eXows
***eX**tended **O**GC **W**eb **S**ervices*

January 2013
(Build 527 – War 162)

Geoscience for a sustainable Earth
brgm

Revision History

Note: The Release column is composed of two references, the first one is the number of this documentation, and the second one (BxxxWxxx) is the version of eXows described in this documentation (BxxxWxxx means Build xxx War xxx)

Date	**Release**	**Editor**	**Description**
26/04/2011	*1 B303W87*	*François Tertre (BRGM)*	*Document Structure and copy paste from 1GEconnector documentation*
27/04/2011	*2 B303W87*	*François Tertre (BRGM)*	*Adaptation of documentation to eXows Addition of references to external concepts*
02/05/2011	*3 B303W87*	*François Tertre (BRGM)*	*Modification of schema for 1GE Use Case Addition of legends*
04/05/2011	*4 B352W136*	*François Tertre (BRGM)*	*New functions in eXows: • INSPIRE configuration for each eXows configuration • External Vocabularies repositories • SRS calculation • Configuration files outside webapps*
05/05/2011	*5 B370W138*	*François Tertre (BRGM)*	*Change in debug behavior Precision about Layer Group for WMS GetCapabilities*
15/06/2011	*6 B453W150*	*François Tertre (BRGM)*	*Change for setting up debug mode Two new modes: • Mini configuration file • No configuration file*
30/09/2011	*7 B461W151*	*François Tertre (BRGM)*	*Add defaultVersion for getCapabilities*
21/10/2011 26/10/2011	*8 B476W153 B478W154 B481W155*	*François Tertre (BRGM)*	*Change in eXows: • New templates location • New footer mechanism*
02/07/2012	*9 B498W157*	*François Tertre (BRGM)*	*Change in eXows: • Corrected example of templates • Translation available also for XML output (not only HTML) • New fields for "entry" : ○ Autogenerated_id ○ Static_value*
30/08/2012	*10 B515W160 B527W162*	*François Tertre (BRGM)*	*Change in documentation: Change in eXows: • Adding examples of config & templates • Removing of "correspondence table" used for OneGeology Europe project*

Content

eXows Diffusion

Licence

How to retrieve eXows

eXows is available on SourceForge: https://sourceforge.net/projects/exows/

All sources are available on the SourceForge SVN. Some binaries version (.war files) are also available in the file repository of SourceForge. The last version of this documentation can be retrieved from the file repository of SourceForge.*

The Bug Tracker, forum, mailing list of eXows are accessible through the SourceForge project pages.

Functions of eXows

eXows is provided to be installed in front of a standard OGC[1] WMS[2]/WFS[3], in order to:

- Handle a language parameter (as it is required by INSPIRE[4])
- Make the Capabilities file INSPIRE compliant
- Transform simple GML into a complex GML (e.g. GeoSciML[5], EarthResourceML[6]…)
- Handle complex GML attributes in `ogc:filter` (either in WFS `getFeature` requests, or in SLD sent with a WMS `GetMap` requests)

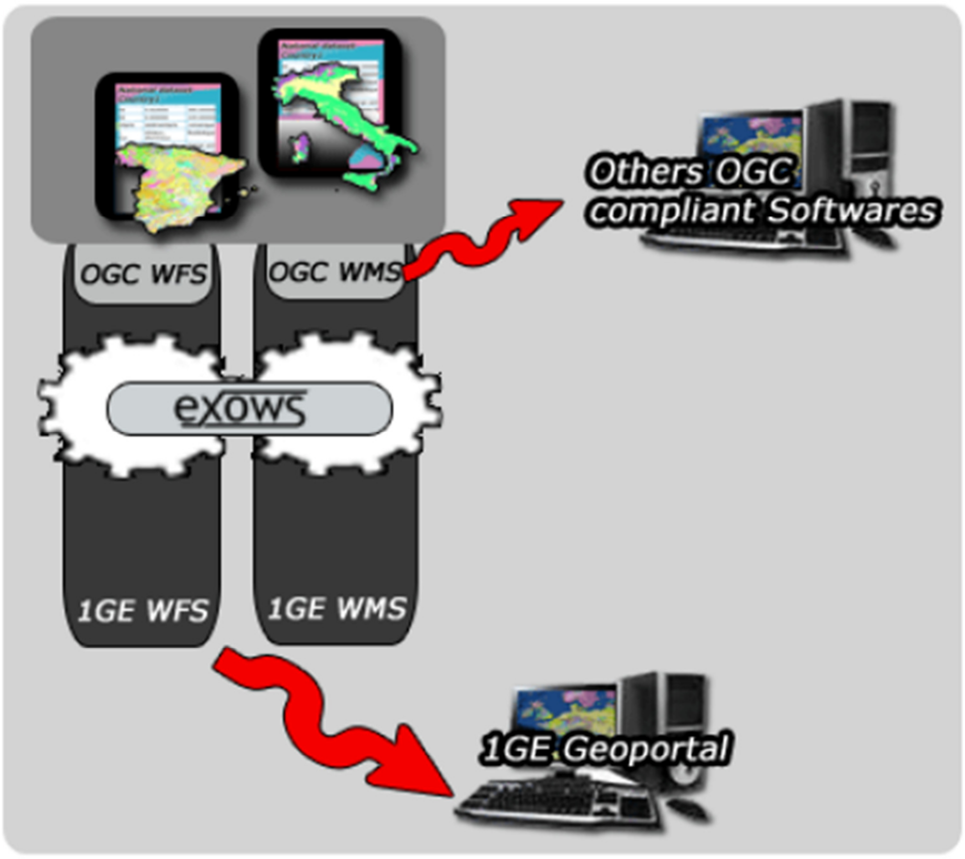

Figure 1 - eXows use in OneGeology Europe

eXows is a Web Application running on Web Server that handle HTTP GET and POST requests (either WMS or WFS requests) and return responses (XML, HTML or image), as a WMS or a WFS service (the requests are not case sensitives).

eXows uses a configuration file defining languages, translations, and another with the INSPIRE tags to be added to the WMS `GetCapabilities` answer, and matching between original layer names and visible layer names (e.g. the original layer name can be `0` and the visible layer name is `Geologic_unit`).

