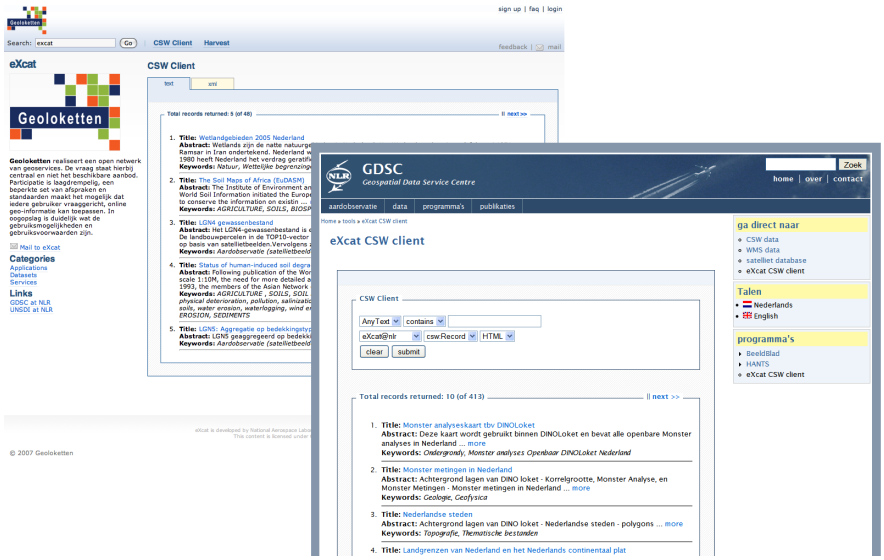
Nationaal Lucht- en Ruimtevaartlaboratorium
National Aerospace Laboratory NLR

Executive summary

eXcat OpenGIS CSW server and client

Manual



Problem area
Within the framework of the national Bsik programme “Space for Geo-information (RGI)”, the National Aerospace Laboratory developed a metadata software tool called eXcat. eXcat is a very efficient tool for storing and publishing of metadata according the OpenGIS standard for “Catalogue Services for Web” (CSW)

eXcat is written in Java and comprises a CSW server and a CSW Client. What makes that eXcat stands out from similar software is that it is built on the open-source XML database system eXist (http://exist.sourceforge.net), which uses web technology standards as XPath and XQuery. Also the Geotools, the open source (LGPL) Java code library is used (http://geotools.codehaus.org). eXcat is very easy to use and allows metadata to be saved in XML format (e.g. ISO 19115/19139) and to be published and queried using the CSW standard. eXcat also presents the metadata in a convenient and a uniform way to the user.

Furthermore, a second CSW client based on Javascript and AJAX was developed which can be included in existing web sites using a few simple steps.

Report no.
NLR-TR-2009-337

Author(s)
R.W. van Swol

Report classification
UNCLASSIFIED

Date
July 2009

Knowledge area(s)
Ruimtevaartgebruik

Descriptor(s)
CSW
OGC OpenGIS
Spatial data infrastrcuture

eXcat OpenGIS CSW server and client
Manual

Nationaal Lucht- en Ruimtevaartlaboratorium, National Aerospace Laboratory NLR

Anthony Fokkerweg 2, 1059 CM Amsterdam,
P.O. Box 90502, 1006 BM Amsterdam, The Netherlands
Telephone +31 20 511 31 13, Fax +31 20 511 32 10, Web site: www.nlr.nl

Nationaal Lucht- en Ruimtevaartlaboratorium
National Aerospace Laboratory NLR

NLR-TR-2009-337

eXcat OpenGIS CSW server and client

Manual

R.W. van Swol

Customer	Stichting RGI
Contract number	
Owner	National Aerospace Laboratory NLR
Division NLR	Aerospace Systems & Applications
Distribution	Limited
Classification of title	Unclassified
	July 2009

Approved by:

Author	Reviewer	Managing department
07/07/09	7/7/09	7.7.9

Summary

Now that the INSPIRE train is at full speed on its track to a standardized geo-information world, more and more tools become available supporting the user to get on this train. Within Geoloketten, a very efficient tool has been developed by the National Aerospace Laboratory for the publishing of metadata.

The importance of metadata is widely recognized, however to everyone who has investigated the matter thoroughly, it is known that generating and maintaining metadata is not the most appealing task. Because of that, the way in which data is described often lacks consistency and this doesn't improve the exchangeability of the data. Fortunately, the geo-information field is constantly moving: an unambiguous policy is being created with regard to standards and there is a growing consciousness that metadata is indispensable. Meanwhile, it is becoming easier to create metadata by the support of intelligent software tools.

When datasets have been provided with metadata in a correct manner, then the question arises how to handle and use these metadata. Often the information is saved in a database and sometimes there is an advanced way of searching this database for specific information. Usually, each organisation will have its own optimal way to manage data and the corresponding metadata. However, an important feature of metadata is that it helps users to find and share datasets. Publishing metadata on internet is thus very important for the multiple use of valuable geo-information.

In order to streamline this process, an interoperable standard was developed by the Open Geospatial Consortium: "Catalogue Service for Web" or CSW in short. The CSW standard describes how digital catalogues for geospatial data and services can be consulted. For the user this means that it is not necessary to know how the underlying database is organized in order to be able to consult the catalogue. For the developer it means that smart software can be developed supporting automated searches in distributed databases.

Within the project "Geoloketten" NLR developed eXcat. eXcat is a CSW server and a CSW client, written in JAVA. Special about eXcat is that it is built on an XML database. This XML database is the open source database system eXist, that uses web technology standards as XPath and XQuery. These are beautiful and impressive terms, but what really matters is that with this, a very easy-to-use system is developed allowing metadata in XML format (e.g. ISO 19115/19139), to be saved and searched for via the CSW standard. In addition, the metadata are

presented to the user in an human-readable and uniform fashion. The CSW client too, has been developed based on XQuery and can easily be integrated in existing websites.

Installation of the eXcat software consists of a few actions and without any further configuration it will work “out-of-the-box” on Windows, Unix or Mac OSX platforms. It is also quite easy for the user to add metadata in XML format to the database. With an administrator program it is possible to read files from disk. The user has the option to create a folder structure for better organisation of the files that are stored. Other administrators might prefer the use of the WebDAV interface; the folders will then be displayed in the explorer, as if it were normal folders of the system. Drag and drop can then be used to add or delete metadata files.

Using the CSW client, other CSW servers on internet can be queried as well and although the CSW protocol supports that multiple servers are queried simultaneously, often “harvesting” of data from other servers is preferred. Using the harvesting method metadata files from other remote CSW servers are retrieved and saved in the local database. This harvest feature is supported by the eXcat harvest module.

Contents

Abbreviations

AJAX	Asynchronous JavaScript and XML
CQL	Common Query Language
CSW	Catalogue Services for the Web
GIS	Geographic Information System
HTTP	Hypertext Transport Protocol
KVP	Key-Value Pair
NGII	National Geo-Information Infrastructure
OGC	Open Geospatial Consortium
RGI	Ruimte voor Geo-Informatie (Space for Geo-Information)
RPC	Remote Procedure Call
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTF	Unicode Transformation Format
WebDAV	Web-based Distributed Authoring and Versioning
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language

This page is intentionally left blank.

1 Introduction

Within the framework of the national Bsik programme Space for Geo-information (RGI) projects “Geoloketten (RGI-006)” and “A Remote Sensing Node within NGII (RGI-405)” the National Aerospace Laboratory developed eXcat. eXcat is a very efficient tool for publishing metadata using the standard for “Catalogue Services for Web” or CSW, which was developed by the OpenGIS Consortium (OGC).

eXcat is written in Java and comprises a CSW server and a CSW Client. What makes that eXcat stands out from similar software is that it is built on the open-source XML database system eXist (http://exist.sourceforge.net), which uses web technology standards as XPath and XQuery. Also the Geotools, the open source (LGPL) Java code library is used (http://geotools.codehaus.org). eXcat is very easy to use and allows metadata to be saved in XML format (e.g. ISO 19115/19139) and to be published and queried using the CSW standard. It also presents the metadata conveniently (human-readable) and in a uniform way to the user. Furthermore, a second CSW client based on Javascript and AJAX, was developed which can be included in existing websites using a few simple steps.

The importance of metadata is widely recognized, however for everyone who has investigated the matter thoroughly, it is known that generating and maintaining metadata is not the most appealing task. Because of that, the way in which data is described often lacks consistency and this doesn’t improve the exchangeability of the data. Fortunately, the geo-information field is constantly moving: an unambiguous policy is being created with regard to standards and there is a growing consciousness that metadata is indispensable. Meanwhile, it is becoming easier to create metadata by the support of intelligent software tools.

When datasets have been provided with metadata in a correct manner, then the question arises how to handle and use these metadata. Often the information is saved in a database and sometimes there is an advanced way of searching this database for specific information. Usually, each organisation will have its own optimal way to manage data and the corresponding metadata. However, an important feature of metadata is that it helps users to find and share datasets. Publishing metadata on internet is thus very important for the multiple use of valuable geo-information.

In order to streamline this process, an interoperability standard was developed by the Open Geospatial Consortium: “Catalogue Service for Web” or CSW in short. The CSW standard describes how digital catalogues for geospatial data and services can be consulted. For the user,

this means that it is not necessary to know how the underlying database is organized in order to be able to consult the catalogue. For the developer it means that smart software can be developed supporting automated searches in distributed databases.

2 The CSW protocol

The CSW protocol has been developed by OGC in order to explore databases via the web without the need of having knowledge of the underlying structure of that database. This is possible because the database comes with a (virtual) catalogue in standard format. As a result, it is possible to search within the ISO 19139 metadata XML records without the need to know the XML scheme. The catalogue provides a mapping between the agreed search terms (like Title, Subject, AnyText, etc.) and the corresponding element or attribute within the XML scheme. Currently, several updates of the specification have been published. The most relevant versions are version 2.0.1 and 2.0.2. These versions differ only slightly from each other. Unless stated otherwise, we only refer to CSW version 2.0.2 in this document.

