GML application schema toolbox plugin

Final report

Version 1.1 - September 2016

Authors

Hugo Mercier, Oslandia, hugo dot mercier at oslandia dot com

Sylvain Grellet, BRGM, s dot grellet at brgm dot fr

Mickaël Beaufils, BRGM, m dot beaufils at brgm dot fr

Document version history

Version 1.1 – September 2016	
Version 1.0 – july 2016	Initial version

Sommaire

Context

BRGM is involved for a long time in the definition of interoperability standards especially linked to OGC and the European INSPIRE directive initiatives.

Existing tools being limited for an easy exploitation of these standards, this project aims at developing a prototype around QGIS and open source software pieces.

In particular, the aim is to develop tools to **manipulate Complex Features streams in a GIS desktop application**.

ISO19109 defines in its General Feature Model, the notion of FeatureType. Each domain can define its own application schema of FeatureTypes by reusing or extending the base types. Thus, used to describe and exchange domain related content, a FeatureType based information flow is often really rich in its datastructure. The new data structure often leads to the generation of a XSDs; basis of XML exchanges.

“Complex Feature” term is used as opposed to “Simple Feature” (OGC® 10-100r3), a subset of XML-Schema and GML to lower the “implementation bar” restricting spatial/non-spatial property types, cardinality of properties....

Complex Features streams are natively represented by an XML content which allows, thanks to its hierarchical structure, to express an instance coming from a rich object model.

Although being developed and tested on a fixed subset of application schemas, this project aims at being **generic and adaptable to any (valid) application schema**. We do not want to limit *a priori* the rich possibilities offered by the Complex Features object model. Possible problems of performances and limit in model complexity will have to be determined as soon as possible.

Most of the example below are based on :

- INSPIRE Environmental Monitoring Facility WFS flow on BRGM piezometers,
- GroundWaterML2.0 WFS flow on French aquifer reference dataset (BD LISA),
- SOS flows on the groundwater level measurements acquired by the piezometers monitoring those aquifers.

Manipulating GML complex features with QGIS revolves around two approaches, each having a dominating data representation mode:

1) **a native XML approach**, where data are stored and manipulated directly in their **XML hierarchical form**. Each “Complex Feature” is associated with a single row in a vector layer. The user is in charge of identifying among the XML tree, which elements are of interest for its use case.

2) **a “relational” approach**, where XML data are first stored in a database with relations between tables. The user then relies on native mechanisms found in QGIS to manipulate these date (joins, relations, form widgets).

The implementation of a fully functional prototype plugin allowed us to exhibit forces and limits of both approaches and to propose the best representation for data, that could even be an hybrid mode.

The plugin has been developed using the native QGIS API and external dependencies are shipped with it (PyXB), so that the plugin is usable on common operating systems and installation environments. The plugin is compatible with QGIS version 2.14 which is a Long Term Release and will be supported for at least one year after its release.

This plugin works either on a local file or a remote URL. Its purpose was not to redevelop pre-existing plugin dialoguing with OGC services (ex : WFS 2.0 client)

Plugin main window

The plugin is launched using the Plugin menu, or using the icon GML A combobox allows then to choose between the two importing modes.

"Import as one layer": when selected, the plugin will use the "native XML approach" described below.