[1] *Open Geospatial Consortium, Inc. ® (http://www.opengeospatial.org/)*
[2] *Web Map Service (http://www.opengeospatial.org/standards/wms)*
[3] *Web Feature Service (http://www.opengeospatial.org/standards/wfs)*
[4] *INSPIRE Directive (http://inspire.jrc.ec.europa.eu/)*
[5] *GeoSciML http://www.cgi-iugs.org/tech_collaboration/geosciml.html*
[6] *EarthResourceML http://www.earthresourceml.org/*

getCapabilities (WMS or WFS)

Note: the repository is a folder containing the translated GetCapabilities files. The path of this folder is configured in the eXows' configuration file.

When the parameter REQUEST is equal to "GetCapabilities" and the parameter SERVICE is equal to "WMS" or "WFS"

1. eXows retrieves the parameters `SERVICE`, `VERSION` and `LANGUAGE`.
2. Then it searches in the repository for a file with a name according to the service, version and language requested. For example, if the service is `WMS`, the version is `1.1.1` and the language is `fra`, then it searches for a file named `getCapabilities-wms-1_1_1-fra.xml`. If the file doesn't exist with the requested language, eXows will search for a file with the default language (defined in the configuration file).
3. If the file exists, eXows returns its content.
4. Else, it transfers the request to the standard WMS/WFS itself and modifies the response.
   - Add INSPIRE tags (only for a WMS service)
   - Add namespace (`inspire_common` and `inspire_vs`) (only for WMS service)
   - Translate title and abstract tags
   - Add some SRS and their BBOX defined in the configuration file
   - Replace online resources (to display eXows url instead of the original WMS)
   - For each layer configured:
     - Translate title and abstract tags
     - Replace the layer name
     - For each style, replace the legend URL (this mechanism allows to define various legend url according to the language, so to define translated legends).
   - Remove layers which are not defined in the configuration file. **Warning: if a configured layer is owned by a group, and this group is not configured in eXows, the whole group is removed, and your configured layer will be removed too. To avoid this situation, configure (at least) the group layers in eXows configuration.**

getMap (WMS)

When the parameter REQUEST is equal to "GetMap".

In this request, the important parameters for eXows are `LAYERS` and `SLD`. These parameters are modified accordingly to the configuration file.

1. eXows reads the `LAYERS` parameter and replace each layer name by its name on the original WMS.
   - *Thanks to this process, a local WMS could be configured with a layer named* `0` *and eXows transforms another layer name such as* `onegeologyeuropegeologicunits` *into "0". In that case* `onegeologyeuropegeologicunits` *is the visible layer name, and 0 is the internal layer name.*
2. If the request contains a `SLD_BODY` or a `SLD` parameter, eXows handles this parameter and process a specific treatment.
   a. In case of `SLD`, it retrieves the `SLD` file pointed by the `SLD` parameter (which is a url)
   b. After that, in both case (`SLD_BODY` or `SLD`), it separates all `NamedLayer` parts and deals with.
   c. In a first step, the connector changes the name of the layer in the `<Name>` markup to its name on the original WMS.

d. Then, it modifies the filter of each `NamedLayer` to change all the complex GML path to simple GML attributes.
e. Finally, the modified content of the SLD is saved to a file accessible from the Internet so the original WMS can request it using a SLD parameter.

- *Example:*
  The following GeoSciML attribute is replaced by the local attribute `lower_age`:
  `gsml:MappedFeature/gsml:specification/gsml:GeologicUnit/gsml:preferredAge/gsml:GeologicEvent/gsml:eventAge/gsml:CGI_TermRange/gsml:lower/gsml:CGI_TermValue/gsml:value`

  The following ogc:filter

```
<ogc:Filter>
<ogc:PropertyIsEqualTo>
<ogc:PropertyName>gsml:MappedFeature/gsml:specification/gsml:Ge
ologicUnit/gsml:preferredAge/gsml:GeologicEvent/gsml:eventAge/g
sml:CGI_TermRange/gsml:lower/gsml:CGI_TermValue/gsml:value</ogc
:PropertyName>
<ogc:Literal>urn:cgi:classifier:ICS:StratChart:2008:MiddleJuras
sic</ogc:Literal>
</ogc:PropertyIsEqualTo>
</ogc:Filter>
```

  Is replaced by

```
<ogc:Filter>
<ogc:PropertyIsEqualTo>
<ogc:PropertyName>lower_age</ogc:PropertyName>
<ogc:Literal>urn:cgi:classifier:ICS:StratChart:2008:MiddleJurass
ic</ogc:Literal>
</ogc:PropertyIsEqualTo>
</ogc:Filter>
```

3. The new request (with `SLD_BODY` and `LAYERS` updated) is sent to the local WMS, and then its response is returned.

getFeatureInfo (WMS)

When the parameter REQUEST is equal to “GetFeatureInfo” and the parameter SERVICE is equal to “WMS”

1. eXows reads the `LAYERS` parameter and replace each layer name by its name on the original WMS.
2. Depending on the parameter `INFO_FORMAT`, three kinds of process can occur.
   a. The `INFO_FORMAT` is `application/vng.ogc.gml` or `text/xml`, eXows requests GML and then transforms the response into complex GML accordingly to the configuration (mapping of GML attributes/complex GML path) and the template.
   b. The `INFO_FORMAT` is `text/html` and in the configuration file a parameter precises that html must be intercepted (recreated); exows requests GML and then transforms the response into HTML accordingly to the configuration (mapping of GML attributes/complex GML path) and the HTML template.
   Note: the HTML outputted eXows can be localized.
   c. The `INFO_FORMAT` is another format or the `INFO_FORMAT` is `text/html` but the configuration precises that html won’t be intercepted (recreated), the connector requests

the original WMS with the same `INFO_FORMAT` that the request it received, and transmit directly the response without any change.

getFeature (WFS)

When the parameter REQUEST is equal to "GetFeature".