2.1 CSW-HTTP binding

The communication between the client and the CSW server runs usually, but not necessarily, via the HTTP protocol. There have been made agreements on how the messages between client and server look like and the way in which they are sent. The messages that can be sent to a server are:

- GetCapabilities (to get a description of the CSW server capabilities)
- DescribeRecords (to get a description of the underlying record structure)
- GetDomain
- GetRecords (to search for certain records)
- GetRecordById (to acquire a record on the basis of a unique identifier)
- Harvest (to tell to the CSW server to retrieve a remotely stored record and save it in the database)
- Transaction

These messages can be packed in XML or in KVP (key-value pairs) format and can be sent via HTTP GET as well as HTTP PUT requests.

The server always answers with one of the following XML messages:
- <csw:Capabilities>...</csw:Capabilities>

- <csw:DescribeRecordResponse>...</csw:DescribeRecordResponse>
- <csw:GetRecordsResponse...</csw:GetRecordsResponse>
- <csw:GetRecordByIdResponse>...</csw:GetRecordByIdResponse>
- <csw:HarvestResponse>...</csw:HarvestResponse>
- <ows:ExceptionReport>...</ows:ExceptionReport>

The last answer above is an error message as a consequence of an incorrect formulated request.

For the sake of clarity, only the root element is shown of the XML document containing the request and response of the aforementioned messages. The full definition of these messages can be found in the CSW specification OGC 07-006r1 and OGC 07-045.

2.2 GetRecords Request

The GetRecords request is perhaps the most important request that can be sent to a CSW server. It contains the criteria on which a selection of (metadata) records is being made. According to the CSW protocol these criteria are provided in a so-called *constraint*. CSW enables two ways in which such a *constraint* can be formulated:
- CQL_TEXT, an as text-string formatted Common Query Language statement
- FILTER, an XML formatted query

Besides *constraint*, there are many more request parameters defined like for instance, the maximum number of records that are to be returned and the format and the scheme of these records. For a comprehensive description, the reader is referred to document OGC 07-045.

2.2.1 CQL_TEXT examples

Hereafter, a couple of examples of the use of CQL_TEXT for the selection of metadata records are presented.

1. All the records in which the string “water” appears are being searched for:

```
AnyText LIKE '%water%'
```

2. All the records in which the string “water” appears in the Title are being searched for:

```
Title LIKE '%water%'
```

3. All the records for which the Title starts with “water” are being searched for:

```
Title LIKE 'water%'
```

4. All the records for which the string “water” appears in the Title are being searched for and for which the geographical boundingbox coincides fully or partly with a certain area (here: the Netherlands in latitude and longitude coordinates):

```
Title LIKE '%water%' AND BBOX(ows:BoundingBox, 3.19,50.67,7.26,53.59)
```

5. All the records for which the string “water” appears in the Title are being searched for and for which the geographical boundingbox lies completely within a certain area: (here: the Netherlands in latitude and longitude coordinates):

```
Title LIKE '%water%' AND WITHIN(ows:BoundingBox,ENVELOPE(3.19,7.26,53.59,50.67))
```

2.2.2 ogc:Filter examples

The examples 1 and 4 from the previous paragraph, expressed as FILTER in XML are:

1. All the records in which the string “water” appears are being searched for:

```
<ogc:Filter xmlns:xml="http://www.w3.org/XML/1998/namespace"
xmlns:gml="http://www.opengis.net/gml" xmlns:ogc="http://www.opengis.net/ogc"
xmlns="http://www.opengis.net/ogc">
  <ogc:PropertyIsLike escape="\" singleChar="_" wildCard="%">
    <ogc:PropertyName>AnyText</ogc:PropertyName>
    <ogc:Literal>%water%</ogc:Literal>
  </ogc:PropertyIsLike>
</ogc:Filter>
```

4. All the records for which the string “water” appears in the Title are being searched for and for which the geographical boundingbox coincides fully or partly with a certain area (here: the Netherlands in latitude and longitude coordinates) (namespace definitions omitted):

```
<ogc:Filter>
  <ogc:And>
    <ogc:PropertyIsLike escape="\" singleChar="_" wildCard="%">
      <ogc:PropertyName>Title</ogc:PropertyName>
      <ogc:Literal>%water%</ogc:Literal>
    </ogc:PropertyIsLike>
    <ogc:BBOX>
      <ogc:PropertyName>ows:BoundingBox</ogc:PropertyName>
      <gml:Envelope>
        <gml:lowerCorner>3.19 50.67</gml:lowerCorner>
        <gml:upperCorner>7.26 53.59</gml:upperCorner>
      </gml:Envelope>
    </ogc:BBOX>
  </ogc:And>
</ogc:Filter>
```

2.2.3 GetRecords examples (KVP and XML)

A full GetRecords request may look like this:

KVP request with CQL_TEXT constraint:

```
http://localhost:8080/excat/csw?request=GetRecords&service=CSW
&version=2.0.2&namespace=xmlns(csw=http://www.opengis.net/cat/csw)
&resultType=results&outputSchema=http://www.opengis.net/cat/csw/2.0.2
&outputFormat=application/xml&maxRecords=10&typeNames=csw:Record
&elementSetName=summary&constraintLanguage=CQL_TEXT
&constraint_language_version=1.1.0
&constraint=AnyText+LIKE+%27%25water%25%27
```

XML request with FILTER constraint:

```
<?xml version="1.0" encoding="UTF-8"?>
<csw:GetRecords xmlns:csw="http://www.opengis.net/cat/csw/2.0.2" maxRecords="10"
outputFormat="application/xml"
           outputSchema="http://www.opengis.net/cat/csw/2.0.2" resultType="results"
service="CSW" version="2.0.2">
  <csw:Query typeNames="csw:Record">
    <csw:ElementSetName>summary</csw:ElementSetName>
    <csw:Constraint version="1.1.0">
      <ogc:Filter xmlns:ogc="http://www.opengis.net/ogc"
xmlns="http://www.opengis.net/ogc" xmlns:gml="http://www.opengis.net/gml">
        <ogc:PropertyIsLike escape="\" singleChar="_" wildCard="%">
          <ogc:PropertyName>AnyText</ogc:PropertyName>
          <ogc:Literal>%water%</ogc:Literal>
        </ogc:PropertyIsLike>
      </ogc:Filter>
    </csw:Constraint>
  </csw:Query>
</csw:GetRecords>
```

3 The eXist XML database

Metadata for geo-information (data and services) are described according to the ISO19115/9 and ISO19139 standards. The way in which these metadata are presented in XML is described by the ISO19139 standard. Thus, metadata for geo-information are often XML formatted documents. For metadata catalogues, usually relational databases are used which can be queried using SQL searches. However, if XML files are saved in such relational databases, the possibility to search on characteristics within the XML files is lost.

The use of an XML database offers therefore important advantages. The most important one is that one can search within documents using XPath and XQuery. XPath and XQuery are standards developed by the World Wide Web Consortium (W3C) and are used in many applications.

3.1 XPath

XPath (see: http://www.w3.org/TR/xpath) is a language to indicate elements and attributes of an XML document and it also offers the possibility to manipulate with text, numbers and booleans. XPath is often used in XSL (stylesheet) transformations whereby an XML document is being converted to another XML document according to the rules that are specified in an XSLT document.

3.2 XQuery

XQuery (see: http://www.w3.org/TR/xquery) is a language designed to explore XML documents in an intelligent and transparent way. It uses XPath to execute tests (queries) on components of XML documents in order to retrieve XML fragments or whole documents from a collection of XML documents.

3.3 eXist

eXist has been used as the underlying XML database for the development of eXcat. eXist (see: http://exist.sourceforge.net) is an open-source database management system that is fully built on XML technology and it supports the standards XPath and XQuery. This XML database is written in Java and can be easily integrated in applications where XML is used. High performance is achieved because a smart indexing scheme is being applied with which relations (like: (parent-child, ancestor-descendant of previous-next) between fragments of the XML documents can be determined quickly.

The eXist database consists of a number of important features of which a few are named here: XQuery support, authorization mechanism, crash recovery, backup-restore functionality, and more. Besides this, the following network protocols are supported: HTTP/REST, WebDAV, SOAP and XML-RPC.
For the eXist database, an Admin client is also available, giving simple access to the database and with which the structure, documents and user rights can be managed. With this client a backup of the XML documents can be made and an existing backup can be restored.

4 eXcat CSW server and client

4.1 eXcat CSW server

Just like eXist, eXcat has been developed in Java and operates in a servlet-engine like Tomcat. In eXcat, the CSW protocol for the "ISO 19115 Application Profile for Metadata" has been implemented. This means that standardized CSW requests are handled by eXcat and responses conforming the CSW standard are being returned. These requests are mostly search requests for the metadata database. The following CSW requests are being supported:

- GetCapabilities
- DescribeRecords
- GetRecords
- GetRecordById
- Harvest

All requests can be sent to the eXcat CSW server as HTTP GET request as well as HTTP PUT requests.

Because eXcat is developed on the eXist XML database, all the interfaces supported by eXist are available, e.g. REST, SOAP, XML-RPC and WebDAV interfaces. The standard Admin client of eXist can be used for managing the metadata (in ISO 19139 XML format).

4.2 eXcat integrated CSW client

An XQuery based CSW client has been developed. This client is available via a web browser, and is intended to be used by mainly experts in the field of CSW. The client offers possibilities to query the eXcat server and other remote CSW servers. Thereby, it is possible to formulate a query in CQL_TEXT which is sent as CQL_TEXT or as ogc:FILTER. [Remark: in the eXcat software, the Geotools library is used for the translation of CQL to ogc:FILTER (see: http://geotools.codehaus.org)] Besides this, the request method (GET or POST) and the request format (KVP or XML) can be selected. Other parameters which can be configured by the user are: version, resultType, maxRecords, outputSchema, ElementSetName and SortBy.

