supports it.
    - *category (integer) [0..n]*: Represents a local category and the id attribute is its local identifier.
- *info*: A container for general information.
    - *lastRun (string)*: The lastRun element will be filled as soon as the harvester starts harvesting from this entry. The value is the
    - *running (boolean)*: True if the harvester is currently running.
- *error*: This element will be present if the harvester encounters an error during harvesting.
    - *code (string)*: The error code, in string form.
    - *message (string)*: The description of the error.
    - *object (string)*: The object that caused the error (if any). This element can be present or not depending on the case.

### Errors

- ObjectNotFoundEx If the id parameter is provided but the node cannot be found.

## xml.harvesting.add

Create a new harvesting node. The node can be of any type supported by GeoNetwork (GeoNetwork node, web folder etc...). When a new node is created, its status is set to inactive. A call to the xml.harvesting.start service is required to start harvesting.

### Request

The service requires an XML tree with all information the client wants to add. In the following sections, default values are given in parenthesis (after the parameter's type) and are used when the parameter is omitted. If no default is provided, the parameter is mandatory. If the type is boolean, only the true and false strings are allowed.

All harvesting nodes share a common XML structure that must be honoured. Please, refer to the previous section for elements explanation. Each node type can add extra information to that structure. The common structure is here described:

- node: The root container. The type attribute is mandatory and must be one of the supported harvesting types.
  - site [0..1]
    - name (*string*, '')
    - account [0..1]
      - use (*boolean*, 'false')
      - username (*string*, '')
      - password (*string*, '')
  - options [0..1]
    - every (*integer*, '90')
    - oneRunOnly (*boolean*, 'false')
  - *privileges [0..1]*: Can be omitted but doing so the harvested metadata will not be visible. Please note that privileges are taken into account only if the harvesting type supports them.
    - *group [0..n]*: It must have the *id* attribute which value should be the identifier of a GeoNetwork group. If the id is not a valid group id, all contained operations will be discarded.
      - *operation [0..n]*: It must have a *name* attribute which value must be one of the supported operation names.
  - *categories [0..1]*: Please, note that categories will be assigned to metadata only if the harvesting type supports them.
    - *category (integer) [0..n]*: The mandatory id attribute is the category's local identifier.

Please note that even if clients can store empty values ('') for many parameters, before starting the harvesting entry those parameters should be properly set in order to avoid errors.

In the following sections, the XML structures described inherit from this one here so the common elements have been removed for clarity reasons (unless they are containers and contain new children).

Standard GeoNetwork harvesting

To create a node capable of harvesting from another GeoNetwork node, the following XML information should be provided:

- *node*: The type attribute is mandatory and must be GeoNetwork.
  - *site*
    - *host (string, '')*: The GeoNetwork node's host name or IP address.
    - *port (string, '80')*: The port to connect to.
    - *servlet (string, 'geonetwork')*: The servlet name chosen in the remote site.
  - *searches [0..1]*: A container for search parameters.
    - *search [0..n]*: A container for a single search on a siteID. You can specify 0 or more searches. If no search element is provided, an unconstrained search is performed.
      - *freeText (string, '')* : Free text to search. This and the following parameters are the same used during normal search using the web interface.
      - *title (string, '')*: Search the title field.
      - *abstract (string, '')* : Search the abstract field.
      - *keywords (string, '')* : Search the keywords fields.
      - *digital (boolean, 'false')*: Search for metadata in digital form.
      - *hardcopy (boolean, 'false')*: Search for metadata in printed form.
      - *source (string, '')*: One of the sources present on the remote node.
  - *groupsCopyPolicy [0..1]*: Container for copy policies of remote groups. This mechanism is used to retain remote metadata privileges.
    - *group*: There is one copy policy for each remote group. This element must have 2 mandatory attributes: *name* and *policy*. The name attribute is the remote group's name. If the remote group is renamed, it is not found anymore and the copy policy is skipped. The policy attribute represents the policy itself and can be: *copy*, *createAndCopy*, *copyToIntranet*. copy means that remote privileges are copied locally if there is locally a group with the same name as the *name* attribute. createAndCopy works like *copy* but the group is created locally if it does not exist. copyToIntranet works only for the remote group named all, which represents the public group. This policy copies privileges of the remote group named *all* to the local Intranet group. This is useful to restrict metadata access.