1. eXows reads the type name parameter (`TYPENAME`) and makes a list of type name to request.
2. For each requested type name, it creates a request to retrieve the GML from the original WFS:
   a. If the request has a filter, it tries to change the parameters (value of the `PropertyName`) accordingly to the mapping given in configuration. If the complex GML paths are not found in the mapping for this type name, the connector lets unchanged the filter.
   b. If there is a `MAXFEATURES` parameter, it checks the number of features already retrieved and set the `MAXFEATURES` of the current request to the number of features that let to request.
   c. It calls the original WFS with this request and then process a simple GML to complex GML transformation on the response. Depending of the number of features returned and the `MAXFEATURES` requested parameter, eXows continues or not with the other type name.
3. eXows concatenates the response from all the WFS after transforms them to complex GML and then sends it to the client.

Other cases

If eXows receives any other request, it is transferred to the WMS or the WFS. The only transformation is the layers or the typenames parameter (modified from the external name to the internal one), as described in the previous operations.

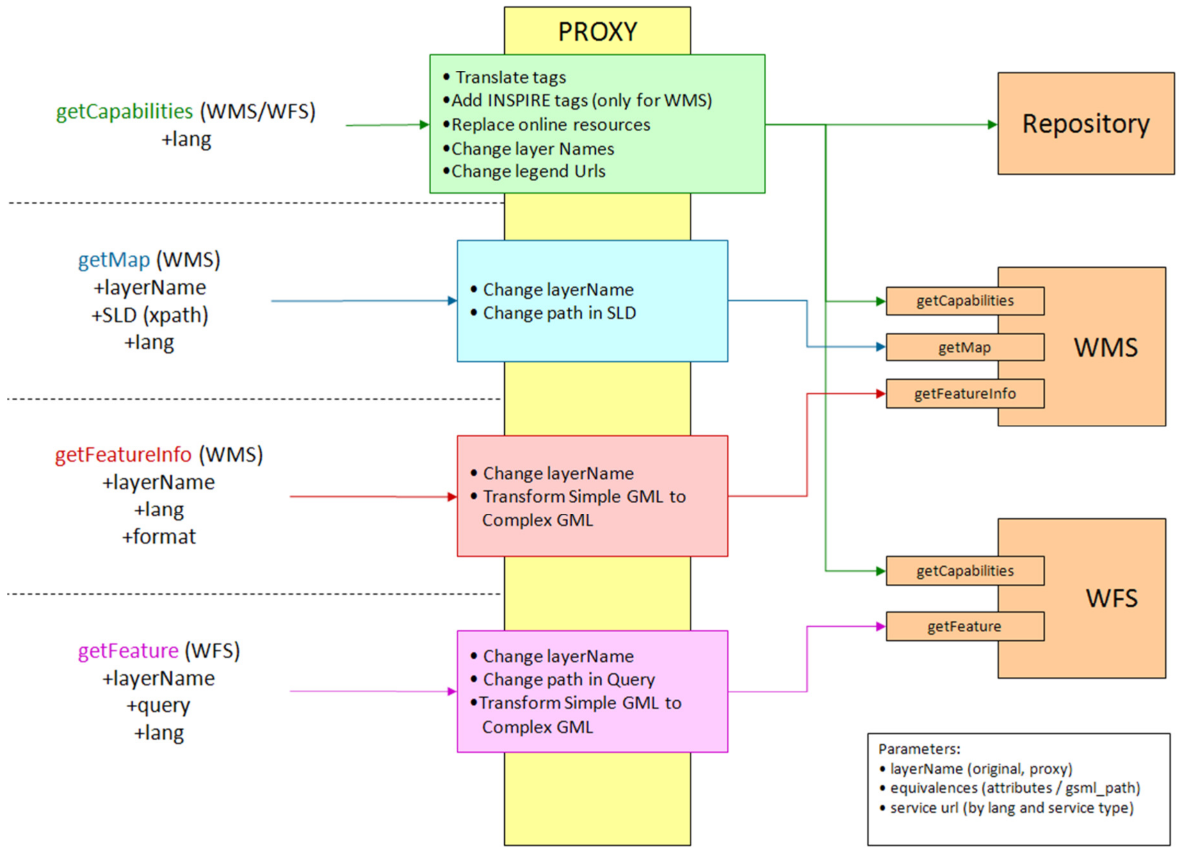

Figure 2 - Architecture

Installation, Configuration, logs and test

Installation

eXows is a war that has to be copied in the `webapps` directory of any Java Servlet engine, such as Tomcat[7].

We consider here that the deployment folder is `<tomcat webapps>/eXows`. In the following explanation we call it `/eXows`.

Location of configuration files

The location of the configuration files is defined in the web.xml file, you can have only one location for the configuration files.

