GeoNetwork's Server Reference

Author: **ANDREA CARBONI**

Revision 9 (28-Aug-2007)

Introduction

This document gives some insights about the internal structure of GeoNetwork opensource. It describes some basic operations, like compiling and running the software, protocols used, a description of XML services and the Settings structure.

Contents

Part I

Basic Operations

Chapter 1

Compiling and running GeoNetwork opensource

1.1 System Requirements

GeoNetwork is a Java application that runs as a servlet so the Java Runtime Environment (JRE) must be installed in order to run it. You can get the JRE from the following address **http://java.sun.com** and you have to download the Java 5 Standard Edition (SE). GeoNetwork won't run with Java 1.4 and Java 6 has some problems with it so we recommend to use Java 5. Being written in Java, GeoNetwork can run on any platform that supports Java, so it can run on Windows, Linux and Mac OSX. For the latter one, make sure to use version 10.4 (Tiger) or newer. Version 10.3 (Panther) has only Java 1.4 so it cannot run GeoNetwork.

Next, you need a servlet container. GeoNetwork comes with an embedded one (Jetty) which is fast and well suited for most applications. If you need a stronger one, you can install Tomcat from the Apache Software Foundation (**http://tomcat.apache.org**). It provides load balance, fault tolerance and other corporate needed stuff. If you work for an organization, it is probable that you already have it up and running. The tested version is 5.5 but GeoNetwork should work with all other versions.

Regarding storage, you need a Database Management System (DBMS) like Oracle, MySQL, PostgreSQL and so on. GeoNetwork comes with an embedded one (McKoi) which is used by default during installation. This DBMS can be used for small or desktop installations, where the speed is not an issue. You can use this DBMS for several thousands of metadata. If you manage more than 10.000 metadata it is better to use a professional, stand alone DBMS. In this case, using a separate DBMS also frees up some memory for the application.

GeoNetwork does not require a strong machine to run. A good performance can be obtained even with 128 Mb of RAM. The suggested amount is 512 Mb. For the hard disk space, you have to consider the space required for the application itself (about 40 Mb) and the space required for data maps, which can require 50 Gb or more. A simple disk of 250 Gb should be ok. Maybe you can choose a fast one to reduce backup time but GeoNetwork itself does not speed up on a faster disk. You also need some space for the search index which is located in **web/WEB-INF/lucene**. Even with a lot of metadata the index is small so usually 10-20 Mb of space is enough.

1.2 Running the software

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

Tomcat You can use the manager web application to start/stop GeoNetwork. You can also use the **startup.*** and **shutdown.*** scripts located into Tomcat's **bin** folder (.* means .sh or .bat depending on your OS) but this way you restart all applications you are running, not only GeoNetwork. After installation and before running GeoNetwork you must link it to Tomcat.

Jetty If you use the provided container you can use the scripts into GeoNetwork's **bin** folder. The scripts are **start-geonetwork.*** and **stop-geonetwork.*** and you must be inside the **bin** folder to run them. You can use these scripts just after installation.

1.3 Compiling GeoNetwork

To compile GeoNetwork you first need to install the source code during installation. If you do so, you get a **build.xml** script and a **src** folder with the full source.

You also need the **Ant** tool to run the build script. You can download Ant from **http://ant.apache.org**. Version 1.6.5 works but any other recent version should be ok. Once installed, you should have the **ant** command in your path (on Windows systems, you have to open a shell to check).

When all is in place, go inside the GeoNetwork's root folder (the one where the **build.xml** file is located) and issue the **ant** command. You should see an output like this one:

```
gemini:/geonetwork/trunk# ant
Buildfile: build.xml
compile:
[delete] Deleting: /geonetwork/trunk/web/WEB-INF/lib/geonetwork.jar
[delete] Deleting: /geonetwork/trunk/csw/lib/csw-client.jar
[delete] Deleting: /geonetwork/trunk/csw/lib/csw-common.jar
[delete] Deleting: /geonetwork/trunk/gast/gast.jar
 [mkdir] Created dir: /geonetwork/trunk/.build
 [javac] Compiling 267 source files to /geonetwork/trunk/.build
 [javac] Note: Some input files use or override a deprecated API.
 [javac] Note: Recompile with -Xlint:deprecation for details.
 [javac] Note: Some input files use unchecked or unsafe operations.
 [javac] Note: Recompile with -Xlint:unchecked for details.
  [copy] Copying 1 file to /geonetwork/trunk/.build
   [jar] Building jar: /geonetwork/trunk/web/WEB-INF/lib/geonetwork.jar
   [jar] Building jar: /geonetwork/trunk/csw/lib/csw-client.jar
   [jar] Building jar: /geonetwork/trunk/csw/lib/csw-common.jar
   [jar] Building jar: /geonetwork/trunk/gast/gast.jar
[delete] Deleting directory /geonetwork/trunk/.build
BUILD SUCCESSFUL
Total time: 9 seconds
gemini:/geonetwork/trunk#
```

The compilation phase, if it has success, puts all jars into the proper place (most of them will be copied into **web/geonetwork/WEB-INF/lib** and **web/intermap/WEB-INF/lib**). After this phase, simply restart GeoNetwork to see the effects.