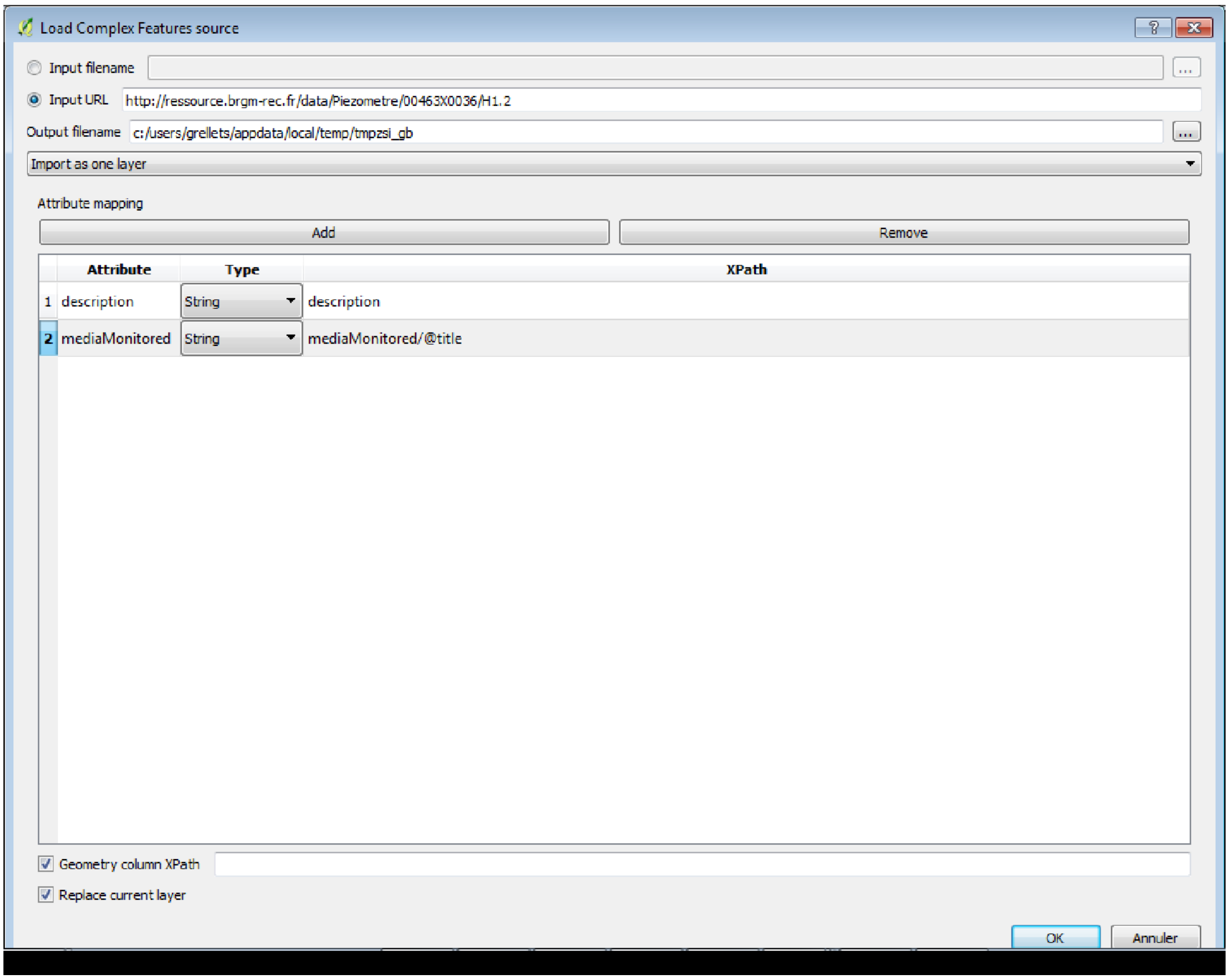
"Import using a relational model": when selected, the plugin will use the relational model approach

Native XML approach

In this approach, data are seen as **natively tree-like** (because represented as XML) and coming from an object model.

A QGIS plugin allows then to extract each "Complex Feature" from a GML document and to store it as a row in a vector layer. It also allows to access the **XML sub-tree** corresponding to each feature. A special column in the vector layer allows to store the XML sub-tree as a string.

The user can explicit **new columns** on the vector layer seen as "short-cuts" to data contained in the XML subtree and allowing to "flatten" the XML model. These short-cuts are expressed in the **Xpath language**.



Input URL: http://ressource.brgm-rec.fr/data/Piezometre/00463X0036/H1.2

Output filename: c:/users/grellets/appdata/local/temp/tmpzsi_gb

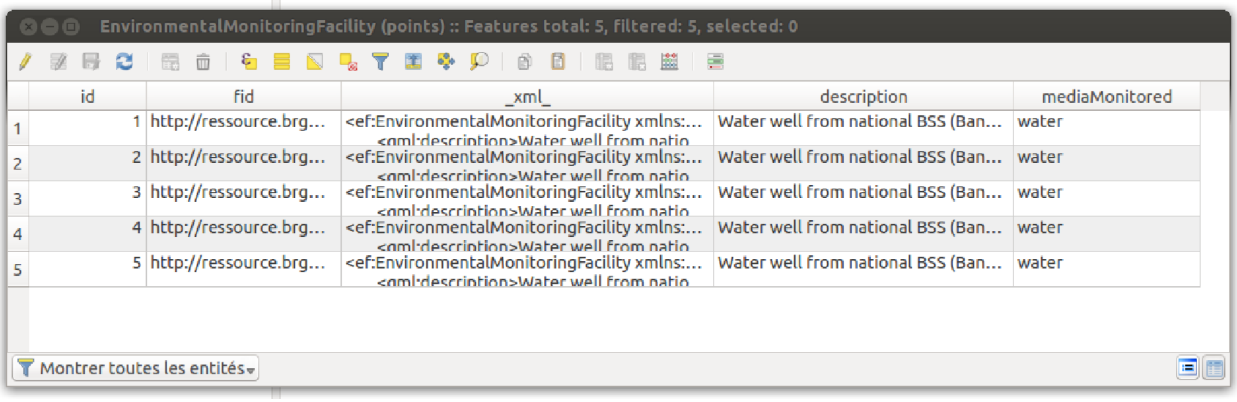
Import as one layer

	Attribute	Type	XPath
1	description	String	description
2	mediaMonitored	String	mediaMonitored/@title

Illustration 1: Main dialog of the plugin for adding/replacing a layer in XML mode

Upon loading, the plugin will create a new vector layer in a Spatialite file where each feature is stored with its geometry, a special column called "_xml_" to store the raw XML string and other columns if the user chose to add Xpath short-cuts.