You must change the `pathConfigs` entry of web.xml to change the location of the configuration files.

```
	<env-entry id="pathConfigs">
		<description></description>
		<env-entry-name>param.pathConfigs</env-entry-name>
		<env-entry-type>java.lang.String</env-entry-type>
		<env-entry-value>/exowsConfig/</env-entry-value>
	</env-entry>
```

[7] *Apache Tomcat (http://tomcat.apache.org/)*

We will consider afterwards that the `pathConfigs` is `/exowsConfig/` as previously defined in the web.xml.

Some example of configuration files are furnished inside the war file, they are located in the folder `exowsConfig` (`<tomcat webapps>/eXows/exowsConfig/`).

Single project configuration

The simple case is when eXows only delivers one WMS and/or one WFS.

Configuration file: `/exowsConfig/exowsConfiguration_default.xml`

Configuration of INSPIRE extended capabilities:
`/exowsConfig/exowsConfiguration_default_Inspire.xml`
Content of this file will be included in the GetCapabilities response after the `UserDefinedSymbolization`.

To access the service: `http://<yourserverandport>/exows` will act as a standard OGC web service (WMS and/or WFS)

Have a look to "Configuration file structure" to have more information about this configuration file.

Multi projects configuration

One single installation of eXows can implement several WMS and/or WFS, using several configuration files.

Configuration files: `/exowsConfig/exowsConfiguration_<nameofconfiguration>.xml`

Configuration of INSPIRE extended capabilities:
`/exowsConfig/exowsConfiguration_<nameofconfiguration>_Inspire.xml`
Content of these files will be included in the GetCapabilities responses, after the `UserDefinedSymbolization`.

To access the services:
`http://<yourserverandport>/eXows?CONFIG=<nameofconfiguration>`
or: `http://<yourserverandport>/eXows/<nameofconfiguration>/?`

Have a look to "Configuration file structure" to have more information about this configuration file.

Mini configuration file

In some very special case, eXows allows a mini configuration; in this case, eXows requires an "URL" parameter in the call of the service. This URL parameter must contain a `map=something.map` that will be used to retrieve the location of the mini configuration file. In other terms, this mini configuration mode is dedicated to a use of eXows with MapServer (MapServer use a mechanism with a parameter `map=` in the url to retrieve its configuration file. So eXows will retrieve the URL of the original server, with its own map parameter and will use this parameter to retrieve its own mini configuration file).

This mini configuration mode only allows the user to define the list of layers he wants to keep in the final service and the content of the INSPIRE tags. Accordingly to this list of layers, the GetCapabilities file will contain only these layers and the WMS won't be able to respond to request to other layers.

Have a look to "Mini configuration file structure" to have more information about this mini configuration file.

No configuration file

In some very special case, eXows allows no configuration, in this case, eXows requires an "URL" parameter in the call of the service (ex: `http://<yourserverandport>/eXows?URL=http://myproxiedservice/something`). This configuration allows eXows to be in front of a WMS to add some INSPIRE information in the GetCapabilities.

In this case, the capabilities of eXows are set at WMS 1.3.0 with support of English and eXows is the proxy of the server passed in URL. eXows doesn't translate any name or style of layers, it just transmits the request to the original server. eXows transforms the GetCapabilities to add the INSPIRE tags. These tags come from the `eXowsConfiguration_default_Inspire.xml`.

Configuration file structure

The configuration file is divided in two parts, WMS and WFS parts. eXows can act like WMS, WFS or both, depending which part of configuration is available.

But some parts of the configuration are not dependent of the type of service.

- Vocas – this part defines the location of the resource (vocabularies) that can be used for the translation of URIs.
    - `default` – the content of this markup is the link to the repository where the Vocabularies are. The Vocabularies must be in some XML files, so eXows search in this repository some files with the good name. E.G. if the content of this markup is `http://myserver/myvocabs/`, eXows will search for the vocabulary `something`, a file at the URL : `http://myserver/myvocabs/something.xml`. So you can use a database like eXist to have some more complex mechanism like `http://myserver/concept.xql?collection=/db/somecollection&request=getVocabulary&Vocabulary=`, then eXows will request `http://myserver/concept.xql?collection=/db/somecollection&request=getVocabulary&Vocabulary=something.xml`
    - `specialized` – this is used to precise the vocabularies that cannot be found in the 'default' repository.
        - `Vocaname` – this is the name of the Vocabulary.

WMS/WFS parts:

- In case of WMS, an extra parameter is `interceptHTML`, it will be used to replace the current `getFeatureInfo` in format `text/html` to a custom `getFeatureInfo` generated by eXows using the GML response
- For both WMS and WFS, a parameter `defaultVersion` précises the default version of the Capabilities (e.g. : 1.3.0)
- `repository`
    - The repository is not mandatory, but allows the connector to have a caching system for the getCapabilities file, and to have multilingualism for the getCapabilities if the original server doesn't provide multilingualism.
    - The value of this markup is a path to the folder where the cached getCapabilities XML files are.
    - The cached getCapabilities XML files need to be named like `getCapabilitieswfs-1_0_0-fra.xml`, `getCapabilities-wms-1_1_1-eng.xml` and so one.