The client also offers exercise possibilities in the use of filters. The as CQL_TEXT entered constraint can be presented as the corresponding ogc:Filter XML representation and as a XQuery expression.

It is also possible to control the Harvest module from the same browser client, which is implemented as a separate servlet (independent of the eXcat CSW server). This control is very

simple and consists only of starting, monitoring and stopping of the harvest process for selected remote CSW servers.

4.3 eXcat CSW Javascript/AJAX client

Because the aforementioned CSW client is built up by so-called *xql* pages, which are only usable in a servlet engine like Tomcat, this client is not usable within just an arbitrary web site. Because of that, a client based on Javascript and AJAX was developed, which can be fitted easily into to existing web pages. The client is also easier to use: the user has the possibility to enter a search term on the basis of which the whole text (AnyText), just the Title or just the Subject is searched. Knowledge of CQL_TEXT or FILTER constraints is not required. In order to let the client communicate with remote servers, a proxy is necessary. For security reasons, browsers don't allow the exchange of XML messages via Javascript with servers other than the one from which the pages are loaded. Via a simple proxy, in this case written in PHP, this limitation can be surpassed.

4.4 eXcat Harvest module

An independent Harvest module is made available. Harvesting is the process of collecting metadata records from remote locations and storing these records in a local database. Usually, these metadata records are retrieved from different remote CSW servers but the records may also be retrieved from a local file system.

Harvesting is often used as an alternative for distributed search actions where a search to a CSW server is passed by that server to another server and the results are collected. In practice, this procedure does not always works efficiently because servers might be irresponsive and consequently the whole search process is slowed down. In addition, it is often difficult to sort the collected records before sending the result to the user. With one database containing all the collected data, these problems don't exist. However, the disadvantage is that it is not sure whether or not the harvested records are still up-to-date. By frequently executing the Harvest process and because in practice metadata are not updated very often, this disadvantage needs not to be of great importance.

The current eXcat Harvest module works semi-automated and has to be started manually by the user. After the metadata are collected on the file system, the records are presented to the eXcat CSW server one by one via the standard CSW Harvest request. The advantage of this is that the Harvester is also capable of operating in combination with any other CSW server supporting the Harvest request method.

It is the task of the CSW server to determine if and how the record will be saved. The standard way of operating of the eXcat CSW server is that metadata already present in the database (based on the unique file identifier) are not stored. Replacing existing metadata records only takes place if the metadata comes from the same remote CSW server. This prevents duplication of records collected from multiple CSW servers.

5 Installation procedure eXcat

5.1 Installation

The eXcat software is available as a so-called Java war-file. This makes the installation very easy. The whole procedure consists of three steps.

Step 1: Installation of the software

Copying the file *excat.war* to the directory $CATALINA_HOME/webapps is sufficient. [$CATALINA_HOME is the directory where Tomcat is installed. In Windows, this is usually C:\Program Files\Apache Software Foundation\Tomcat 5.5.] When Tomcat is started, the file *excat.war* will be unpacked automatically and all files will be placed in the directory $CATALINA_HOME/webapps/excat. A new and empty XML database is generated automatically. The directory structure is shown in figure 1. A description of the folders follows below.

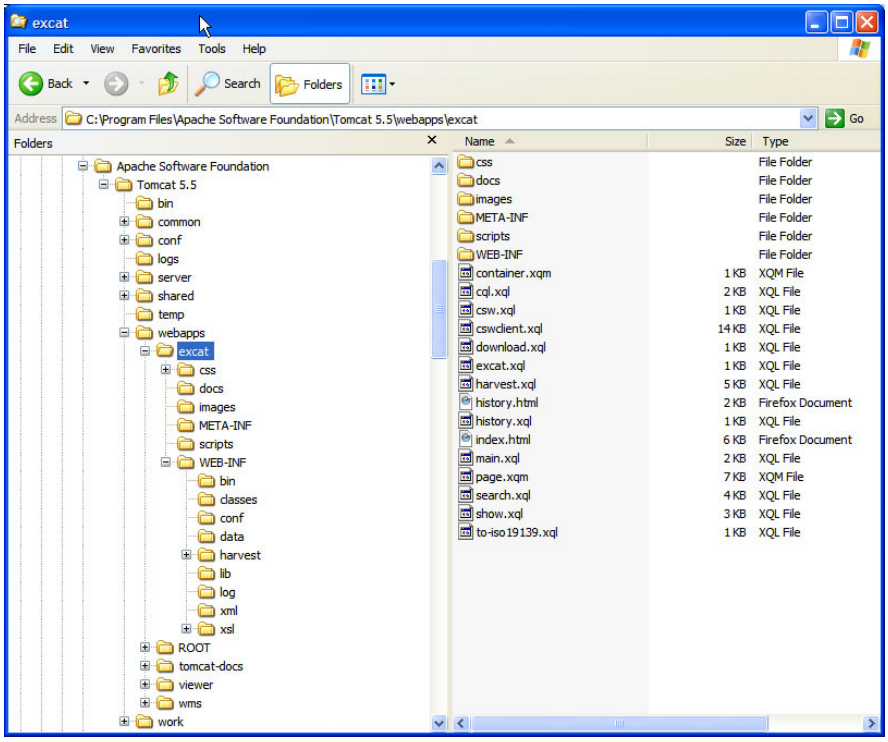


Fig.1 Folder structure of the eXcat web application

`css`: contains the css stylesheets which determine the lay-out of the client .
`doc`: contains a number of documents with background information.
`images`: contains images used for the lay-out of the client webpages.
`scripts`: contains the excat.js and prototype.js javascript files. Javascript is used by the client in order to solve the “back-button” problem. Dynamically modified pages give the impression of a freshly loaded page, but in fact, they are not. Because of that, the browser back-button cannot be used to go to the previous page. With some Javascript code however, it is still possible to apply this intuitive method.
`META-INF`: is a standard directory in web-applications in which, among other things, the automatic reloading of the applications is controlled.
`WEB-INF`: contains all the help files and other files which remain hidden for the web user. This directory contains the following subdirectories:

`WEB-INF/bin`: contains the program to start the Admin client of the eXist XML (Unix and Windows versions).
`WEB-INF/classes`: contains the configuration file for the log-messages.
`WEB-INF/conf`: contains the eXcat configuration files (see hereafter).
`WEB-INF/data`: contains the eXist database files.

`WEB-INF/harvest`: contains the (temporary) XML metadata records which are retrieved by the Harvest module from remote CSW servers.
`WEB-INF/lib`: all Java jar libraries.
`WEB-INF/log`: the various log-files
`WEB-INF/xml`: some XML-files: capabilities, describerecord and a sample metadata record.
`WEB-INF/xsl`: XSL stylesheets for the CSW client and for the server. Transformation from XML to another XML format are defined by these stylesheets. XSL stylesheets for transforming ISO19139 records to Dublin Core format are located here as well.

Step 2: Check the correct functioning

The eXcat web application is readily available. Browse to http://localhost:8080/excat and the main page of eXcat becomes visible (see Fig. 2). It is assumed that Tomcat is configured on port 8080. This is the standard setting. If this is changed, of course a different port number has to be used.

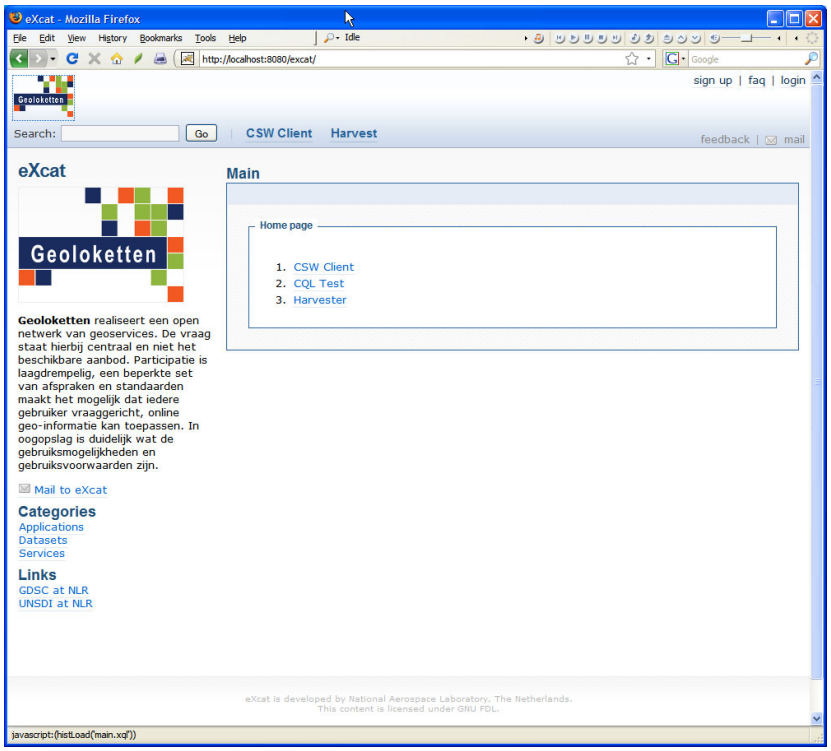


Fig. 2 eXcat client main page

Step 3: Install sample record

Go from the main page to CSW Client. A search form appears (see Fig. 3). Don't fill in anything but press the "Submit" button. Now, one metadata record will be found, namely the description of eXcat in ISO19139 format (see Fig. 4). This record is saved automatically in the XML database if a search is formulated and there are no records in the database yet. The XML

response received from the server can be shown by clicking the "XML-tab". If the record is selected (by clicking on the title or on “more”) a human-readable representation of the file is shown. Again, the “XML-tab” gives the possibility to look at the raw XML response of the server.