Figure 5.32, "Example of an xml.harvesting.add request for a GeoNetwork node" shows an example of an XML request to create a GeoNetwork node.

```
<node type="geonetwork">
    <site>
        <name>South Africa</name>
        <host>south.africa.org</host>
        <port>8080</port>
        <servlet>geonetwork</servlet>
        <account>
            <use>true</use>
            <username>admin</username>
            <password>admin</password>
        </account>
    </site>
    <searches>
        <search>
            <freeText />
            <title />
            <abstract />
            <keywords />
            <digital>true</digital>
            <hardcopy>false</hardcopy>
            <source>0619cc50-708b-11da-8202-000d9335906e</source>
        </search>
    </searches>
    <options>
        <every>90</every>
        <oneRunOnly>false</oneRunOnly>
    </options>
    <groupsCopyPolicy>
        <group name="all" policy="copy"/>
        <group name="mygroup" policy="createAndCopy"/>
    </groupsCopyPolicy>
    <categories>
        <category id="4"/>
    </categories>
</node>
```

*Figure 5.32. Example of an xml.harvesting.add request for a GeoNetwork node*

WebDAV harvesting

To create a web DAV node, the following XML information should be provided.

- *node*: The type attribute is mandatory and must be WebDAV.
  - site
    - *url (string, '')*: The URL to harvest from. If provided, must be a valid URL starting with 'HTTP://'.
    - *icon (string, 'default.gif')* : Icon file used to represent this node in the search results. The icon must be present into the images/harvesting folder.
  - options
    - *recurse (boolean, 'false')*: When true, folders are scanned recursively to find metadata.
    - *validate (boolean, 'false')*: When true, GeoNetwork will validate every metadata against its schema. If the metadata is not valid, it will not be imported.

This type supports both privileges and categories assignment.

Figure 5.33, “Example of an xml.harvesting.add request for a WebDAV node” shows an example of an XML request to create a web DAV entry.

```
<node type="webdav">
    <site>
        <name>Asia remote node</name>
        <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>90</every>
        <oneRunOnly>false</oneRunOnly>
        <recurse>false</recurse>
        <validate>true</validate>
    </options>
    <privileges>
        <group id="0">
            <operation name="view" />
        </group>
        <group id="14">
            <operation name="features" />
        </group>
    </privileges>
    <categories>
        <category id="4"/>
    </categories>
</node>
```

*Figure 5.33. Example of an xml.harvesting.add request for a WebDAV node*

CSW harvesting

To create a node to harvest from a CSW capable server, the following XML information should be provided:

- *node*: The type attribute is mandatory and must be csw.
  - *site*
    - *capabilitiesUrl (string)*: URL of the capabilities file that will be used to retrieve the operations address.
    - *icon (string, 'default.gif')* : Icon file used to represent this node in the search results. The icon must be present into the images/harvesting folder.
  - *searches [0..1]*
    - *search [0..n]*: Contains search parameters. If this element is missing, an unconstrained search will be performed.
      - *freeText (string, '')* : Search the entire metadata.
      - *title (string, '')*: Search the dc:title queryable.
      - *abstract (string, '')*: Search the dc:abstract queryable.

- *subject (string, ”)*: Search the dc:subject queryable.

This type supports both privileges and categories assignment.

Figure 5.34, “Example of an xml.harvesting.add request for a CSW node” shows an example of an XML request to create a CSW entry.

```
<node type="csw">
    <site>
        <name>Minos CSW server</name>
        <capabilitiesUrl>http://www.minos.org/csw?request=GetCapabilities
            &amp;amp;service=CSW&amp;amp;acceptVersions=2.0.1</capabilitiesUrl>
        <icon>default.gif</icon>
        <account>
            <use>true</use>
            <username>admin</username>
            <password>admin</password>
        </account>
    </site>
    <options>
        <every>90</every>
        <oneRunOnly>false</oneRunOnly>
        <recurse>false</recurse>
        <validate>true</validate>
    </options>
    <privileges>
        <group id="0">
            <operation name="view" />
        </group>
        <group id="14">
            <operation name="features" />
        </group>
    </privileges>
    <categories>
        <category id="4"/>
    </categories>
</node>
```

*Figure 5.34. Example of an xml.harvesting.add request for a CSW node*

## Response

The service’s response is the output of the `xml.harvesting.get` service of the newly created node.