- `serverList`
  - The ServerList contains a list of servers (can be only one server) with the proposed language and type of server (mapserver, esri, geoserver, other). The reserved word default precise if this server will be used as a default server.
  - `server`
    - `lang` – this attribute precise the language of the server (only one language by server, to add another language to a server, just duplicate the entire markup server)
    - `default` – this attribute precise if the current server will be used as a default server
    - `type` – this attribute precise the type of the server (`mapserver`, `geoserver`, `esri`, `other`)
      Warning: for the last version of ESRI Map Server, the type of Server for WFS must be set up to `other`.
    - `url` – this markup is used to configure the URL to accede the server.
    - `title` – title in, the configured language. This attribute is not mandatory if a repository with predefined capabilities is available for this language.
    - `abstract` – abstract in the configured language. This attribute is not mandatory if a repository with predefined capabilities is available for this language.
    - `description` – description in the configured language. This attribute is not mandatory if a repository with predefined capabilities is available for this language.
- `LayerList`
  - The LayerList contains a list of Layer (it can be only one layer) with the matching with the layer name from original server (or the layers names if there is multilingualism), and with the mapping between the attributes from the GML from the original server and the attributes from the output GSML.
  - `Layer`
    - `Name` – this attribute precise the name showed by eXows to design the layer. The translation from the original name to this name is automatic in the getCapabilities and in all requests to eXows.
    - `SRSList/SRS` – these markups are used to precise some available SRS for the WMS Layer. eXows will add in the getCapabilities the SRS (CRS for WMS1.3.0) and the BoundingBox. For the BoundingBox, eXows calculates the BBOX with the LatLonBoundingBox (or the EX_GeographicalBoundingBox) of the Layer, if this BBOX doesn't exist, eXows takes the BBOX of the Service, and if this BBOX doesn't exist too, eXows use an arbitrary BBOX -180,-90,180,90. **Be careful, it's not sure that eXows is capable to handle all SRS.**
    - `templates/template` – these markups are used to precise what are the template to use for this layer, at this stage only two types of template can be defined, one for the complex GML (`format="application/vnd.ogc.gml"`) and one for the HTML (`format="text/html"`). With this mechanism, some different templates can be used for each layer served by the connector (e.g. a special template for lithology, another for ages…). *For more information about templates, please read the part 2.5.*
    - `originals/original` – these markups are used to match to the original name (from the original server). Multi markups can be used as you can have one layer by language and by style.

- About the styles parameter of the WMS request, when more than one layer is requested in the same request, if the number of layers is bigger than the number of styles, the connector sets all the styles to `default`.
- Example of use, one layer, with two accepted language, and two accepted styles, but the two styles are related to the same layer in the original WMS:

```
<originals>
      <original lang="eng" style="default" url="http://legendUrl.png">
            OGE_1M_surface_GeologicUnit
      </original>
      <original lang="fra" style="default" url="http://legendUrl.png">
            OGE_1M_surface_GeologicUnit
      </original>
      <original lang="eng" style="age" url="http://legendUrl.png">
            OGE_1M_surface_GeologicUnit
      </original>
      <original lang="fra" style="age" url="http://legendUrl.png">
            OGE_1M_surface_GeologicUnit
      </original>
</originals>
```

- Another example of use, one layer presented by the connector, in front of two layers that accept two languages. The first layer is called when the default style is called. The second one is called when the age style is called, and it will request the layer named `1` of the original WMS, with the default style:

```
<originals>
      <original lang="eng" style="default" url="http://legendUrl.png">0</original>
      <original lang="ita" style="default" url="http://legendUrl.png">0</original>
      <original lang="eng" style="age" original_style="default"
            url="http://legendUrl.png">1</original>
      <original lang="ita" style="age" original_style="default"
            url="http://legendUrl.png">1</original>
</originals>
```

- `languages/title` and `languages/abstract` : these markups are used to translate the layer title and the layer abstract in the requested language :

```
<languages>
      <title lang="eng">BRGM Superficial geology</title>
      <abstract lang="eng"> BRGM Superficial geology</abstract>
      <title lang="fra"> BRGM Géologie superficielle</title>
      <abstract lang="fra"> BRGM Géologie </abstract>
</languages>
```

- `GMLMarkups/GMLMarkup` – these markups precise, in the GML responded by the original server (for GetFeatureInfo or for GetFeature), the markup that starts/finishes a feature, this markup can depend of language requested.
- `GeoSciML_Mapping`
  - The mapping allows eXows to transform a GML to a complex GML. It precise the fields to use from the GML to fill the complex GLM.
  - The mapping is composed of `entry` and `group`
    - entry contains the `gsml_uri` (uri to fill in the complex GML) and one of the following possibilities:
      - `attribute` (name of the field from the GML)
      - `static_value` (a static value settled up in the configuration file)

        - `autogenerated_id` (an id that will be generated automatically – this id will be different each time – two different format of ID can be generated: UUID, UID, the format must be precised as value of the markup).
    - `group` contains the `gsml_path` of the group, and some `entry`, a group represents some attributes that are linked in the GSML.

Mini configuration file structure

The mini configuration file is divided in two parts, the LayerList and the Inspire part.

The LayerList contains a list of Layer with name. The name corresponds to the name of the layers on the original server. This list is just used by eXows to hide some layer in the GetCapabilities and to refuse requests on the layers which are not in the list.

The Inspire part contains the INSPIRE tags to be added in the GetCapabilities of this specific configuration.

Templates

The templating mechanism of eXows is based on the Apache Velocity Project[8].

The templates are located in your configuration path:

`<pathConfigs>/templates`.

Some example of templates files are furnished inside the war file, they are located in the folder `exowsConfig/templates` (`<tomcat webapps>/eXows/exowsConfig/templates/`).

Additionally to this, a file with the extension `*.header` is needed as header of the template. This file must have the same name than the template (except the extension), another file with the extension `*.footer` is needed as footer of the template. This file must have the same name than the template (except the extension).