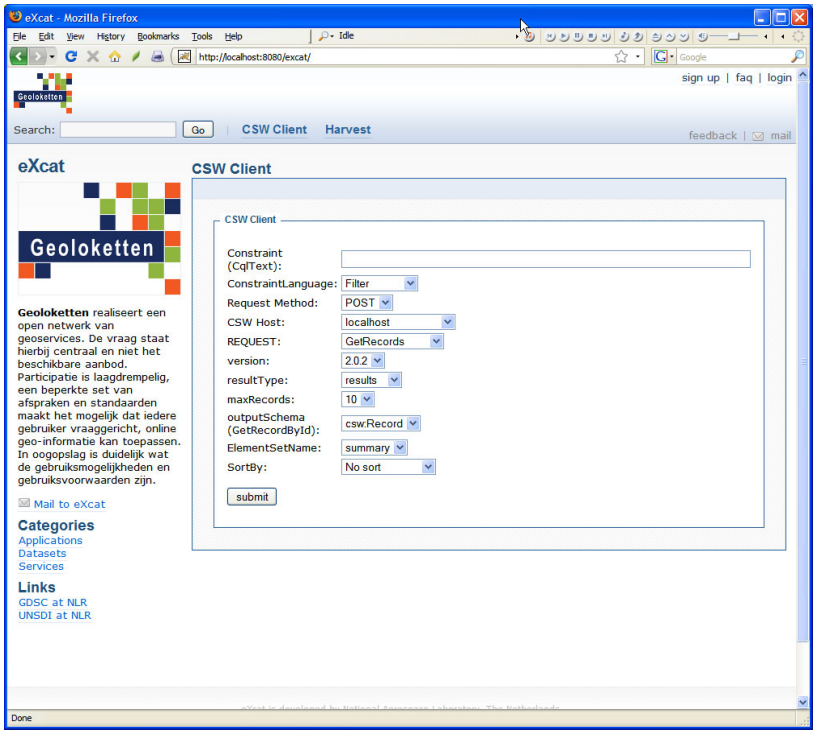


Fig. 3 eXcat client search form

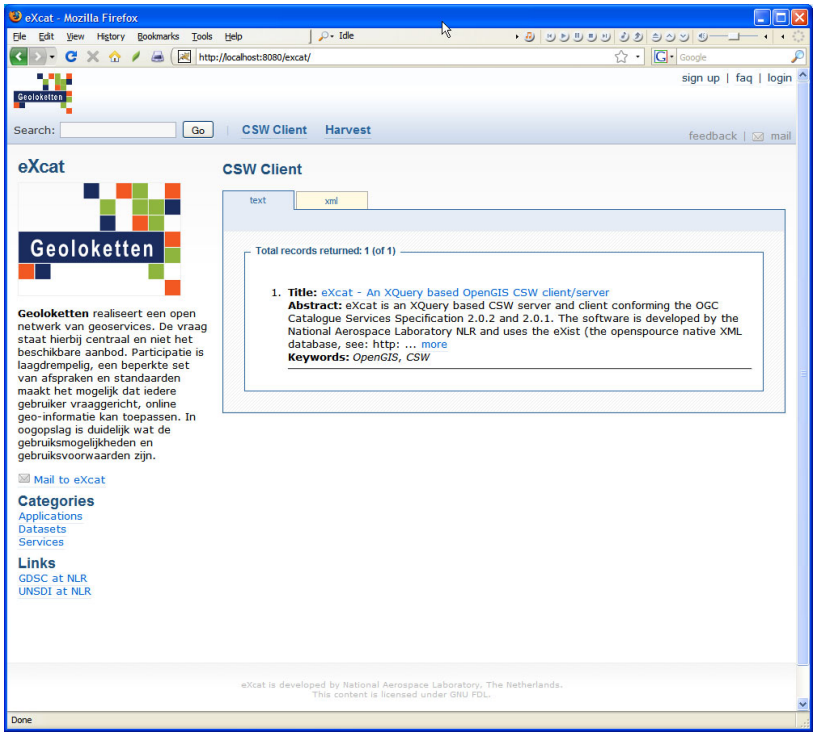


Fig. 4 eXcat metadata search result

5.2 Adding metadata to the database

The newly installed software is now ready for use and metadata records can be added. There are three ways to offer metadata to the CSW server:

1: using the Admin client

2: using the WebDAV protocol

3: using the Harvest request

5.2.1 Use of the Admin client

The Admin client provided together with the eXist database, offers a practical method for adding metadata records. It allows the user to save data in an organized and orderly manner using a folder structure.

The Admin client is a Java application which can be started by executing the program client.bat (Windows) or client.sh (Unix, OSX) from the WEB-INF/bin directory. Next, a couple of informative messages will appear, but they can be ignored. The login screen (see Fig. 5) appears and it is sufficient to press the “OK” button.

[**Remark**: One also has the possibility here, to change the URL and to choose a different address. Furthermore, addresses of other eXist databases can be saved and added to the list of favourites.]

The client now shows the underlying database structure as a folder structure. In Fig. 6, the meaning of the buttons on top of the screen is described.

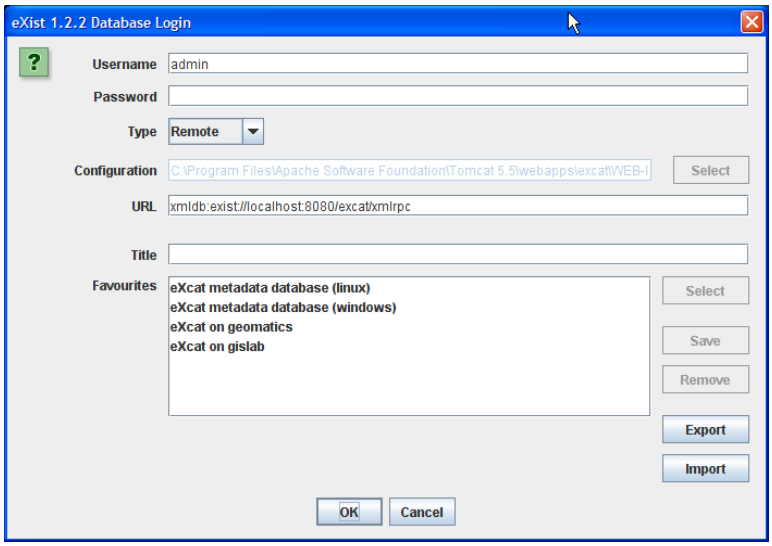


Fig. 5 eXist Admin client login screen

Two folders are visible: *csw* and *system*. It is recommended to ignore the folder *system* and double-click on *csw* to enter the folder. Inside *csw*, a folder *excat* is visible in which the example record *excat.xml* is stored. [Note: this sample record is stored when the CSW server is accessed from the browser for the first time; see above.] This record can be shown in an editing pane by just double-clicking. In principle, the shown file can now be modified, but this is not advised for proper management of the metadata. It is recommended to modify metadata outside the database environment since the database will often contain just a copy of the original metadata record, the original being kept close to the data it describes.

It is recommended to introduce some structure to the way in which metadata are being saved and to create a new folder inside the *csw* folder. This can be done using button 3 (see Fig. 6). Go into the new folder (double click) and use button 4 (see Fig. 6) in order to add new files. A new window is opened (Fig. 7) in which the XML-files that one wants to add can be selected. From the eXcat client (in the browser) the files can be immediately found and made visible.

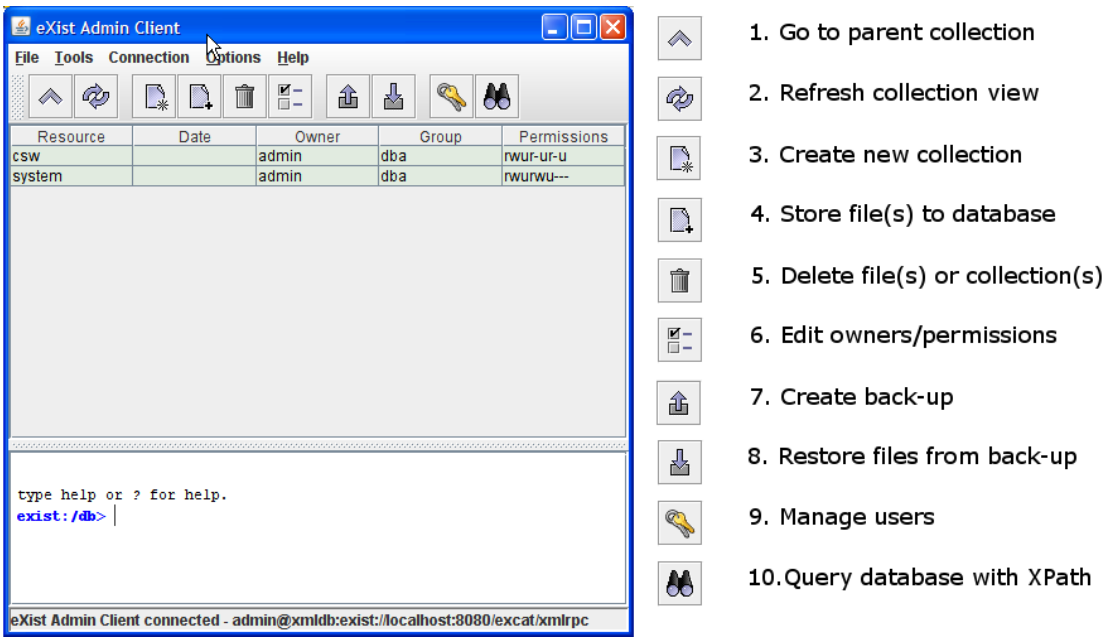


Fig. 6 eXist Admin main screen

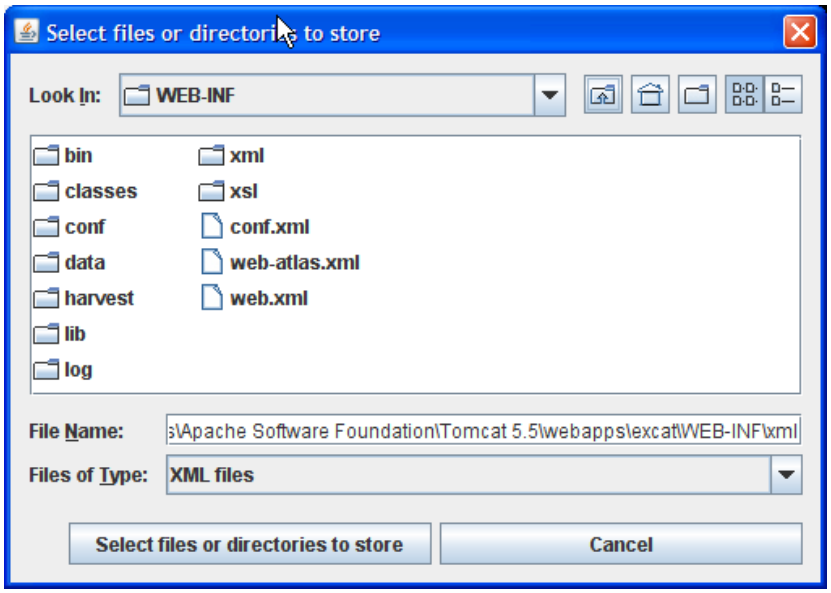


Fig. 7 Adding metadata using the eXist client

5.2.2 Use of WebDAV

WebDAV is a protocol with which files on a server can be made accessible as if they were on the file system of the local computer. eXist supports this protocol by which the content of the database can be made accessible as a folder with files.