## Summary

The following table:

*Table 5.1. Summary of features of the supported harvesting types*

| Harvesting type | Authentication | Privileges ? | Categories ? |
|---|---|---|---|
| GeoNetwork | native | through policies | yes |
| Web DAV | HTTP digest | yes | yes |
| CSW | HTTP Basic | yes | yes |

## xml.harvesting.update

This service is responsible for changing the node's parameters. A typical request has a node root element and must include the id attribute:

```
     <node id="24">
    ...
</node>
```

The body of the node element depends on the node's type. The update policy is this:

- If an element is specified, the associated parameter is updated.
- If an element is not specified, the associated parameter will not be changed.

So, you need to specify only the elements you want to change. However, there are some exceptions:

1. *privileges*: If this element is omitted, privileges will not be changed. If specified, new privileges will replace the old ones.
2. *categories*: Like the previous one.
3. *searches*: Some harvesting types support multiple searches on the same remote note. When supported, the updated behaviour should be like the previous ones.

Note that you cannot change the type of an node once it has been created.

### Request

The request is the same as that used to add an entry. Only the id attribute is mandatory.

### Response

The response is the same as the xml.harvesting.get called on the updated entry.

## xml.harvesting.remove/start/stop/run

These services are put together because they share a common request interface. Their purpose is obviously to remove, start, stop or run a harvesting node. In detail:

1. *start*: When created, a node is in the inactive state. This operation makes it active, that is the countdown is started and the harvesting will be performed at the timeout.
2. *stop*: Makes a node inactive. Inactive nodes are never harvested.
3. *run*: Just 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>
```

If the request is empty, nothing is done.

### Response

The same as 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:

```
<request>
    <id status="ok">123</id>
    <id status="not-found">456</id>
    <id status="inactive">789</id>
</request>
```

Table 5.2, “Summary of status values” summarises, for each service, the possible status values.

*Table 5.2. Summary of status values*

| Status value | remove | start | stop | run |
|---|---|---|---|---|
| ok | + | + | + | + |
| not-found | + | + | + | + |
| inactive | - | - | - | + |
| already-inactive | - | - | + | - |
| already-active | - | + | - | - |
| already-running | - | - | - | + |

## 5.6 MEF services

### Introduction

This chapter describes the services related to the Metadata Exchange Format. These services allow to import/export metadata using the MEF format.

### mef.export

As the name suggests, this service exports a GeoNetwork’s metadata using the MEF file format.

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’s temporary directory to build the MEF file. With full exports of big data maybe it is necessary to change this directory. In this case, use the Java’s -D command line option to set the new directory before running GeoNetwork (if you use Jetty, simply change the script into the bin directory).

### Request

This service accepts requests in GET/POST and XML form. The input parameters are:

*UUID* the universal unique identifier of the metadata

*format* which format to use. Can be one of: `simple, partial, full`.

*skipUuid* If provided, tells the exporter to not export the metadata's UUID. Without the UUID (which is a unique key inside the database) the metadata can be imported over and over again. Can be one of: `true, false`. The default value is false.

### Response

The service's response is a MEF file with these characteristics:

- the name of the file is the metadata's UUID
- the extension of the file is mef

## mef.import

This service is reserved to administrators and is used to import a metadata provided in the MEF format.

### Request

The service accepts a `multipart/form-data` POST request with a single *mefFile* parameter that must contain the MEF information.

### Response

If all goes well, the service returns an OK element containing the local id of the created metadata. Example:

```
<ok>123</ok>
```

## Metadata ownership

Version 1.0 of the MEF format does not take into account the metadata owner (the creator) and the group owner. This implies that this information is not contained into the MEF file. During import, the user that is performing this operation will become the metadata owner and the group owner will be set to null.

# 6. System configuration

## 6.1 System configuration

### Introduction

The GeoNetwork's configuration is made up of a set of parameters that can be changed to accommodate any installation need. These parameters are subdivided into 2 groups:

- parameters that can be easily changed through a web interface.
- parameters not accessible from a web interface and that must be changed when the system is not running.

The first group of parameters can be queried or changed through 2 services: xml.config.get and xml.config.update. The second group of parameters can be changed using the GAST tool.

### xml.config.get

This service returns the system configuration's parameters.

#### Request

No parameters are needed.

#### 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's name.
  - *organisation*: Site's organisation name.
- *server*: A container for server information.
  - *host*: Name of the host from which the site is reached.
  - *port*: Port number of the previous host.
- *Intranet*: Information about the Intranet of the organisation.
  - *network*: IP address that specifies the network.
  - *netmask*: netmask of the network.
- *z3950*: Configuration about Z39.50 protocol.
  - *enable*: true means that the server component is running.
  - *port*: Port number to use to listen for incoming Z39.50 requests.

- *proxy*: Proxy configuration
  - *use*: true means that the proxy is used when connecting to external nodes.
  - *host*: Proxy's server host.
  - *port*: Proxy's server port.
  - *username*: Proxy's credentials.
  - *password*: Proxy's credentials.
- *feedback*: A container for feedback information
  - *email*: Administrator's email address
  - *mailServer*: Email server to use to send feedback
    - *host*: Email's host address
    - *port*: Email's port to use in host address
- *removedMetadata*: A container for removed metadata information
  - *dir*: Folder used to store removed metadata in MEF format
- *ldap*: A container for LDAP parameters
  - *use*:
  - *host*:
  - *port*:
  - *defaultProfile*:
  - *login*:
    - *userDN*:
    - *password*:
  - *distinguishedNames*:
    - *base*:
    - *users*:
  - *userAttribs*:
    - *name*:
    - *password*:
    - *profile*:

Figure 6.1, "Example of xml.config.get response" shows an example of `xml.config.get` response

```
<config>
    <site>
        <name>dummy</name>
        <organisation>dummy</organization>
    </site>
    <server>
        <host>localhost</host>
        <port>8080</port>
    </server>
    <Intranet>
        <network>127.0.0.1</network>
        <netmask>255.255.255.0</netmask>
    </intranet>
    <z3950>
        <enable>true</enable>
        <port>2100</port>
    </z3950>
    <proxy>
        <use>false</use>
        <host/>
        <port/>
        <username>proxyuser</username>
        <password>proxypass</password>
    </proxy>
    <feedback>
        <email/>
        <mailServer>
            <host/>
            <port>25</port>
        </mailServer>
    </feedback>
    <removedMetadata>
        <dir>WEB-INF/removed</dir>
    </removedMetadata>
    <ldap>
        <use>false</use>
        <host />
        <port />
        <defaultProfile>RegisteredUser</defaultProfile>
        <login>
            <userDN>cn=Manager</userDN>
            <password />
        </login>
        <distinguishedNames>
            <base>dc=fao,dc=org</base>
            <users>ou=people</users>
        </distinguishedNames>
        <userAttribs>
            <name>cn</name>
            <password>userPassword</password>
            <profile>profile</profile>
        </userAttribs>
    </ldap>
</config>
```

*Figure 6.1. Example of xml.config.get response*

## xml.config.update

This service is used to update the system’s information and so it is restricted to administrators.

### Request

The request format must have the same structure returned by the `xml.config.get` service and can contain only elements that the caller wants to be updated. If an element is not included, it will not be updated. However, when included some elements require mandatory information (i.e. the value cannot be empty). Please, refer to Table 6.1, “Mandatory and optional parameters for the xml.config.update service”.

*Table 6.1. Mandatory and optional parameters for the xml.config.update service*

| Parameter | Type | Mandatory |
|---|---|---|
| site/name | string | yes |
| site/organization | string | - |
| server/host | string | yes |
| server/port | integer | - |
| intranet/network | string | yes |
| intranet/netmask | string | yes |
| z3950/enable | bool | yes |
| z3950/port | integer | - |
| proxy/use | bool | yes |
| proxy/host | string | - |
| proxy/port | integer | - |
| proxy/username | string | - |
| proxy/password | string | - |
| feedback/email | string | - |
| feedback/mailServer/host | string | - |
| feedback/mailServer/port | integer | - |
| removedMetadata/dir | string | yes |
| ldap/use | bool | yes |
| ldap/host | string | - |
| ldap/port | integer | - |
| ldap/defaultProfile | string | yes |
| ldap/login/userDN | string | yes |
| ldap/login/password | string | - |
| ldap/distinguishedNames/base | string | yes |
| ldap/distinguishedNames/users | string | yes |
| ldap/userAttribs/name | string | yes |
| ldap/userAttribs/password | string | yes |
| ldap/userAttribs/profile | string | - |

### Response

On success, the service returns a response element with the OK text. Example:

```
<response>ok</response>
```

Otherwise a proper error element is returned.

# Part II. Java development with XML Services

This part gives some insights on developing Java client applications that use GeoNetwork opensource XML services described in previous section.

An example application it's provided to show how develop a client application using login service in GeoNetwork opensource and some of the XML services avalaible that require authentication.

# 7. 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.

## 7.1 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>
```

## 7.2 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="374" 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" />
    <at class="javax.servlet.http.HttpServlet"
        file="HttpServlet.java" line="820" method="service" />
    <at class="org.mortbay.jetty.servlet.ServletHolder"
        file="ServletHolder.java" line="502" method="handle" />
    <at class="org.mortbay.jetty.servlet.ServletHandler"
        file="ServletHandler.java" line="363" method="handle" />
    <at class="org.mortbay.jetty.security.SecurityHandler"
```

```
        file="SecurityHandler.java" line="216" method="handle" />
    <at class="org.mortbay.jetty.servlet.SessionHandler"
        file="SessionHandler.java" line="181" method="handle" />
  </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
```

## 7.3 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" line="1548"
    method="receiveErrorResponse" />
    <at class="org.postgresql.core.v3.QueryExecutorImpl"
    file="QueryExecutorImpl.java" line="1316" method="processResults" />
    <at class="org.postgresql.core.v3.QueryExecutorImpl"
    file="QueryExecutorImpl.java" line="191" method="execute" />
    <at class="org.postgresql.jdbc2.AbstractJdbc2Statement"
    file="AbstractJdbc2Statement.java" line="452" method="execute" />
    <at class="org.postgresql.jdbc2.AbstractJdbc2Statement"
    file="AbstractJdbc2Statement.java" line="351"
    method="executeWithFlags" />
    <at class="org.postgresql.jdbc2.AbstractJdbc2Statement"
```

```
    file="AbstractJdbc2Statement.java" line="305"
    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" 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" />
  </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"
```

# Part III. CSW Service

This part gives some insights of GeoNetwork opensource CSW service implementation.

# 8. CSW service

## 8.1 Introduction

GeoNetwork opensource catalog publishes metadata using 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 it's compliant with 2.0.2 version of specification supporting the next CSW operations:

- GetCapabilities
- DescribeRecord
- GetRecordById
- GetRecords
- Transaction

In this chapter a brief description of the different operations supported in GeoNetwork and 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 before chapter for XML services.

## 8.2 CSW operations

The GeoNetwork opensource catalog CSW service operations are accesible thought the url:

*http://localhost:8080/geonetwork/srv/en/csw*

The CSW operations can be accesed using POST, GET methods and SOAP encoding.

### 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&acceptVersions=2.0.2&acceptFormats=application%2Fxml
```

POST request

```
Url:
```

```
http://localhost:8080/geonetwork/srv/en/csw

Mime-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

Mime-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=2.0.2&outputFormat=application%2Fxml
&schemaLanguage=http%3A%2F%2Fwww.w3.org%2FXML%2FSchema
&namespace=csw%3Ahttp%3A%2F%2Fwww.opengis.net%2Fcat%2Fcsw%2F2.0.2
```

POST request

```
Url:
http://localhost:8080/geonetwork/srv/en/csw
```

```
Mime-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" version="2.0.2" outputFormat="application/xml"
schemaLanguage="http://www.w3.org/XML/Schema" />
```

SOAP request

```
Url:
http://localhost:8080/geonetwork/srv/en/csw

Mime-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" version="2.0.2" outputFormat="application/xml"
    schemaLanguage="http://www.w3.org/XML/Schema" />
  </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=2.0.2&elementSetName=full
&id=5df54bf0-3a7d-44bf-9abf-84d772da8df1
```

POST request