Three variables are available in the templates:

- `$feature`
    - This variable represents the current feature read by eXows. This variable proposes four ways to retrieve attributes:
        - `$feature.getGmlValueFromGSMLxpath('<GSML_path>’)` use this way if you want retrieve a single attribute using complex GML path (the complex GML path is the path used in the configuration file),
        - `$feature.getGmlTranslatedValueFromGSMLxpath(‘<GSML_path>’,$lang)` use this way if you want retrieve a single attribute translated in the current requested language,
        - `$feature.getGmlListValuesFromGSMLxpath('<GSML_path>’)` use this way if you want to retrieve a list of attribute using complex GML path,
        - `$feature.getGroupFromGSMLxpath(‘<GSML_path>’)` use this way if you want to retrieve list of attributes grouped. To access an attribute of the group, you need to use the path of this attribute in the group. You will able to access your attributes by `$<mygroup>.get('<GSML_path>’)`
- `$lang`

[8] *Apache Velocity Project* (http://velocity.apache.org/)

    - This variable represents the language requested by the user, it's used to translate the label and the urn in case of `getFeatureInfo` html
- `$vocabularies`
    - This variable proposes a translation function that can be used to translate an urn in the current language.
    - The function can be used like this example: `$vocabularies.translate($geologicHistoryGeologicEvent.get('gsml:name/gsml:CGI_TermValue/gsml:value'),$lang)`

The template can contain some values that are not dependant of the feature of the original server. But in this case, these complex GML paths won't be available for filtering.

Examples:

```
$feature.getGmlValueFromGSMLxpath('gsml:MappedFeature/gsml:specification/gs
ml:GeologicUnit/gml:description')
```

The previous example retrieves the value of the attribute with the complex GML path `gsml:MappedFeature/gsml:specification/gsml:GeologicUnit/gml:description`

```
$feature.getGmlTranslatedValueFromGSMLxpath('gsml:MappedFeature/gsml:specif
ication/gsml:GeologicUnit/gsml:preferredAge/gsml:GeologicEvent/gsml:eventAg
e/gsml:CGI_TermRange/gsml:lower/gsml:CGI_TermValue/gsml:value',$lang)
```

This example retrieves the value of the attribute of the lower age of the geologic event translated in the requested language `$lang`.

```
#foreach( $metamorphicFacies in
$feature.getGmlListValuesFromGSMLxpath('gsml:MappedFeature/gsml:specificati
on/gsml:GeologicUnit/gsml:metamorphicCharacter/gsml:MetamorphicDescription/
gsml:metamorphicFacies/gsml:CGI_TermValue/gsml:value') )
    <gsml:metamorphicFacies>
        <gsml:CGI_TermValue>
            <gsml:value codeSpace="http://www.cgiiugs.org/uri">
                $metamorphicFacies
            </gsml:value>
        </gsml:CGI_TermValue>
    </gsml:metamorphicFacies>
#end
```

This example retrieves the list of value of the attributes of the metamorphic facies and allows the user to retrieve them through the `$metamorphicFacies` variable. For more information about how to use `foreach` loop and variable, take a look to the velocity user manual[9].

```
#foreach( $compositionPart in
$feature.getGroupFromGSMLxpath('gsml:MappedFeature/gsml:specification/gsml:
GeologicUnit/gsml:composition/gsml:CompositionPart') )
    <gsml:composition>
        <gsml:CompositionPart>
            <gsml:role codeSpace="http://www.cgiiugs.org/uri">
                $compositionPart.get('gsml:role').get(0)
            </gsml:role>
            <gsml:lithology
  xlink:href="$compositionPart.get('gsml:lithology@xlink:href').get(0)"/>
            <gsml:proportion>
                <gsml:CGI_TermValue>
                    <gsml:value codeSpace="http://www.cgiiugs.org/uri">
```

[9] *Velocity User Manual* (http://velocity.apache.org/engine/releases/velocity-1.6.2/user-guide.html)

```
      $compositionPart.get('gsml:proportion/gsml:CGI_TermValue/gsml:value')
       .get(0)
                    </gsml:value>
                </gsml:CGI_TermValue>
            </gsml:proportion>
        </gsml:CompositionPart>
    </gsml:composition>
#end
```

This example retrieves a group of values that are linked together. For this example, the group is about `CompositionPart`. To retrieve the value of each attribute that compound the group, you need to use the GSML path relative to the group.

Logs

The logs are presented in two types, the first type is a classical Tomcat log, and the second is an HTML log that can be consulted from the outside of the server. (`http://<yourserver>/eXows/logs.jsp`). Only three days of log are kept, and the log available at the previous address is the log of the current day.

The content of these logs depends of the requested debug mode of eXows. This debug mode is settled up in the web.xml file, you can have only one mode of debug for one instance of eXows. To activate the debug mode, just put the value `true` to this attribute.

You must change the `debugMpde` entry of web.xml to change the behavior of the logging system.

```
      <env-entry id="debugMode">
            <env-entry-name>param.debugMode</env-entry-name>
            <env-entry-type>java.lang.Boolean</env-entry-type>
            <env-entry-value>false</env-entry-value>
      </env-entry>
```

Tests

A webpage presents some useful link to perform tests on the present configuration of eXows. This page is automatically generated from the getCapabilities (for the spatial extent) and the Configuration file (for the layers name, the styles and languages).

This page is accessible for the default configuration at the following address:
`http://<yourserver>/eXows/autotest.jsp`

For other configurations, the page can be found at:
`http://<yourserver>/ eXows /<configuration>/autotest.jsp`
`http://<yourserver>/ eXows /autotest.jsp?CONFIG=<configuration>` where `<configuration>` is the name of the configuration.

This page presents also the eXows' version, with its build number (and the date of last build) and the war number (and the date of last war creation), please communicate these information for all question about eXows.

Technologies used

eXows is a JAVA web application and uses some libraries:

- From the commons Apache project (http://commons.apache.org/):
  - BeanUtils (1.7.1)

- Collections (3.1)
- Digester (1.7)
- Lang (2.1)
- Logging (1.1)

- Apache Log4j libraries (http://logging.apache.org/log4j/)
- Apache Velocity librairies (http://velocity.apache.org/)

Functional point of view

eXows is divided into several components.