Important: the file WEB_INF/web.xml should contain the following section:

```
<servlet>
   <servlet-name>WebDAVServlet</servlet-name>
   <servlet-class>org.exist.http.servlets.WebDAVServlet</servlet-class>
</servlet>
<servlet-mapping>
   <servlet-name>WebDAVServlet</servlet-name>
   <url-pattern>/webdav/*</url-pattern>
</servlet-mapping>
```

In Windows, the eXist database can be attached as a network folder using the following procedure:

1. Go via **My Computer** to **My Network Places** (or use the explorer to show **My Network Places**) and click with the right mouse button on **open**. Next choose **Add a network place** in the upper left corner of the sidebar.
2. The window that opens is the **Add Network Place Wizard**. Click on **Next**.
3. On the next page enter the URL of the WebDAV folder in the field **Internet or network address**. Example:
http://localhost:8080/excat/webdav and click on **Next**.

4. By default the eXist database is protected with a username and password. Enter the name (default: admin) and password (default: empty) and click on **OK**.
5. Now, enter the name of this folder (e.g. eXcat webdav localhost) - this name will appear in the list of **My Network Places** (see Figure 8).
6. Finally, click on **Finish** on the next page.

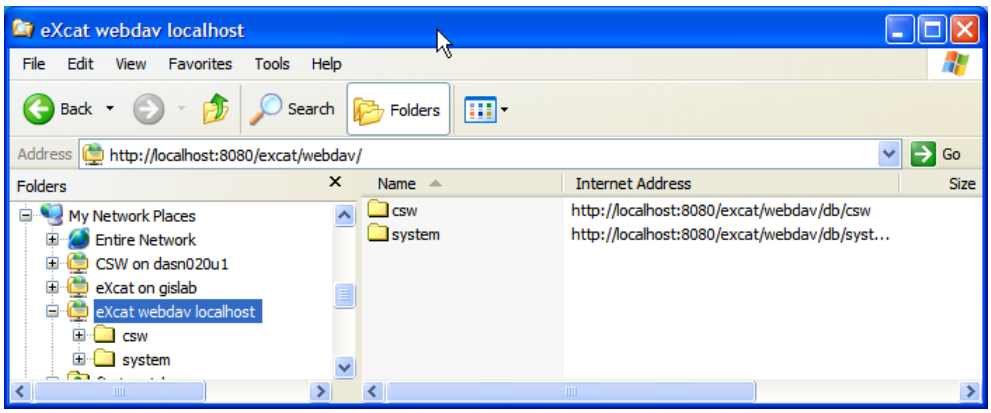


Fig. 8 eXcat as web folder (via WebDAV)

New folders and XML metadata files can now be added or deleted. However, it is not possible to modify the files, which is not recommended anyway (see discussion above).

5.2.3 Use of the Harvest request

Theoretically the use of the Harvest request also offers the possibility to place XML files in the CSW server, but this method has to be considered as an expert method. For the sake of completeness, this method is described here.

When the metadata file *my-metadata.xml* is located in the folder WEB-INF/harvest/my-xml, than the eXcat CSW server will accept the following KVP request (via HTTP-GET):

```
http://localhost:8080/excat/csw?request=Harvest&service=CSW
&version=2.0.2&namespace=xmlns(csw=http://www.opengis.net/cat/csw)
&source=my-xml/my-metadata.xml
&resourceFormat=application/xml
&resourceType=http://www.isotc211.org/2005/gmd
&collection=my-xml
```

The same request in XML (via HTTP-POST) looks like this:

```
<?xml version="1.0" encoding="UTF-8"?><csw:Harvest
xmlns:csw="http://www.opengis.net/cat/csw/2.0.2" service="CSW" version="2.0.2">
  <csw:Source>my-xml/my-metadata.xml</csw:Source>
  <csw:ResourceType>http://www.isotc211.org/2005/gmd</csw:ResourceType>
  <csw:ResourceFormat>application/xml</csw:ResourceFormat>
  <csw:Collection>my-xml</csw:Collection>
</csw:Harvest>
```

Remark 1: the parameter *collection* is not standard and is being used by eXcat to place the file in a specific folder (here: my-xml). The Harvest module uses this method to store metadata in a dedicated folder using the unique id of the remote server (see the csw configuration file) as the collection name.

Remark 2: the parameter *source* is a URI and can also have the following form:

```
source=file:/C:\Documents and Settings\Desktop\my-metadata.xml
```

When the protocol specification (here: file:/) is missing than the path is relative with respect to WEB-INF/harvest. This standard path can be modified in the configuration file (see later).

5.3 Installation Javascript/AJAX client

Besides the integrated eXcat CSW (expert) client, also a simple client was developed based on of Javascript and AJAX. This client can easily be integrated in existing HTML pages by adding the HTML code below into the page and by saving the source files according to the folder structure in figure 9.

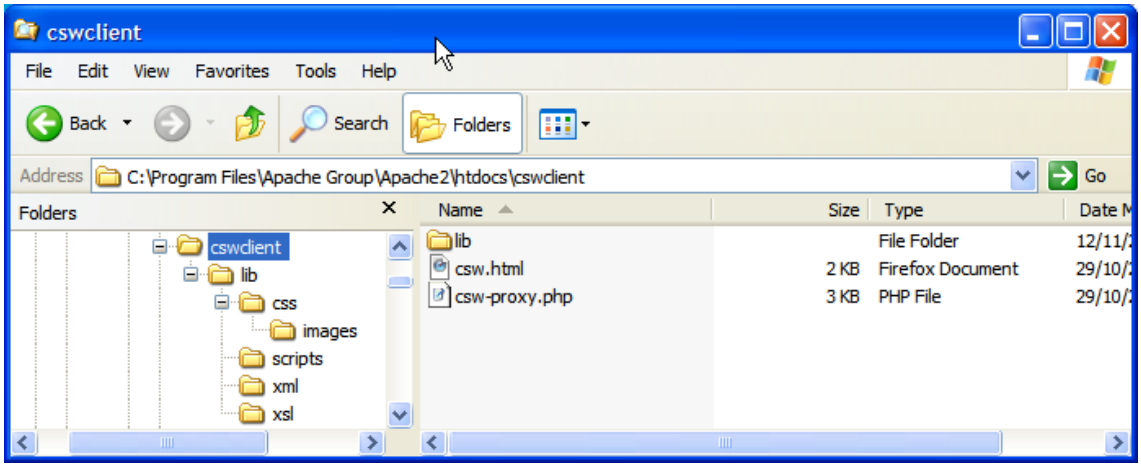


Fig. 9 Folder structure Javascript/AJAX CSW client

The HTML code consists of references to a css-stylesheet (*cswclient.css*), to three javascript libraries (*sarissa.js*, *sarissa_ieemu_xpath.js* and *cswclient.js*) and a section with the Javascript code that places the CSW client in the page.

```
<link rel="stylesheet" type="text/css" href="lib/css/cswclient.css"/>
<script type="text/javascript" src="./lib/scripts/sarissa.js"></script>
<script type="text/javascript" src="./lib/scripts/sarissa_ieemu_xpath.js"></script>
<script type="text/javascript" src="./lib/scripts/cswclient.js"></script>

<div class="csw-wrapper" id="csw-wrapper" style="width:80%">
     <script type="text/javascript">
        var csw_client = new CSWClient();
        csw_client.writeClient("csw-wrapper");
     </script>
</div>
```

This is the standard method, where the client offers access to the CSW servers defined in the configuration file *lib/xml/cswclient.xml.* In order to get access to remote CSW servers with AJAX, the use of a proxy is necessary. A PHP proxy is used (*csw-proxy.php*) which implies that the web-server (like Apache) has to support the use of PHP. [**Important**: standard installations of PHP require that the Curl extension is activated by removing the comment (;) character from the php.ini configuration file.]
Optionally, the CSW client can also be initialized using one of the following ways:

1. Use only http://some-csw-server/excat/csw as CSW host and csw-proxy.php can be found in http://my-host/some/path (instead of in the standard path):

```
var csw client = new CSWClient("http://some-csw-server/excat/csw","http://my-
host/some/path");
```

2. Use only http://some-csw-server/excat/csw as CSW host and csw-proxy.php can be found in the standard path:

```
var csw_client = new CSWClient("http://some-csw-server/excat/csw");
```

3. Use only http://my-host/excat/csw as CSW host (on the same server and port with which the HTML pages are loaded with the CSW client) and a proxy is not necessary:

```
var csw client = new CSWClient("http://my-host/excat/csw");
csw_client.useProxy(false);
```

Examples of the eXcat Javascript CSW client are shown in figure 10, figure 11 and figure 12.