```
Url:
http://localhost:8080/geonetwork/srv/en/csw

Mime-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" version="2.0.2">
  <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

Mime-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" version="2.0.2">
      <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)

```
http://localhost:8080/geonetwork/srv/en/csw?request=GetRecords
&service=CSW&version=2.0.2
&namespace=xmlns%28csw%3Dhttp%3A%2F%2Fwww.opengis.net%2Fcat%2Fcsw
%2F2.0.2%29%2Cxmlns%28gmd%3Dhttp%3A%2F%2Fwww.isotc211.org%2F2005
%2Fgmd%29&constraint=AnyText+like+%25africa%25
&constraintLanguage=CQL_TEXT
&constraint_language_version=1.1.0&typeNames=csw%3ARecord
```

POST request

```
Url:
http://localhost:8080/geonetwork/srv/en/csw

Mime-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="2.0.2">
  <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

Mime-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" version="2.0.2">
      <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>
  </env:Body>
</env:Envelope>
```

## 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

Mime-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" service="CSW">
  <csw:Insert>
    <gmd:MD_Metadata xmlns:gmd="http://www.isotc211.org/2005/gmd"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xmlns:gml="http://www.opengis.net/gml"
```

```
                ....>

      ...

    </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

Mime-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" service="CSW">
  <csw:Update>

    <gmd:MD_Metadata xmlns:gmd="http://www.isotc211.org/2005/gmd"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xmlns:gml="http://www.opengis.net/gml"
                ....>

      ...