The first component is the entry point, `ConnectorServlet`. This servlet checks the input parameters and dispatches the request to the second component according to the request parameter. eXows can manage **getCapabilities**, **getMap**, **getFeatureInfo** and **getFeature** requests, other requests are just transmitted to the original server.

The second component retrieves the parameters from the `ConnectorServlet`. This component is composed of five different parts, one by type of request available in eXows, the **getCapabilities**, the **getMap**, the **getFeatureInfo**, the **getFeature** and the other requests that are just transmitted to the original server.

The **getCapabilities** part proposes a `loadCapabilities` function that delivers the getCapabilities of the server or the content of a getCapabilities file located in the repository. In a first time, it retrieves the value of the `SERVICE`, `VERSION` and `LANGUAGE` parameters. According to these three parameters, eXows retrieves from the configuration file the location of the repository (if exists), the URL of the original server and it checks if the language is available for this service, and if not, it changes the language to the default one. After that, eXows looks into the repository for a file with a name like `getCapabilities-<SERVICE>-<VERSION>-<LANGUAGE>.xml`. If the file exists, eXows retrieves and returns it to the user; otherwise, eXows just requests a getCapabilities to the original server, modifies the response (translate tags, add Inspire extended capabilities in case of a WMS) and then returns the response to the user.

The **getMap** part proposes a `loadMap` function that delivers the requested map from the original server. In a first time, it retrieves the value of the `LANGUAGE`, `STYLE` and `FORMAT`. After that, eXows goes through all the parameters to process specific treatment. For `LAYERS` parameter, eXows translates the name of layers from the displayed name to the original name in the original server, according to the `LANGUAGE` and `STYLE` parameters. For the `SLD` and `SLD_BODY` parameters, the connector acts in the same way. In case of `SLD`, eXows starts by retrieved the `SLD` and keeps it name because this is the name that will be reused to locally store the new `SLD`. In case of `SLD_BODY`, it retrieves the parameter `SLD_BODY` and generates a new name that will be used to store locally the new SLD; the name is generated by using a timestamp. When the content of the `SLD/SLD_BODY` is retrieved, eXows can modify it to reflect the real configuration of the original server. For this, for each part of the SLD (each `NamedLayer`), eXows will transform the content. The first steps are to retrieve the name and its translated name if it exists. The third step is to retrieve the transformation rules for translating GML to complex GML. The fourth step is to change the name of the layer to the original one. The fifth step is to change the filter according to the rules of translation GML to complex GML, this step consists in changing the `PropertyIsEqualTo` and `PropertyIsLike` attributes to reflect the original ones (the GML ones), due to this, eXows can add some alternative rules (Or) if one complex GML attribute is linked to more than one GML attribute. And then, as a last step, eXows closes the filter and the `NamedLayer`. It continues for each layer described in the SLD. Then it stores locally the SLD and requests the original server with the new request (that has original layer name, SLD modified according to the original server configuration…) and retrieves the response and returns it to the user.

The **getFeatureInfo** part reposes on another layer that will be explain later. This part proposes a `loadFeatureInfo` function that delivers the requested FeatureInfo in the good format; this format can be plain text, html (directly outputted from server or created from GML) or complex GML. In a first time, eXows retrieves the parameters `LANGUAGE` and `INFO_FORMAT`; and then it goes through the list of parameters. When the connector processes the `QUERY_LAYERS` parameter, it retrieves all layer names in an array that will be used later. After that, the connector writes the header of the response according to the `INFO_FORMAT` given in the request. And then eXows goes through the list of layer names available in the `QUERY_LAYERS`; for each layer name it search the real layer name (the name of the layer in the original server). In the next step, the connector checks the kind of `INFO_FORMAT` given in the request, and according to this format, it calls a specific function that can be for just plain text or a HTML redirection (directly respond the HTML answer of the original server), a HTML response with a new HTML creation or a complex GML response with complex GML creation. Additionally to this, eXows checks the type of the original server, and doesn't call the same function for ESRI than for GeoServer or MapServer. The functions that can be called are `GFI.redirect`, `ESRIml2Out.ToHTML`, `ESRIml2Out.ToGSML`, `Gml2Out.ToHTML` and `Gml2Out.ToGSML`; they will be explained in the next layer.

The **getFeature** part reposes on another layer (the same layer than for **getFeatureInfo**). This part proposes a `loadFeature` function that delivers the requested Feature in complex GML. In a first time, eXows retrieves the parameter `LANGUAGE` and the server associated to this language. After that, the connector goes through the parameters. It retrieves the list of requested layer name (`TYPENAME`), the `MAXFEATURE` and the `FILTER`. Next it goes through the list of layer name (`TYPENAME`) and for each layer name; it searches the real name (the name of the layer on the original server), according to the requested language and it starts to create the request with this information. If a `MAXFEATURE` is settled up, eXows checks if it's reached, and if not, it changes the current MaxFeature to the new value (number of features that let to retrieve for example), and complete the request. The last step for completing the request is to treat the filter. For this step, eXows calls a function `FilterGSML2GML.transform` that processes transformation for the filter; this function will be explained later. After that, the connector checks if some features are requested (if the `TYPENAME` parameter contains more than one layer name and if the `MAXFEATURE` is settled up, maybe the maximum number of feature requested is already reached), and if so, it checks the kind of original server, ESRI, GeoServer or MapServer, and calls the good function, `ESRIml2Out.ToGSML` or `Gml2Out.ToGSML`.