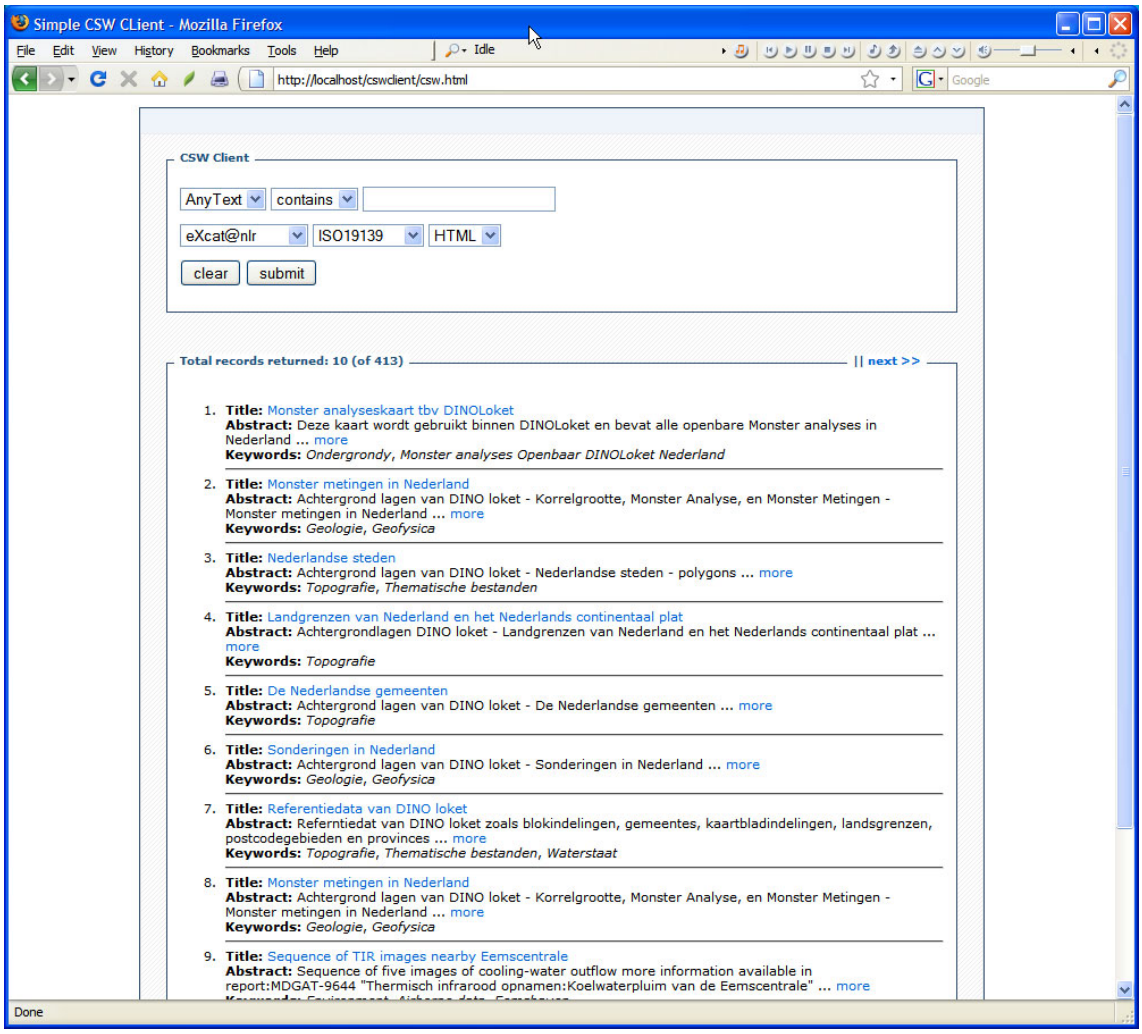


Fig. 10 Search result Javascript CSW client

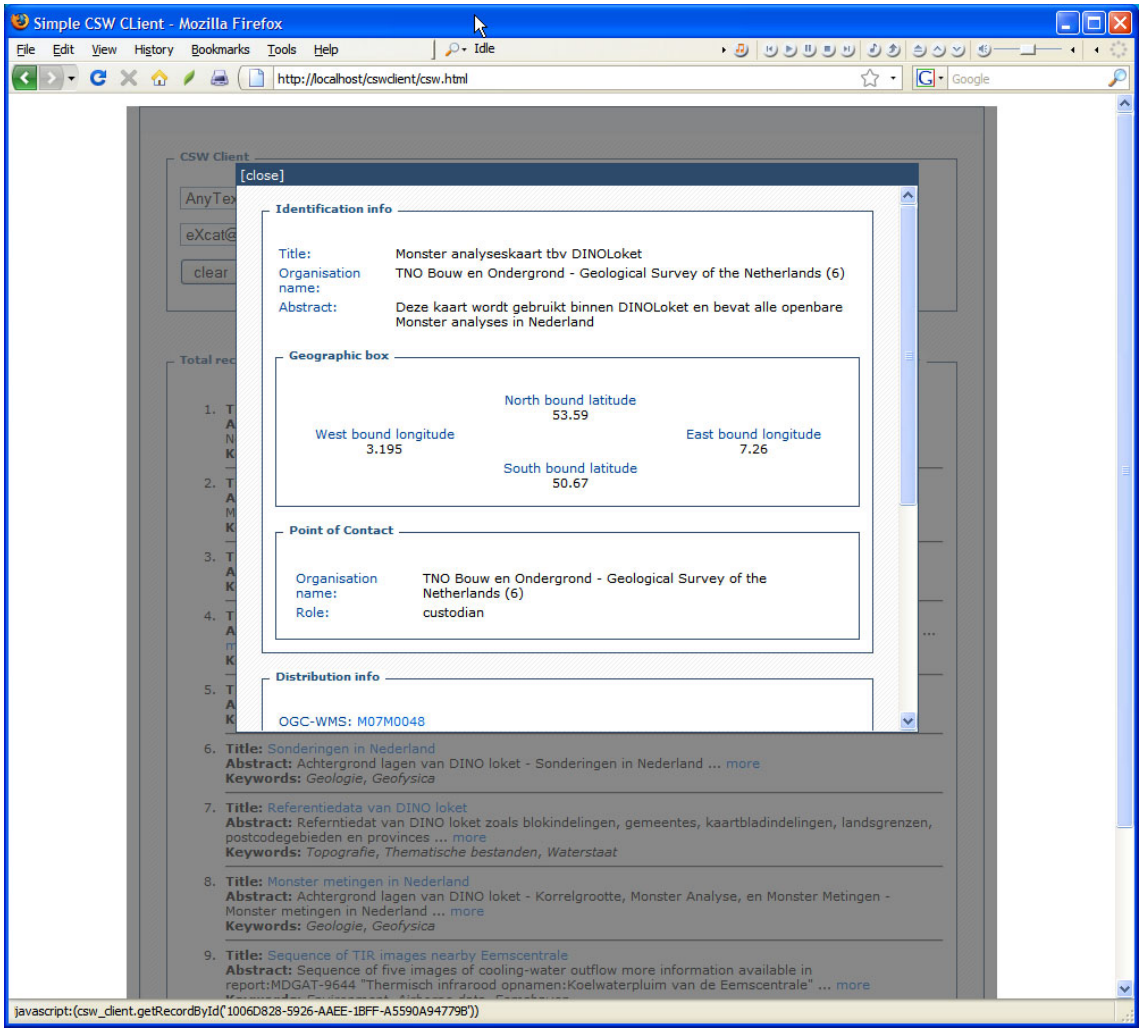


Fig. 11 Metadata presentation in Javascript CSW client

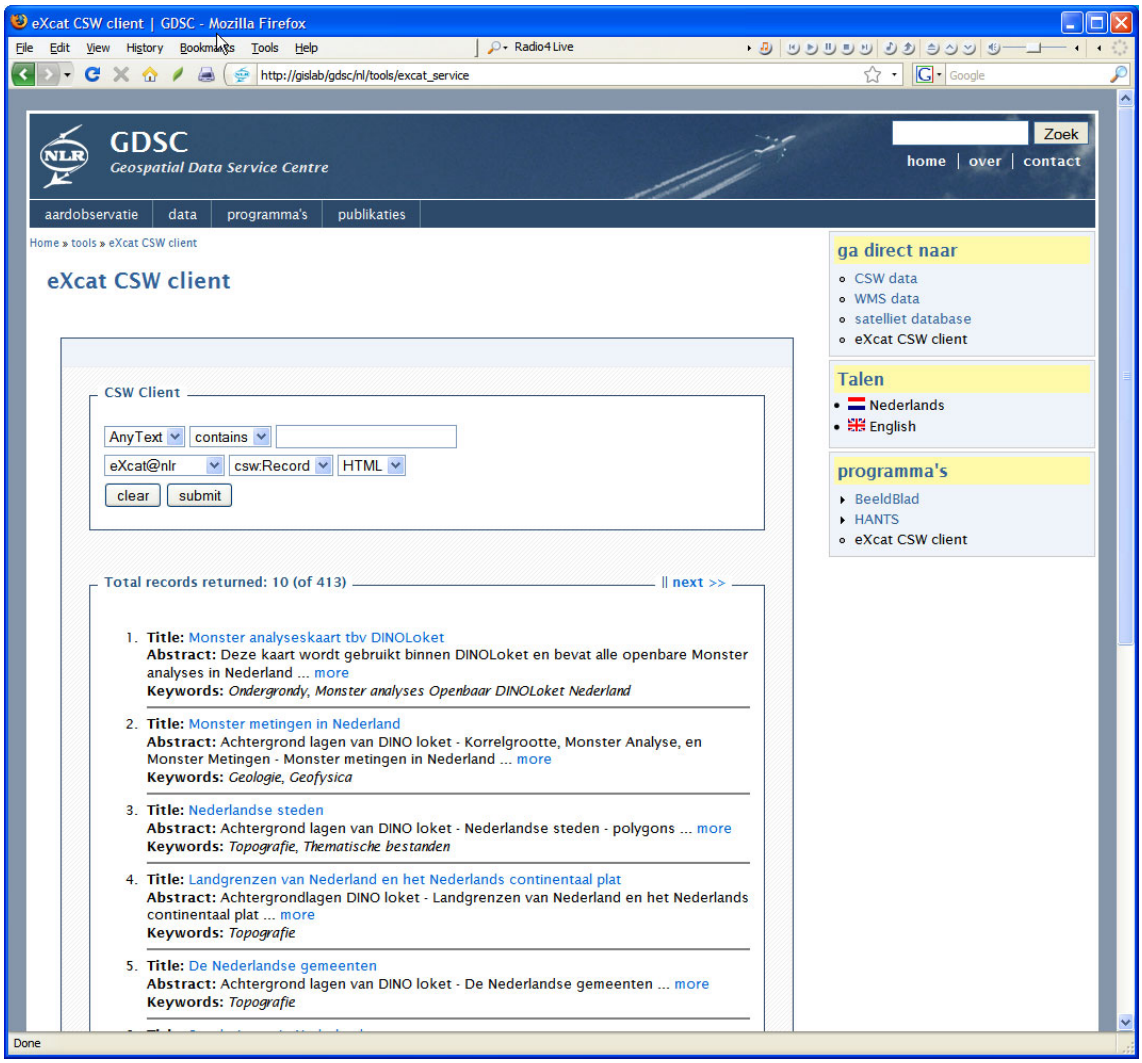


Fig. 12 eXcat CSW client in GDSC site

6 Advanced use of eXcat

In the next sections, it is described how eXcat can be configured when the standard settings don't suffice. These modifications are reserved for the administrator of the eXcat installation and some background knowledge of the CSW protocol is required.

6.1 Adapted configuration of eXcat

The standard configuration of eXcat can be changed by modifying a number of configuration files. These files are in the eXcat web application in the WEB-INF/conf folder: *allow.xml*, *csw.properties*, *csw-hosts.xml*, *harvest.properties*.

6.1.1 allow.xml

This XML file controls which hosts have access to the Harvest procedure via the eXcat client. The file contains the IP addresses of the machines from where one may run a Harvest process. The CSW server uses the same information when a Harvest request is sent to the CSW server. Machines not appearing in the list cannot see the options Harvest and CQL Test in the main page of the CSW client. Also the Harvest requests coming from these machines are not answered by the CSW server.

An example of this file:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<harvest>
  <allow host="127.0.0.1" />
  <allow host="137.17.2.14" />
</harvest>
```

6.1.2 csw.properties

An example of this file *csw.properties* is shown hereafter. The most important parameters which and administrator might want to change are:

csw.xmldb.uri: the URI of the eXist database; in principle, this can be on a different machine than the one on which the eXcat servlet is installed.
csw.capabilities.file: the name of the capabilities document in case that the standard file is not being used.
csw.describerecord.file: the name of the describerecord document in case that the standard file is not being used.
csw.xmldb.admin: the username required to get access to the eXist database.
csw.xmldb.admin.password: the password required to get access to the eXist database.
csw.harvest.base.uri: this is the base URI from which the CSW server loads the XML files during the Harvest process (if no absolute URI is given in the *source* parameter).
csw.xquery.collection.schema: this is an important parameter. Even though every type of XML file can be saved in the eXist database, using the CSW protocol it is only possible to search for one specific type of document at the time. This relates to the unique mapping of so-called *queryables* to the corresponding fragments in the XML document. Even though the XML scheme for the ISO19115 metadata is determined by the ISO19139 standard, in practice there are also other metadata schemes. (For example the ESRI scheme for ISO19115 or the Dublin Core metadata scheme)
By the parameter *csw.xquery.collection.schema* the scheme is defined which is actually used by the CSW server. The value of this parameter is used in the mapping definition. So if

csw.xquery.collection.schema = iso19139 than the mapping is defined by the parameters *csw.xquery.**iso19139**.xxx* (where xxx=subject|title|abstract|...). In principle, the administrator has also the possibility to define his own mapping by mapping a parameter of his choice on a XML fragment.

```
# National Aersospace Laboratory, NLR
# Rob van Swol (vanswol@nlr.nl)
# September 2007

# @(#)csw.properties
#
# xmldb database parameters
# user, password may be left blank or set to guest/guest

csw.xmldb.driver     = org.exist.xmldb.DatabaseImpl
#csw.xmldb.uri        = xmldb:exist://localhost:8080/excat/xmlrpc
csw.xmldb.uri        = xmldb:exist://
csw.xmldb.collection = /db/csw
csw.xmldb.user       =
csw.xmldb.password   =