Part II

Implementation guidelines

Chapter 2

Harvesting

2.1 Structure

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

Javascript code

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

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

Java code

The harvesting package is located in src/org/fao/geonet/kernel/harvest. Here too, there is one subfolder for each harvesting type. The most important classes for the implementor are:

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

All others are small utility classes used by harvesting types.

XSL stylesheets

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

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

In each subfolder, there are usually 4 files:

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

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

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

The last file to take into consideration contains all localized strings and is located at web/geonetwork/loc/XX/xml (where XX refers to a language code). This file is used by both Javascript code and the server.

2.2 Data storage

Harvesting nodes are stored inside the **Settings** table. Further useful information can be found in chapters 6 and 11.

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

2.3 Guidelines

To add a new harvesting type, follows these steps:

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

Here follows a list of general notes to follow when adding a new harvesting type:

- Every harvesting node (not type) must generate its UUID. This uuid is used to remove metadata when the harvesting node is removed and to check if a metadata (which has another uuid) has been already harvested by another node.
- If a harvesting type supports multiple searches on a remote site, these must be done sequentially and results merged.
- Every harvesting type must save in the folder images/logos a GIF image whose name is the node's uuid. This image must be deleted when the harvesting node is removed. This is necessary to propagate harvesting information to other GeoNetwork nodes.

- When a harvesting node is removed, all collected metadata must be removed too.
- During harvesting, take in mind that a metadata could have been removed just after being added to the result list. In this case the metadata should be skipped and no exception raised.
- The only settable privileges are: view, dynamic, featured. It does not make sense to use the others.
- If a node raises an exception during harvesting, that node will be deactivated.
- If a metadata already exists (its uuid exists) but belong to another node, it must not be updated even if it has been changed. This way the harvesting will not conflict with the other one. As a side effect, this prevent locally created metadata from being changed.
- The harvesting engine does not store results on disk so they will get lost when the server will be restarted.
- When some harvesting parameters are changed, the new harvesting type must use them during the next harvesting without requiring to reboot the server.

Part III

File formats

Chapter 3

Metadata Exchange Format v1.0

3.1 Introduction

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

This format has been designed with these needs in mind:

- Export a metadata record for backup purposes
- Import a metadata record from a previous backup
- Import a metadata record from a different GeoNetwork version to allow a smooth migration from one version to another.

All these operations regard the metadata and its related data as well.

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

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

3.2 File format

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

- **metadata.xml** : this file contains the metadata itself, in XML format. The text encoding of the metadata is that one specified into the XML declaration.

- **info.xml** : this is a special XML file which contains information related to the metadata but that cannot be stored into it. Examples of such information are the creation date, the last change date, privileges on the metadata and so on. Now this information is related to the GeoNetwork's architecture.
- **public** : this is a directory used to store the metadata thumbnails and other public files. There are no restrictions on the images' format but it is strongly recommended to use the portable network graphics (PNG), the JPEG or the GIF formats.
- **private** : this is a directory used to store all data (maps, shape files etc...) associated to the metadata. Files in this directory are *private* in the sense that an authorization is required to access them. There are no restrictions on the file types that can be stored into this directory.

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

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

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

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

3.3 The info.xml file

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

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

The root element must have the following children:

- **general** : a container for general information. It must have the following children:
  - **uuid** : this is the universally unique identifier assigned to the metadata and must be a valid uuid. This element is optional and, when omitted, the reader should generate one. A metadata without a uuid can be imported several times into the same system without breaking uniqueness constraints. When missing, the reader should also generate the **siteId** value.

  - **createDate** : This date indicates when the metadata was created.
  - **changeDate** : This date keeps track of the most recent change to the metadata.
  - **siteId** : This is an uuid that identifies the actor that created the metadata and must be a valid uuid. When the **uuid** element is missing, this element should be missing too. If present, it will be ignored.
  - **siteName** : This is a human readable name for the actor that created the metadata. It must be present only if the **siteId** is present.
  - **schema** : Indicates the metadata's schema. The value can be assigned as will but if the schema is one of those describe below, that value must be used:
    * **dublin-core** : A metadata in the dublin core format as described in http://dublincore.org
    * **fgdc-std** : A metadata in the Federal Geographic Data Committee.
    * **iso19115** : A metadata in the ISO 19115 format
    * **iso19139** : A metadata in the ISO 19115/2003 format for which the ISO19139 is the XML encoding.
  - **format** : Indicates the MEF export format. The element's value must belong to the following set: { *simple, partial*, *full* }.
  - **localId** : This is an optional element. If present, indicates the id used locally by the sourceId actor to store the metadata. Its purpose is just to allow the reuse of the same local id when reimporting a metadata.
  - **isTemplate** : A boolean field that indicates if this metadata is a template used to create new ones. There is no real distinction between a real metadata and a template but some actors use it to allow fast metadata creation. The value must be: { *true*, *false* }.