The last part is the **transmitter** part. This part proposes just a transmit function that allows eXows to transmit the request it received to the original server (depending of the language requested). eXows just retrieved the address of the original server, and then goes through the parameters to change the `LAYERS`, `LAYER` and `TYPENAME` parameters to reflect the layer names of the original server. And then the requested is created, sent to the original server and the response is retrieved and displayed to the user.

As a part of the third component, the `FilterGSML2GML.transform` function transforms a filter with complex GML paths to a filter with simple GML attributes according to the original server. This function reads the filter passed in parameters character by character and search for all GML filter markup like `PropertyIsEqualTo`, `PropertyIsLike` or `BBOX`. After that, the connector checks if the property available in this part of the filter matches to a complex GML path available in the configuration file; if yes, eXows recreate this part of the filter, just by changing the `PropertyName` if the complex GML path matches only one simple GML attribute, or by creating new parts of filter if the complex GML path matches more than one simple GML attribute. For example, if the configuration is:

```
<entry>
      <attribute>gu_composition_1_lithology</attribute>
      <gsml_uri>gsml:lithology@xlink:href</gsml_uri>
</entry>
<entry>
      <attribute>gu_composition_2_lithology</attribute>
      <gsml_uri>gsml:lithology@xlink:href</gsml_uri>
</entry>
```

and the part of the filter is:

```
<ogc:PropertyIsEqualTo>
      <ogc:PropertyName>gsml:lithology@xlink:href</ogc:PropertyName>
      <ogc:Literal>Sand</ogc:Literal>
</ogc:PropertyIsEqualTo>
```

so eXows will create a new part of filter like this:

```
<ogc:Or>
      <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>gu_composition_1_lithology</ogc:PropertyName>
            <ogc:Literal>Sand</ogc:Literal>
      </ogc:PropertyIsEqualTo>
      <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>gu_composition_2_lithology</ogc:PropertyName>
            <ogc:Literal>Sand</ogc:Literal>
      </ogc:PropertyIsEqualTo>
</ogc:Or>
```

Please note that if in the template the complex GML path corresponds to a constant, the filter cannot acts on it.

As another part of the third layer, the functions that come from `ESRIml2Out` and `Gml2Out`. These functions transform the GML (real GML or XML responded by ESRI server) into another format, HTML or complex GML.

For the `ESRIml2Out` part, the HTML and the complex GML parts act in the same way. First, they create the template to be used (a HTML template or a complex GML template), then they read the GML line by line and retrieve the features. For each feature, they extract the attribute of this feature and then create a new feature/html part using the template, and finally, they send the part created to the user.

For the `Gml2Out` part, the HTML and the complex GML parts act in the same way. First, they create a template to be used (a HTML template or a complex GML template), then they read the GML line by line in order to find the beginning of a new feature, in case of GeoServer, the GML can be on only in one line so features are searched in this line. For each feature, they extract the attribute of this feature and then create a new feature/html part using the template, and finally, they send the part created to the user.

Filters and SLD

Filters

eXows only accepts the following Operators in filter:

- Logical Operators
  - And
  - Or
  - Not
- Spatial Operators
  - BBOX
    - GML2 (i.e. using Box)

        - GML3 (i.e. using Envelope)
- Comparison Operators
    - PropertyIsEqualTo
    - PropertyIsLike

All other operators will be removed from the filter when the request is sent to the original service.

Filters need to have the good namespaces (ogc).

SLD

Except for the filter part and the name of the layer concerned by the SLD, eXows doesn't change the content of the SLD. To have a better compatibility with MapServer, ESRI Web Map Server and so one, we recommend having a SLD well-formed with the good namespaces, like the following example:

```
<sld:StyledLayerDescriptor version="1.0.0"
xmlns="http://www.opengis.net/ogc" xmlns:sld="http://www.opengis.net/sld"
xmlns:ogc="http://www.opengis.net/ogc"
xmlns:gml="http://www.opengis.net/gml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sld
http://schemas.opengis.net/sld/1.0.0/StyledLayerDescriptor.xsd">
<sld:NamedLayer>
<sld:Name>OGE_1M_surface_GeologicUnit</sld:Name>
<sld:UserStyle>
<sld:FeatureTypeStyle>
<sld:Rule>
<ogc:Filter>
<ogc:PropertyIsEqualTo>
<ogc:PropertyName>gsml:MappedFeature/gsml:specification/gsml:GeologicUnit/g
sml:preferredAge/gsml:GeologicEvent/gsml:eventAge/gsml:CGI_TermRange/gsml:l
ower/gsml:CGI_TermValue/gsml:value</ogc:PropertyName>
<ogc:Literal>urn:cgi:classifier:ICS:StratChart:200908:Visean</ogc:Literal>
</ogc:PropertyIsEqualTo>
</ogc:Filter>
<sld:PolygonSymbolizer>
<sld:Fill>
<sld:CssParameter name="fill">#00FF00</sld:CssParameter>
</sld:Fill>
<sld:Stroke>
<sld:CssParameter name="stroke">#00FF00</sld:CssParameter>
<sld:CssParameter name="stroke-width">1</sld:CssParameter>
</sld:Stroke>
</sld:PolygonSymbolizer>
</sld:Rule>
</sld:FeatureTypeStyle>
</sld:UserStyle>
</sld:NamedLayer>
</sld:StyledLayerDescriptor>
```

Concerning ESRI and the SLD parameter, you must have in the request a `STYLES` parameters with no style defined, otherwise, the ESRI Map Server just return an error if the `STYLES` is not defined, or the default style if the `STYLES=default`. The SLD generated by the 1GEconnector are stored in a specific folder of the connector named slds. You can remove from time to time the files inside this folder to retrieve space.