# optional file name of capabilities document (in WEB-INF/xml)
# default = capabilities.xml
#csw.capabilities.file = my-capabilities.xml

# optional file name of describerecord document (in WEB-INF/xml)
# default = describerecord.xml
#csw.describerecord.file = my-describerecord.xml


# for harvesting administrator and password must be set!
csw.xmldb.admin            = admin
csw.xmldb.admin.password   =

# harvest base uri (should end with a slash(/) !)
# default = file:///pathtocontext/WEB-INF/harvest/
#csw.harvest.base.uri = file:///somepath/to/harvestdir/

# here we select the xml files in the database based on their format
#csw.xquery.collection.schema = iso19115
csw.xquery.collection.schema = iso19139

# ESRI ISO 19115
csw.xquery.iso19115.collection = //Metadata

# mapping paths are relative to collection
```

```
csw.xquery.iso19115.mapping.subject    = /dataIdInfo/descKeys/keyword
csw.xquery.iso19115.mapping.title      = /dataIdInfo/idCitation/resTitle
csw.xquery.iso19115.mapping.abstract   = /dataIdInfo/idAbs
csw.xquery.iso19115.mapping.anytext    = /*
csw.xquery.iso19115.mapping.format     =
/dataIdInfo/idCitation/presForm/PresFormCd/@value
csw.xquery.iso19115.mapping.identifier        = /mdFileID
#csw.xquery.iso19115.mapping.modified  = /mdDateSt
#csw.xquery.iso19115.mapping.modified  = /dataIdInfo/idCitation/resRefDate/refDate
csw.xquery.iso19115.mapping.type       = /mdHrLv/ScopeCd/@value
csw.xquery.iso19115.mapping.boundingbox  = /dataIdInfo/geoBox/westBL,\
                                           /dataIdInfo/geoBox/southBL,\
                                          /dataIdInfo/geoBox/eastBL,\
                                          /dataIdInfo/geoBox/northBL


# ISO 19139
csw.xquery.iso19139.collection = //gmd:MD_Metadata

# mapping paths are relative to collection
csw.xquery.iso19139.mapping.subject    =
/gmd:identificationInfo/gmd:MD_DataIdentification/gmd:descriptiveKeywords/gmd:MD_Keyword
s/gmd:keyword
csw.xquery.iso19139.mapping.title      =
/gmd:identificationInfo/gmd:MD_DataIdentification/gmd:citation/gmd:CI_Citation/gmd:title
csw.xquery.iso19139.mapping.abstract   =
/gmd:identificationInfo/gmd:MD_DataIdentification/gmd:abstract
csw.xquery.iso19139.mapping.anytext    = /*
csw.xquery.iso19139.mapping.format     =
/gmd:distributionInfo/gmd:MD_Distribution/gmd:distributionFormat/gmd:MD_Format/gmd:name
csw.xquery.iso19139.mapping.identifier        = /gmd:fileIdentifier
csw.xquery.iso19139.mapping.modified   = /gmd:dateStamp/gco:Date
csw.xquery.iso19139.mapping.type       =
/gmd:hierarchyLevel/gmd:MD_ScopeCode/@codeListValue
csw.xquery.iso19139.mapping.boundingbox  =
/gmd:identificationInfo/gmd:MD_DataIdentification/gmd:extent/gmd:EX_Extent/gmd:geographi
cElement/gmd:EX_GeographicBoundingBox/gmd:westBoundLongitude,\

/gmd:identificationInfo/gmd:MD_DataIdentification/gmd:extent/gmd:EX_Extent/gmd:geographi
cElement/gmd:EX_GeographicBoundingBox/gmd:southBoundLatitude,\

/gmd:identificationInfo/gmd:MD_DataIdentification/gmd:extent/gmd:EX_Extent/gmd:geographi
cElement/gmd:EX_GeographicBoundingBox/gmd:eastBoundLongitude,\

/gmd:identificationInfo/gmd:MD_DataIdentification/gmd:extent/gmd:EX_Extent/gmd:geographi
cElement/gmd:EX_GeographicBoundingBox/gmd:northBoundLatitude
```

```
# extra queryables
csw.xquery.iso19139.mapping.creator     =
/gmd:identificationInfo/gmd:MD_DataIdentification/gmd:pointOfContact/gmd:CI_ResponsibleP
arty[gmd:role/gmd:CI_RoleCode/@codeListValue = "originator"]/gmd:organisationName
csw.xquery.iso19139.mapping.relation    =
/gmd:identificationInfo/gmd:MD_DataIdentification/gmd:supplementalInformation
```

6.1.3 csw-hosts.xml

The configuration file csw-hosts.xml is used by the eXcat client as well as by the Harvest module. The file contains a list of CSW servers which can be queried from the client.
An example is listed below:

```
<hosts>
  <host id="LOCAL" harvest="no">
    <name>localhost</name>
    <url>http://localhost:8080/excat/csw</url>
  </host>
  <host id="GROEN" harvest="no" method="get">
    <name>groene omgeving</name>
    <url>http://www.groene-omgeving.nl/aimscsw/csw2.0</url>
  </host>
  <host id="NLR" harvest="no">
    <name>eXcat@nlr</name>
    <url>http://geomatics.nlr.nl/excat/csw</url>
  </host>
  <host id="TNO" maxrecords="1" keepfiles="true" harvest="yes">
    <name>eXcat@tno</name>
    <url>http://dinolks03.nitg.tno.nl/excat/csw</url>
  </host>
  <host id="MNP" harvest="yes" method="post" maxrecords="-1" keepfiles="false"
                                             overwrite="false" support-hits="false">
    <name>mnp-CSW2.0</name>
    <url>http://mapserver.mnp.nl/aimscsw/csw2.0</url>
    <constraint language="FILTER">Title LIKE '%'</constraint>
  </host>
  <host id="GEONOVUM" harvest="no">
    <name>Geonovum</name>
    <url>http://geonovum.nitg.tno.nl/geonetwork/srv/en/csw</url>
  </host>
</hosts>
```

For each CSW server (within the xml element *host*) a name and address are specified by the xml elements *name* and *url*, respectively The *host*-element may contain the following attributes:

Mandatory attributes:
Id: a unique identifier
Optional attributes:
Harvest: "yes|no" (default=no => this host cannot be harvested)
Method: "post|get" (default=post => http request method)
Maxrecords: negative|positive value (negative means that all records are harvested)
Keepfiles: "true|false" (default=false => the files in the harvest folder are not saved after they have been stored in the database)
Overwrite: "true|false" (default=false => files already present in the database are not saved (based on the fileID))
support-hits: "true|false" (default=true => use resultType=hits only for requesting the number of records; this value has to equal *false* for an ESRI-type server because this server doesn't return the parameter *numberOfMatchingRecords)*.

Optionally, the element *constraint* can be added containing a constraint (in CQL_TEXT format) to be used when harvesting metadata records. Thereby, the following optional attribute can be used:
language: "FILTER|CQL_TEXT" (default=FILTER => constraint language)

6.1.4 harvest.properties

The file harvest.properties only knows two parameters. With *harvest.base.path* one can change the path under which the received files are (temporarily) saved and with *harvest.csw.host* the address of the collecting CSW server is defined.

```
# @(#)harvest.properties

# file path where metadata records are stored
# default = /WEB-INF/harvest/
# harvest.base.path = /some/path/to/harvest/directory

# csw host where harvested records are stored
harvest.csw.host = http://localhost:8080/excat/csw
```

6.2 A second eXcat instantiation

In the previous section it was mentioned that the eXist database knows no limitations with regard to using different XML schemes, but the eXcat software does have this limitation. It is only possible to search for one type of metadata at a time. However, it is possible to deploy two different instantiations of eXcat simultaneously both using the same database. To this end, the following steps have to be taken (where we will call the second instantiation *atlas*):

1. Define and initiate a servlet via the configuration file WEB-INF/web.xml. This servlet is called via http://localhost:8080/excat/atlas.

```
    <servlet>
        <servlet-name>
            AtlasServlet
        </servlet-name>
        <servlet-class>
            nl.nlr.geomatics.csw.CSWServlet
        </servlet-class>
        <!--load-on-startup>1</load-on-startup-->
        <init-param>
          <param-name>csw.conf</param-name>
          <param-value>/WEB-INF/conf/atlas.properties</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>AtlasServlet</servlet-name>
        <url-pattern>/atlas</url-pattern>
    </servlet-mapping>
```

2. Create a configuration file (e.g. atlas.properties) containing the definition of the mapping of the metadata and an adapted reference to the capabilities document.

```
# @(#)csw.properties
#
#
# xmldb database parameters
# user, password may be left blank or set to guest/guest

csw.xmldb.driver     = org.exist.xmldb.DatabaseImpl
#csw.xmldb.uri        = xmldb:exist://localhost:8080/excat/xmlrpc
csw.xmldb.uri        = xmldb:exist://
csw.xmldb.collection = /db/csw
csw.xmldb.user       =
csw.xmldb.password   =

# optional file name of capabilities document (in WEB-INF/xml)
# default = capabilities.xml
```

```
csw.capabilities.file = atlas-capabilities.xml

# optional file name of describerecord document (in WEB-INF/xml)
# default = describerecord.xml

# for harvesting administrator and password must be set!
csw.xmldb.admin            = admin
csw.xmldb.admin.password   =

# harvest base uri (should end with a slash(/) !)
# default = file:///pathtocontext/WEB-INF/harvest/
#csw.harvest.base.uri = file:///somepath/to/harvestdir/

# harvest base uri (should end with a slash(/) !)
# default = file:///pathtocontext/WEB-INF/harvest/
#csw.harvest.base.uri = file:///somepath/to/harvestdir/

# here we select the xml files in the database based on their format
csw.xquery.collection.schema = dublin-core
# DC
csw.xquery.dublin-core.collection = //simpledc

# mapping paths are relative to collection
csw.xquery.dublin-core.mapping.subject       = /dc:subject
csw.xquery.dublin-core.mapping.title         = /dc:title
csw.xquery.dublin-core.mapping.abstract      = /dct:abstract
csw.xquery.dublin-core.mapping.anytext       = /*
csw.xquery.dublin-core.mapping.format        = /dc:format
csw.xquery.dublin-core.mapping.identifier    = /dc:identifier
csw.xquery.dublin-core.mapping.modified      = /dct:modified
csw.xquery.dublin-core.mapping.type          = /dc:type
csw.xquery.dublin-core.mapping.boundingbox   = /ows:BoundingBox
```

3. Create an adapted capabilities document and place it as *atlas-capabilities.xml* in WEB-INF/xml.

4. Add the Atlas host to *csw-hosts.xml* in order to make the server visible in CSW client.

```
  <host id="ATLAS" harvest="no">
    <name>localhost</name>
    <url>http://localhost:8080/excat/atlas</url>
  </host>
```

6.3 Create and restore a backup

The eXist client offers the possibility to make a backup of the database. When the client is started from the *WEB-INF/bin* folder one can use buttons 7 and 8 (see Fig. 6) to make a backup and restore a backup, respectively. If a backup is made, a compressed (zip) file is automatically generated with the name *eXist-backup.zip*. The location (folder) can be chosen. To restore such a backup, the file first has to be un-zipped. This will result in a new folder *db*. The file *db/__contents__.xml* can now be read and the whole structure in the folder *db* is automatically recovered in the database.

6.4 Use of OpenLayers viewer

When the metadata contains references to layers which can be made visible in a WMS client, it is possible to link the metadata presentation in the CSW client to the supplied OpenLayers viewer. This will work on the condition that in the XML metadata element *MD_DigitalTransferOptions* the URL, the protocol (OGC-WMS) and the name of the layer are defined. Expert users might want to make modifications in the XSL stylesheet *WEB-INF/xsl/metadata-iso19139.xsl* in order to make it work with another viewer. The OpenLayers viewer is installed as web application by the placing the file *viewer.war* in the folder *webapps* of Tomcat.

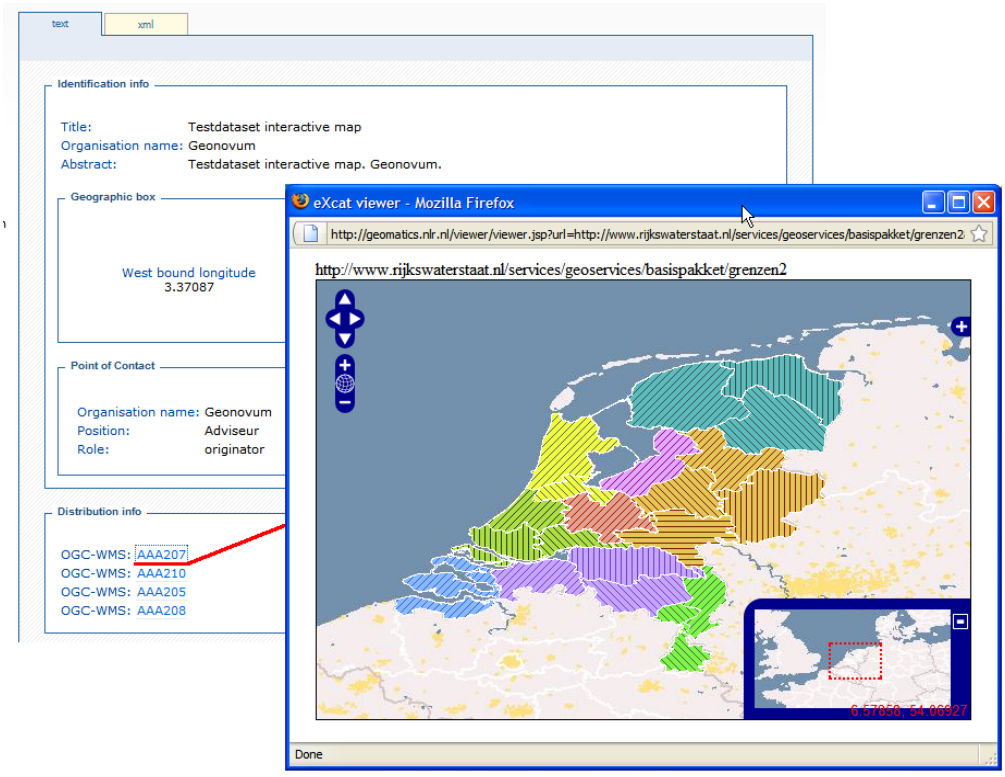


Fig. 13 OpenLayers viewer example

References

[1] *OpenGIS Catalogue Service Specification, version 2.0.2*, OGC 07-006r1, Open Geospatial Consortium Inc., 2007-02-23.

[2] *OpenGIS Catalogue Service Specification 2.0.2 – ISO Metadata Application Profile*, OGC 07-045, Open Geospatial Consortium Inc., 2007-07-19.

[3] eXist, Open Source Native XML Database, http://exist.sourceforge.net.

[4] Geotools, The Open Source Java GIS Toolkit, http://geotools.codehaus.org.

[5] XML Path Language (XPath), http://www.w3.org/TR/xpath.

[6] XQuery 1.0: An XML Query Language, http://www.w3.org/TR/xquery.

[7] OpenLayers, Free Maps for the Web, http://www.openlayers.org.

[8] Nederlands profiel op ISO 19115 voor geografie, versie 1.2, Geonovum, december 2008.