- **categories** : a container for categories associated to this metadata. A category is just a name, like 'audio-video' that classifies the metadata to allow an easy search. Each category is specified by a **category** element which must have a **name** attribute. This attribute is used to store the category's name. If there are no categories, the **categories** element will be empty.

- **privileges** : a container for privileges associated to this metadata. Privileges are operations that a group (which represents a set of users) can do on a metadata and are specified by a set of **group** elements. Each one of these, has a mandatory **name** attribute to store the group's name and a set of **operation** elements used to store the operations allowed on the metadata. Each **operation** element must have a **name** attribute which value must belong to the following set: { *view*, *download*, *notify*, *dynamic*, *featured* }. If there are no groups or the actor does not have the concept of group, the **privileges** element will be empty. A **group** element without any **operation** element must be ignored by readers.

- **public** : All metadata thumbnails (and any other public file) must be listed here. This container contains a **file** element for each file. Mandatory attributes of this element are **name**, which represents the file's name and **changeDate**, which contains the date of the latest change to the file. The **public** element is optional but, if present, must contain all the files present in the metadata's **public** directory and any reader that imports these files must set the latest change date on these using the provided ones. The purpose of this element is to provide more information in the case the MEF format is used for metadata harvesting.

- **private** : This element has the same purpose and structure of the **public** element but is related to maps and all other private files.

```
<info version="1.0">
   <general>
      <uuid>0619abc0-708b-eeda-8202-000d98959033</uuid>
      <createDate>2006-12-11T10:33:21</createDate>
      <changeDate>2006-12-14T08:44:43</changeDate>
      <siteId>0619cc50-708b-11da-8202-000d9335906e</siteId>
      <siteName>FAO main site</siteName>
      <schema>iso19139</schema>
      <format>full</format>
      <localId>204</localId>
      <isTemplate>false</isTemplate>
   </general>
   <categories>
      <category name="maps"/>
      <category name="datasets"/>
   </categories>
   <privileges>
      <group name="editors">
         <operation name="view"/>
         <operation name="download"/>
      </group>
   </privileges>
   <public>
      <file name="small.png" changeDate="2006-10-07T13:44:32"/>
      <file name="large.png" changeDate="2006-11-11T09:33:21"/>
   </public>
   <private>
      <file name="map.zip" changeDate="2006-11-12T13:23:01"/>
   </private>
</info>
```

Figure 3.1: Example of info file

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

Figure 3.1 shows an example of info file.

Date format

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

Part IV

XML Services

Chapter 4

Calling specifications

4.1 Calling 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.

4.1.1 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 4.1 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 specialized 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

```
GET /geonetwork/srv/en/main.search?hitsPerPage=10&any=
<request>
   <hitsPerPage>10</hitsPerPage>
   <any />
</request>
```

Figure 4.1: A GET request to a XML service and its request encoding

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.

4.1.2 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.

4.2 Exception handling

A response document having an **error** root element means that the XML service raised an exception. This can happen under several conditions: bad parameters, internal errors etc... 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 4.1 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.

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

Table 4.1: Summary of error ids

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

Figure 4.2 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** paremeter while the mandatory **uuid** parameter (as specified by the **message** element) is missing.

```
<error id="missing-parameter">
   <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 4.2: An example of generated exception

Chapter 5

General services

5.1 xml.info

The xml.info service can be used to query the site about its configuration, services, status and so on. For example, it is used by the harvesting web interface to retrieve information about a remote node.

Request

The xml request should contain at least one **type** element to indicates the kind of information to retrieve. More **type** elements can be specified to obtain more information at once. The set of allowed values are:

site Returns general information about the site like its name, id, etc...

categories Returns all site's categories

groups Returns all site's groups visible to the requesting user. If the user does not authenticate theirselves, only the **intranet** and the **all** groups are visible.

operations Returns all possible operations on metadata

regions Returns all geographical regions usable for queries

sources Returns all geonetwork sources that the remote site knows. The result will contain:

- The remote node's name and siteId
- All source uuids and names that have been discovered through harvesting.
- All source uuids and names of metadata that have been imported into the remote node through the MEF format.

users Returns all users visible to the calling one. The visibility policy is the following one:

- Administrators can see all users into the system (normal, other administrators, etc...)
- User administrators can see all users they can administrate and all other user administrators in the same group set. The group set is defined by all groups visible to the user administration, beside the All and the intranet groups.

- A logged user can se only theirself.
- A guest cannot see any user.

Request example:

```
<request>
   <type>site</type>
   <type>groups</type>
</request>
```

Response

Each **type** element produces an XML subtree so the response to the previous request is like this:

```
<info>
   <site>...</site>
   <categories>...</categories>
   <groups>...</groups>
   ...
</info>
```

Here follows the structure of each subtree:

- **site** : This is the container
  - **name** : Human readable site name
  - **siteId** : Universal unique identifier of the site
  - **platform** : This is just a container to hold the site's backend
    - **name** : Plaform name. For GeoNetwork installations it must be **geonetwork**.
    - **version** : Platform version, given in the X.Y.Z format
    - **subVersion** : Additional version notes, like 'alpha-1' or 'beta-2'.

  Example:

```
<site>
   <name>My site</name>
   <organization>FAO</organization>
   <siteId>0619cc50-708b-11da-8202-000d9335906e</siteId>
   <platform>
      <name>geonetwork</name>
      <version>2.1.0</version>
   </platform>
</site>
```

- **categories** : This is the container for categories.
  - **category** [0..n] : A single GeoNetwork's category. This element has an **id** attribute which represents the local identifier for the category. It can be usefull to a client to link back to this category.
    - **name** : Category's name
    - **label** : The localized labels used to show the category on screen. See 5.1 below.

Example:

```
<categories>
   <category id="1">
      <name>datasets</name>
      <label>
         <en>Datasets</en>
         <fr>I don't know</fr>
      </label>
   </category>
</categories>
```

- **groups** : This is the container for groups
  - **group** [2..n] : This is a Geonetwork group. There are at least the internet and intranet groups. This element has an **id** attribute which represents the local identifier for the group.
    - **name** : Group's name
    - **description** : Group's description
    - **referrer** : The user responsible for this group
    - **email** : The email address to notify when a map is downloaded
    - **label** : The localized labels used to show the group on screen. See 5.1 below.

Example:

```
<groups>
   <group id="1">
      <name>editors</name>
      <label>
         <en>Editors</en>
         <fr>I don't know</fr>
      </label>
   </group>
</groups>
```

- **operations** : This is the container for the operations
  - **operation** [0..n] : This is a possible operation on metadata. This element has an **id** attribute which represents the local identifier for the operation.
    - **name** : Short name for the operation.
    - **reserved** : Can be **y** or **n** and is used to distinguish between system reserved and user defined operations.
    - **label** : The localized labels used to show the operation on screen. See 5.1 below.

Example:

```
<operations>
   <operation id="0">
      <name>view</name>
      <label>
         <en>View</en>
         <fr>I don't know</fr>
      </label>
   </operation>
</operations>
```

- **regions** : This is the container for geographical regions

  - **region** [0..n] : This is a region present into the system. This element has an **id** attribute which represents the local identifier for the operation.
    * **north** : North coordinate of the bounding box.
    * **south** : South coordinate of the bounding box.
    * **west** : West coordinate of the bounding box.
    * **east** : east coordinate of the bounding box.
    * **label** : The localized labels used to show the region on screen. See 5.1 below.

  Example:

```
<regions>
   <region id="303">
      <north>82.99</north>
      <south>26.92</south>
      <west>-37.32</west>
      <east>39.24</east>
      <label>
         <en>Western Europe</en>
         <fr>Western Europe</fr>
      </label>
   </region>
</regions>
```

- **sources** : This is the container.
  - **source** [0..n] : A source known to the remote node.
    * **name** : Source's name
    * **uuid** : Source's unique identifier

  Example:

```
<sources>
   <source>
      <name>My Host</name>
      <uuid>0619cc50-708b-11da-8202-000d9335906e</uuid>
   </source>
</sources>
```

- **users** : This is the container for user information
  - **user** [0..n] : A user of the system
    * **id** : The local identifier of the user
    * **username** : The login name
    * **surname** : The user's surname. Used for display purposes.
    * **name** : The user's name. Used for display purposes.
    * **profile** : User's profile, like Administrator, Editor, UserAdmin etc...
    * **address** :
    * **state** :
    * **zip** :
    * **country** :
    * **email** :
    * **organisation** :

* **kind** :

Example:

```
<users>
   <user>
      <id>3</id>
      <username>eddi</username>
      <surname>Smith</surname>
      <name>John</name>
      <profile>Editor</profile>
      <address/>
      <state/>
      <zip/>
      <country/>
      <email/>
      <organisation/>
      <kind>gov</kind>
   </user>
</users>
```

Localized entities

Localized entities have a general **label** element which contains the localized strings in all supported languages. This element has as many children as the supported languages. Each child has a name that reflect the language code while its content is the localized text. Here is an example of such elements:

```
<label>
   <en>Editors</en>
   <fr>I don't know</fr>
   <es>Neither<es>
</label>
```

5.2 xml.forward

This is just a router service. It is used by Javascript code to connect to a remote host because a Javascript program cannot access a machine other than its server. For example, it is used by the harvesting web interface to query a remote host and retrieve the list of site ids.

Request

The service's request has this form:

```
<request>
   <site>
      <url>...</url>
      <type>...</type>
      <account>
         <username>...</username>
         <password>...</password>
      </account>
   </site>
    <params>...</params>
</knownNodes>
```

where:

site A container for site information where the request will be forwarded.

- **url** Indicates the remote url to connect to. Usually points to a GeoNetwork's xml service but can point to any XML service.
- **type** Its only purpose is to discriminate geonetwork nodes which use a different authentication scheme. The value **geonetwork** indicates these nodes. Any other value, or if the element is missing, indicate a generic node.
- **account** This element is optional. If present, the provided credentials will be used to authenticate to the remote site.

params This is just a container for the request that must be executed remotely.

Example:

```
<request>
   <site>
      <url>http://mynode.org:8080/geonetwork/srv/en/xml.info</url>
   </site>
    <params>
      <request>
         <type>site<type>
      </request>
   </params>
</request>
```

Please note that this service uses the GeoNetwork's proxy configuration.

Response

The response is just the response from the remote service.

Chapter 6

Harvesting services

6.1 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.

6.2 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 6.1 shows an example of output.

If you specify an id, you get a response like that one in figure 6.2 (for a web DAV 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.

6.3 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 honored. 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 choosen 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 usefull to restrict metadata access.

Figure 6.3 shows an example of an XML request to create 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 6.4 shows an example of an XML request to create a web DAV entry.

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 6.5 shows an example of an XML request to create a CSW entry.

Response

The service's response is the output of the **xml.harvesting.get** service of the newly created node.

Summary

The following table summaries the 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

6.4 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:

privileges If this element is omitted, privileges will not be changed. If specified, new privileges will replace the old ones.

categories Like the previous one.

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.

6.5 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:

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.

stop Makes a node inactive. Inactive nodes are never harvested.

run Just start the harvester now. Used to test the harvesting.

Request

A set of **id**s 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>
```

The following table summarizes, for each service, the possible status values:

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

```
<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 Organization</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 6.1: Example of an xml.harvesting.get response for a geonetwork 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 6.2: Example of an xml.harvesting.get response for a web DAV 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 6.3: Example of an xml.harvesting.add request for a geonetwork node

```
<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 6.4: Example of an xml.harvesting.add request for a web DAV node

```
<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 6.5: Example of an xml.harvesting.add request for a CSW node

Chapter 7

System configuration

7.1 Introduction

The GeoNetwork's configuration is made up of a set of parameters that can be changed to accomodate 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.

7.2 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** hyerarchy into the settings structure. See 11.2 for more information. The response has the following elements:

- **site** : A container for site information.
  - **name** : Site's name.
  - **organization** : Site's organization 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 organization.
  - **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 feeback 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 7.1 shows an example of xml.config.get response.

```
<config>
   <site>
      <name>dummy</name>
      <organization>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 7.1: Example of xml.config.get response.

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	-

Table 7.1: Mandatory and optional parameters for the **xml.config.update** service.

7.3 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 7.1.

Response

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

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

Otherwise a proper error element is returned.

Chapter 8

MEF services

8.1 Introduction

This chapter describes the services related to the Metadata Exchange Format (see chapter 3). These services allow to import/export metadata using the MEF format.

8.2 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**

8.3 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>
```

8.4 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.

Chapter 9

Relations

9.1 Introduction

This chapter describes general services used to get and set relations between metadata records inside GeoNetwork. The association is performed by a **Relations** table which stores a metadata **id** and a metadata **relatedId** fields (see figure 9.1).

9.2 xml.relation.get

This service retrieves all relations between metadata.

Request

The request accepts an **id** and a **relation** parameters, whose meaning is this:

- **id** (*integer*) : This is the local GeoNetwork identifier of the metadata whose relations are requested.
- **relation** (*string*, 'normal') : This optional parameter identifies the kind of relation that the client wants to be returned. It can be one of these values :
  - **normal** : The service performs a query into the **id** field and returns all **relatedId** records.
  - **reverse** : The service performs a query into the **relatedId** field and returns all **id** records.
  - **full** : Includes both normal and reverse queries (duplicated ids are removed).

Field	Datatype	Description
id	foreign key to Metadata(id)	Source metadata whose relation is being described.
relatedId	foreign key to Metadata(id)	Metadata related to the source one

Figure 9.1: Structure of table **Relations**

Here is an example of POST/XML request:

```
<request>
   <id>10</id>
   <relation>full</relation>
</request>
```

Response

The response has a **response** root element with several **metadata** children depending on the relations found. Example:

```
<response>
   <metadata>...</metadata>
   <metadata>...</metadata>
   ...
</response>
```

Each **metadata** element has the following structure:

- **title** : Metadata title
- **abstract** : A brief explanation of the metadata
- **keyword** : Keywords found inside the metadata
- **image** : Information about thumbnails
- **link** : A link to the source site
- **geoBox** : coordinates of the bounding box
- **geonet:info** : A container for GeoNetwork related information

Here is an example of the structure:

```
<metadata>
   <title>Globally threatened species of the world</title>
   <abstract> Contains information on animals. </abstract>
   <keyword>biodiversity</keyword>
   <keyword>endangered animal species</keyword>
   <keyword>endangered plant species</keyword>
   <link type="url">http://www.mysite.org</link>
   <geoBox>
      <westBL>-180.0</westBL>
      <eastBL>180.0</eastBL>
      <southBL>-90.0</southBL>
      <northBL>90.0</northBL>
   </geoBox>
   <geonet:info>
      <id>11</id>
      <schema>fgdc-std</schema>
      <createDate>2005-03-31T19:13:31</createDate>
      <changeDate>2007-03-12T14:52:46</changeDate>
      <isTemplate>n</isTemplate>
      <title/>
      <source>38b75c1b-634b-443e-9c36-a12e89b4c866</source>
      <uuid>84b4190b-de43-4bd7-b25f-6ed47eb239ac</uuid>
      <isHarvested>n</isHarvested>
      <view>true</view>
      <admin>false</admin>
      <edit>false</edit>
      <notify>false</notify>
      <download>true</download>
      <dynamic>false</dynamic>
      <featured>false</featured>
   </geonet:info>
</metadata>
```

Chapter 10

Schema information

10.1 Introduction

GeoNetwork is able to handle several metadata schema formats. Up to now, the supported schemas are:

- **ISO-19115** (iso19115) : GeoNetwork implements an old version of the draft, which uses short names for elements. This is not so standard so this schema is obsolete and will be removed in future releases.
- **ISO-19139** (iso19139) : This is the XML encoding of the ISO 19115:2007 metadata specification.
- **Dublin core** (dublin-core) : This is a simple metadata schema based on a set of elements capable of describing any metadata.
- **FGDC** (fgdc-std) : It stands for Federal Geographic Data Committee and it is a metadata schema used in North America.

In parenthesis is indicated the name used by GeoNetwork to refer to that schema. These schemas are handled through their XML schema files (XSD), which GeoNetwork loads and interprets to allow the editor to add and remove elements. Beside its internal use, GeoNetwork provides some useful XML services to find out some element properties, like label, description and so on.

10.2 xml.schema.info

This service returns information about a set of schema elements or codelists. The returned information consists of a localized label, a description, conditions that the element must satisfy etc...

Request

Due to its nature, this service accepts only the POST binding with application/XML content type. The request can contain several **element** and **codelist** elements. Each element indicate the will to retrieve information for that element. Here follows the element descriptions:

- **element** : It must contain a **schema** and a **name** attribute. The first one must be one of the supported schemas (see the section above). The second must be the qualified name of the element which information must be retrieved. The namespace must be declared into this element or into the root element of the request.
- **codelist** : Works like the previous one but returns information about codelists.

```
<request xmlns:gmd="http://www.isotc211.org/2005/gmd">
   <element schema="iso19139" name="gmd:constraintLanguage" />
   <codelist schema="iso19115" name="DateTypCd" />
</request>
```

Note The returned text is localized depending on the language specified during the service call. A call to /geonetwork/srv/**en**/xml.schema.info will return text in the English language.

Response

The response's root element will be populated with information of the elements/codelists specified into the request. The structure is the following:

- **element** : A container for information about an element. It has a **name** attribute which contains the qualified name of the element.
  - **label** : The human readable name of the element, localized into the request's language.
  - **description** : A generic description of the element.
  - **condition** [0..1] : This element is optional and indicates if the element must satisfy a condition, like the element is always mandatory or is mandatory if another one is missing.
- **codelist** : A container for information about a codelist. It has a **name** attribute which contains the qualified name of the codelist.
  - **entry** [1..n] : A container for a codelist entry. There can be many entries.
    - **code** : The entry's code. This is the value that will be present inside the metadata.
    - **label** : This is a human readable name, used to show the entry into the user interface. It is localized.
    - **description** : A generic localized description of the codelist.

Error code	Description
unknown-schema	The specified schema is not supported
unknown-namespace	The namespace of the specified prefix was not found
not-found	The requested element / codelist was not found

Figure 10.1: Possible errors returned by xml.schema.info service.