    </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

Mime-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.opengis.net/ogc"
  version="2.0.2" service="CSW">
  <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.w3.org/2001/XMLSchema-instance" version="1.0.0"
  xsi:schemaLocation=
  "http://www.opengis.net/ows http://schemas.opengis.net/ows/1.0.0/owsExceptionReport.xsd">
  <ows:Exception exceptionCode="NoApplicableCode">
    <ows:ExceptionText>Cannot process transaction: User not authenticated.</ows:ExceptionText>
  </ows:Exception>
</ows:ExceptionReport>
```

# Glossary

| | |
|---|---|
| CSW | Catalog Service for the Web. The OGC Catalog Service defines common interfaces to discover, browse, and query metadata about data, services, and other potential resources.<br>See Also OGC. |
| ISO | International Standards Organisation is an international-standard-setting body composed of representatives from various national standards organizations. http://www.iso.org.<br>See Also ISO TC211. |
| ISO TC211 | ISO/TC 211 is a standard technical committee formed within ISO, tasked with covering the areas of digital geographic information (such as used by geographic information systems) and geomatics. It is responsible for preparation of a series of International Standards and Technical Specifications numbered in the range starting at 19101. |
| GeoNetwork | GeoNetwork opensource is a standards based, Free and Open Source catalog application to manage spatially referenced resources through the web.http://geonetwork-opensource.org |
| GN | See GeoNetwork. |
| XML | Extensible Markup Language is a general-purpose specification for creating custom markup languages. |
| XSD | XML Schema, published as a W3C recommendation in May 2001, is one of several XML schema languages. http://en.wikipedia.org/wiki/XSD |
| DB (or DBMS) | A database management system (DBMS) is computer software that manages databases. DBMSes may use any of a variety of database models, such as the network model or relational model. In large systems, a DBMS allows users and other software to store and retrieve data in a structured way. |
| SOA | Service Oriented Architecture provides methods for systems development and integration where systems package functionality as interoperable services. A SOA infrastructure allows different applications to exchange data with one another. |
| FGDC | The Federal Geographic Data Committee (FGDC) is an interagency committee that promotes the coordinated development, use, sharing, and dissemination of geospatial data on a national basis in the USA. See http://www.fgdc.gov |
| SOAP | Simple Object Access Protocol is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks. |
| OGC | Open Geospatial Consortium. A standards organization for geospatial information systems http://www.opengeospatial.org |

| | |
|---|---|
| OSGeo | The Open Source Geospatial Foundation (OSGeo), is a non-profit non-governmental organization whose mission is to support and promote the collaborative development of open geospatial technologies and data. http://www.osgeo.org |
| FAO | Food and Agriculture Organisation of the United Nations is a specialised agency of the United Nations that leads international efforts to defeat hunger. http://www.fao.org |
| WFP | World Food Programme of the United Nations is the food aid branch of the United Nations, and the world's largest humanitarian organization. http://www.wfp.org |
| UNEP | The UN Environment Programme (UNEP) coordinates United Nations environmental activities, assisting developing countries in implementing environmentally sound policies and encourages sustainable development through sound environmental practices. http://www.unep.org |
| OCHA | United Nations Office for the Coordination of Humanitarian Affairs is designed to strengthen the UN's response to complex emergencies and natural disasters. http://ochaonline.un.org/ |
| URL | A Uniform Resource Locator specifies where an identified resource is available and the mechanism for retrieving it. |
| GAST | GeoNetwork Administrator Survival Tool. A desktop application that allows administrators of a GeoNetwork catalog to perform a range of admin operations. |
| WebDAV | Web-based Distributed Authoring and Versioning. WebDAV is a set of extensions to the Hypertext Transfer Protocol (HTTP) that allows users to edit and manage files collaboratively on remote World Wide Web servers. |
| OAI-PMH | Open Archive Initiative Protocol for Metadata Harvesting. It is a protocol developed by the Open Archives Initiative. It is used to harvest (or collect) the metadata descriptions of the records in an archive so that services can be built using metadata from many archives. |
| WMS | Web Map Service is a standard protocol for serving georeferenced map images over the Internet that are generated by a map server using data from a GIS database. The specification was developed and first published by the Open Geospatial Consortium in 1999.<br>See Also OGC. |
| WFS | Web Feature Service provides an interface allowing requests for geographical features across the web using platform-independent calls. One can think of geographical features as the "source code" behind a map.<br>See Also OGC. |

| | |
|---|---|
| WCS | Web Coverage Service provides an interface allowing requests for geographical coverages across the web using platform-independent calls. The coverages are objects (or images) in a geographical area<br>See Also OGC. |
| WPS | Web Processing Service is designed to standardize the way that GIS calculations are made available to the Internet. WPS can describe any calculation (i.e. process) including all of its inputs and outputs, and trigger its execution as a Web Service.<br>See Also OGC. |
| UUID | A Universally Unique Identifier (UUID) is an identifier standard used in software construction, standardized by the Open Software Foundation (OSF) as part of the Distributed Computing Environment (DCE). |
| MAC address | Media Access Control address (MAC address) is a unique identifier assigned to most network adapters or network interface cards (NICs) by the manufacturer for identification, and used in the Media Access Control protocol sublayer. See also MAC address[1] on Wikipedia |
| MEF | Metadata Exchange Format. An export format developed by the GeoNetwork community. More details can be found in this manual in Chapter Metadata Exchange Format. |
| SKOS | The Simple Knowledge Organisation Systems (SKOS) http://www.w3.org/2004/02/skos/ is an area of work developing specifications and standards to support the use of knowledge organisation systems (KOS) such as thesauri, classification schemes. |
| Z39.50 protocol | Z39.50 is a client-server protocol for searching and retrieving information from remote computer databases. It is covered by ANSI/NISO standard Z39.50, and ISO standard 23950. The standard's maintenance agency is the Library of Congress. |
| SMTP | Simple Mail Transfer Protocol is an Internet standard for electronic mail (e-mail) transmission across Internet Protocol (IP) networks. |
| LDAP | Lightweight Directory Access Protocol is an application protocol for querying and modifying directory services running over TCP/IP. |
| Shibboleth | The Shibboleth System is a standards based, open source software package for web single sign-on across or within organisational boundaries. It allows sites to make informed authorisation decisions for individual access of protected online resources in a privacy-preserving manner. |
| DC | The Dublin Core metadata element set is a standard for cross-domain information resource description. It provides a simple and standardised set of conventions for describing things online in ways that make them easier to find. |
| FOSS | Free and Open Source Software, also F/OSS, FOSS, or FLOSS (free/libre/open source software) is software which is liberally licensed to |

grant the right of users to study, change, and improve its design through the availability of its source code. http://en.wikipedia.org/wiki/FOSS