```
<response>
   <element name="gmd:constraintLanguage">
      <label>Constraint language</label>
      <description>language used in Application Schema</description>
      <condition>mandatory</condition>
   </element>
   <codelist name="DateTypCd">
      <entry>
         <code>creation</code>
         <label>Creation</label>
         <description>date when the resource was brought into
               existence</description>
      </entry>
      <entry>
         <code>publication</code>
         <label>Publication</label>
         <description>date when the resource was issued</description>
      </entry>
      <entry>
         <code>revision</code>
         <label>Revision</label>
         <description>date identifies when the resource was examined
               or re-examined and improved or amended</description>
      </entry>
   </codelist>
</response>
```

Error management

Beside the normal exceptions management discussed in section 4.2, the service can encounter some errors trying to retrieve an element/codelist information. In this case, the object is copied verbatim to the response with the addition of an **error** attribute that describes the encountered error. The returned errors are described in figure 10.1. Here follows an example of such response:

```
<response>
   <element schemma="iso19139" name="blablabla" error="not-found"/>
</response>
```

Part V

Settings internal structure

Chapter 11

Settings hierarchy

11.1 Introduction

GeoNetwork stores many options and information inside the **Settings** table. Information is grouped into hierarchies where each node has a key/value pair and can have many children. Each key is limited to 32 characters while each value is limited to 250. The 2 top level hierarchies are **system** and **harvesting**.

In the following sections, the indentation is used to show hierarchies. Names in bold represent keys with the value's datatype in parenthesys. An *italic* font is used to indicate basic types (string, integer, boolean) while normal font with a | is used to represent a set of allowed values. Regarding the boolean type, value can be only **true** or **false**. A missing datatype means that the value of the node is not used. Square brackets indicate cardinality. If they are missing, a cardinality of [1..1] should be considered.

11.2 The system hierarchy

- **site** : Contains information about the site
  - **name** (*string*) : Name used to present this site to other sites. Used to fill comboboxes or lists.
  - **organization** (*string*) : Name of the organization/company/institute that is running GeoNetwork
  - **siteId** (*string*) : A UUID that uniquely identifies the site. It is generated by the installer.
- **platform** : Contains information about the current version
  - **version** (*string*) : GeoNetwork's version in the X.Y.Z format
  - **subVersion** (*string*) : A small description about the version, like 'alpha-1', 'beta' etc...
- **server** : Used when it is necessary to build absolute URLs to the GeoNetwork server. This is the case, for example, when creating links inside a metadata or when providing CS/W capabilities.
  - **host** (*string*) : Main HTTP server's address
  - **port** (*integer*) : Main HTTP server's port (can be empty)

- **intranet** : specify the network of the intranet
  - **network** (*string*) : Network's address
  - **netmask** (*string*) : Network's netmask
- **z3950** : A container for Z39.50 server parameters
  - **enable** (*boolean*) : If true, GeoNetwork will start the Z30.50 server
  - **port** (*integer*) : The port opened by GeoNetwork to listen to Z39.50 requests. Usually is 2100.
- **proxy** : This container speficy proxy configuration to use
  - **use** (*boolean*) : If true, GeoNetwork will use the given proxy for outgoing connections
  - **host** (*string*) : Proxy's host
  - **port** (*integer*) : Proxy's port
  - **username** (*string*) : Proxy's credentials.
  - **password** (*string*) : Proxy's credentials.
- **feedback** : Feedback is sent with proper web form or when downloading a resource.
  - **email** (*string*) : email address of a GeoNetwork administrator or someone else
  - **mailServer** : This container represents the mail server that will be used to send emails
    - **host** (*string*) : Address of the SMTP server to use
    - **port** (*string*) : SMTP port to use
- **removedMetadata** : This container contains settings about removed metadata.
  - **dir** : This folder will contain removed metadata in MEF format. It gets populated when the user deletes a metadata using the web interface.
- **ldap** : Parameters for LDAP authentication
  - **use** (*boolean*)
  - **host** (*string*)
  - **port** (*integer*)
  - **defaultProfile** (*string*) : Default GeoNetwork's profile to use when the profile user attribute does not exist.
  - **login**
    - **userDN** (*string*)
    - **password** (*string*)
  - **distinguishedNames**
    - **base** (*string*)
    - **users** (*string*)
  - **userAttribs** : A container for user attributes present into the LDAP directory that must be retrieved and used to create the user in GeoNetwork.
    - **name** (*string*)
    - **password** (*string*)
    - **profile** (*string*)

11.3 Harvesting nodes

The second top level hierarchy is **harvesting**. All nodes added using the web interface are stored here. Each child has **node** in its key and its value can be **geonetwork**, **webdav**, **csw** or another depending on the node's type.

All harvesting nodes share a common setting structure, which is used by the harvesting engine to retrieve these common parameters. This imply that any new harvesting type must honor this structure, which is the following:

- **site** : A container for site information.
  - **name** (*string*) : Node's name as shown in the harvesting list.
  - **uuid** (*string*) : A unique identifier assigned by the system when the harvesting node is created.
  - **useAccount** (*boolean*) : Indicates if the harvester has to authenticate to access the data.
    - **username** (*string*) :
    - **password** (*string*) :
- **options** :
  - **every** (*integer*) : Timeout, in minutes, between 2 consecutive harvesting.
  - **oneRunOnly** (*boolean*) : If true, the harvester will harvest one time from this node and then it will set the status to inactive.
  - **status** (activelinactive) : Indicates if the harvesting from this node is stopped (inactive) or if the harvester is waiting until the timeout comes.
- **privileges** [0..1] **:** This is a container for privileges to assign to each imported metadata
  - **group** (*integer*) [0..n] : Indicate a local group. The node's value is its local identifier. There can be several group nodes each with its set of privileges.
    - **operation** (*integer*) [0..n] : Privilege to assign to the group. The node's value is the numeric id of the operation like 0=view, 1=download, 2=edit etc...
- **categories** [0..1] : This is a container for categories to assign to each imported metadata
  - **category** (*integer*) [0..n] : Indicate a local category and the node's value is its local identifier.
- **info** : Just a container for some information about harvesting from this node.
  - **lastRun** (*string*) : If not empty, tells when the harvester harvested from this node. The value is the current time in millis since 1 January, 1970.

Privileges and categories nodes can or cannot be present depending on the harvesting type. In the following structures, this common structure is not shown. Only extra information specific to the harvesting type is described.

11.3.1 Nodes of type geonetwork

This is the native harvesting supported by geonetwork 2.1 and above.

- **site** : Contains host and account information
  - **host** (*string*)
  - **port** (*integer*)
  - **servlet** (*string*)
- **search** [0..n] : Contains the search parameters. If this element is missing, an unconstrained search will be performed.
  - **freeText** (*string*)
  - **title** (*string*)
  - **abstract** (*string*)
  - **keywords** (*string*)
  - **digital** (*boolean*)
  - **hardcopy** (*boolean*)
  - **source** (*string*)
- **groupsCopyPolicy** [0..n] : Represents a copy policy for a remote group. It is used to maintain remote privileges on harvested metadata.
  - name (*string*) : Internal name (not localized) of a remote group.
  - policy (*string*) : Copy policy. For the group **all**, policies are: **copy**, **copyToIntranet**. For all other groups, policies are: **copy**, **createAndCopy**. The intranet group is not considered.

11.3.2 Nodes of type geonetwork20

This type allows harvesting from old geonetwork 2.0.x nodes.

- **site** : Contains host and account information
  - **host** (*string*)
  - **port** (*integer*)
  - **servlet** (*string*)
- **search** [0..n] : Contains the search parameters. If this element is missing no harvesting will be performed but the host's parameters will be used to connect to the remote node.
  - **freeText** (*string*)
  - **title** (*string*)
  - **abstract** (*string*)
  - **keywords** (*string*)
  - **digital** (*boolean*)
  - **hardcopy** (*boolean*)
  - **siteId** (*string*)

11.3.3 Nodes of type webdav

This harvesting type is capable of connecting to a web server which is WEB DAV enabled.

- **site** : Contains the URL to connect to and account information
  - **url** (*string*) : URL to connect to. Must be well formed, starting with 'http://', 'file://' or a supported protocol.
  - **icon** (*string*) : This is the icon that will be used as the metadata source's logo. The image is taken from the images/harvesting folder and copied to the images/logos folder.
- **options**
  - **recurse** (*boolean*) : Indicates if the remote folder must be recursively scanned for metadata.
  - **validate** (*boolean*) : If set, the harvester will validate the metadata against its schema and the metadata will be harvested only if it is valid.

11.3.4 Nodes of type csw

This type of harvesting is capable of querying a Catalogue Services for the Web (CSW) server and retrieving all found metadata.

- **site**
  - **capabUrl** (*string*) : URL of the capabilities file that will be used to retrieve the operations address.
  - **icon** (*string*) : This is the icon that will be used as the metadata source's logo. The image is taken from the images/harvesting folder and copied to the images/logos folder.
- **search** [0..n] : Contains search parameters. If this element is missing, an unconstrained search will be performed.
  - **freeText** (*string*)
  - **title** (*string*)
  - **abstract** (*string*)
  - **subject** (*string*)