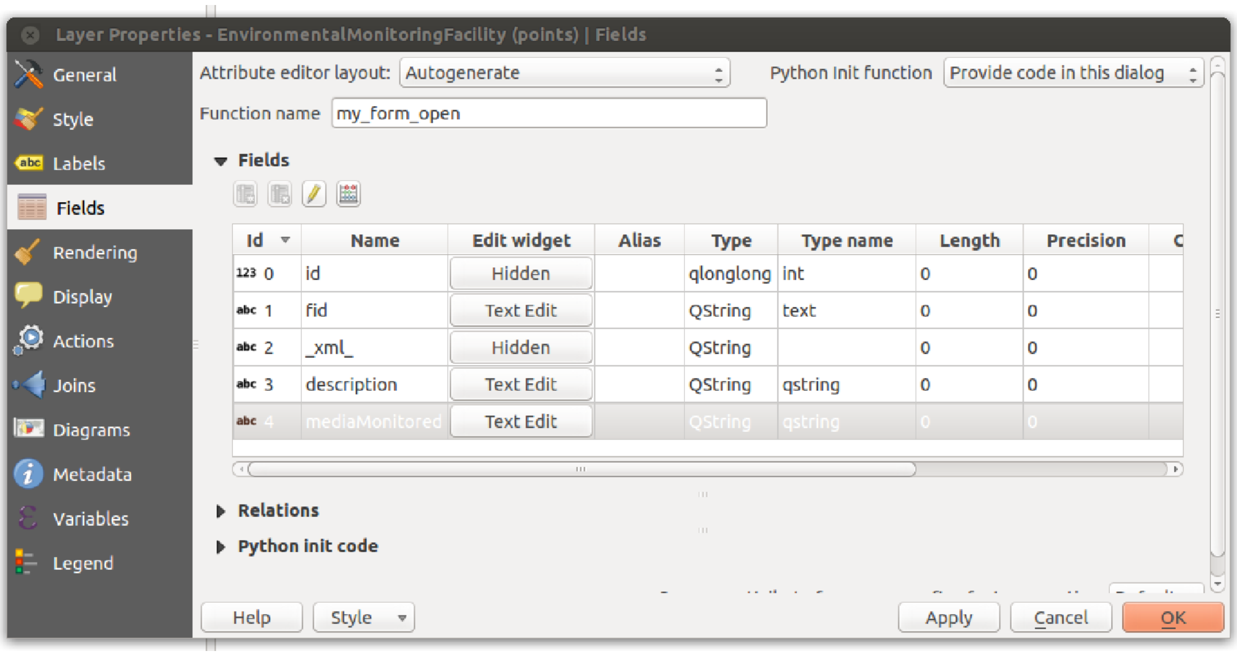
Here is an illustration of the raw attribute table of such a vector layer:



EnvironmentalMonitoringFacility (points) :: Features total: 5, filtered: 5, selected: 0

	id	fid	_xml_	description	mediaMonitored
1	1	http://ressource.brg...	<ef:EnvironmentalMonitoringFacility xmlns:... <gml:description>Water well from natio	Water well from national BSS (Ban...	water
2	2	http://ressource.brg...	<ef:EnvironmentalMonitoringFacility xmlns:... <gml:description>Water well from natio	Water well from national BSS (Ban...	water
3	3	http://ressource.brg...	<ef:EnvironmentalMonitoringFacility xmlns:... <gml:description>Water well from natio	Water well from national BSS (Ban...	water
4	4	http://ressource.brg...	<ef:EnvironmentalMonitoringFacility xmlns:... <gml:description>Water well from natio	Water well from national BSS (Ban...	water
5	5	http://ressource.brg...	<ef:EnvironmentalMonitoringFacility xmlns:... <gml:description>Water well from natio	Water well from national BSS (Ban...	water

The plugin automatically declares form widget types for each of these fields on the vector layer in order to clean the representation for the identification tool and the attribute table when opened in “form” mode.



Attribute editor layout: Autogenerate

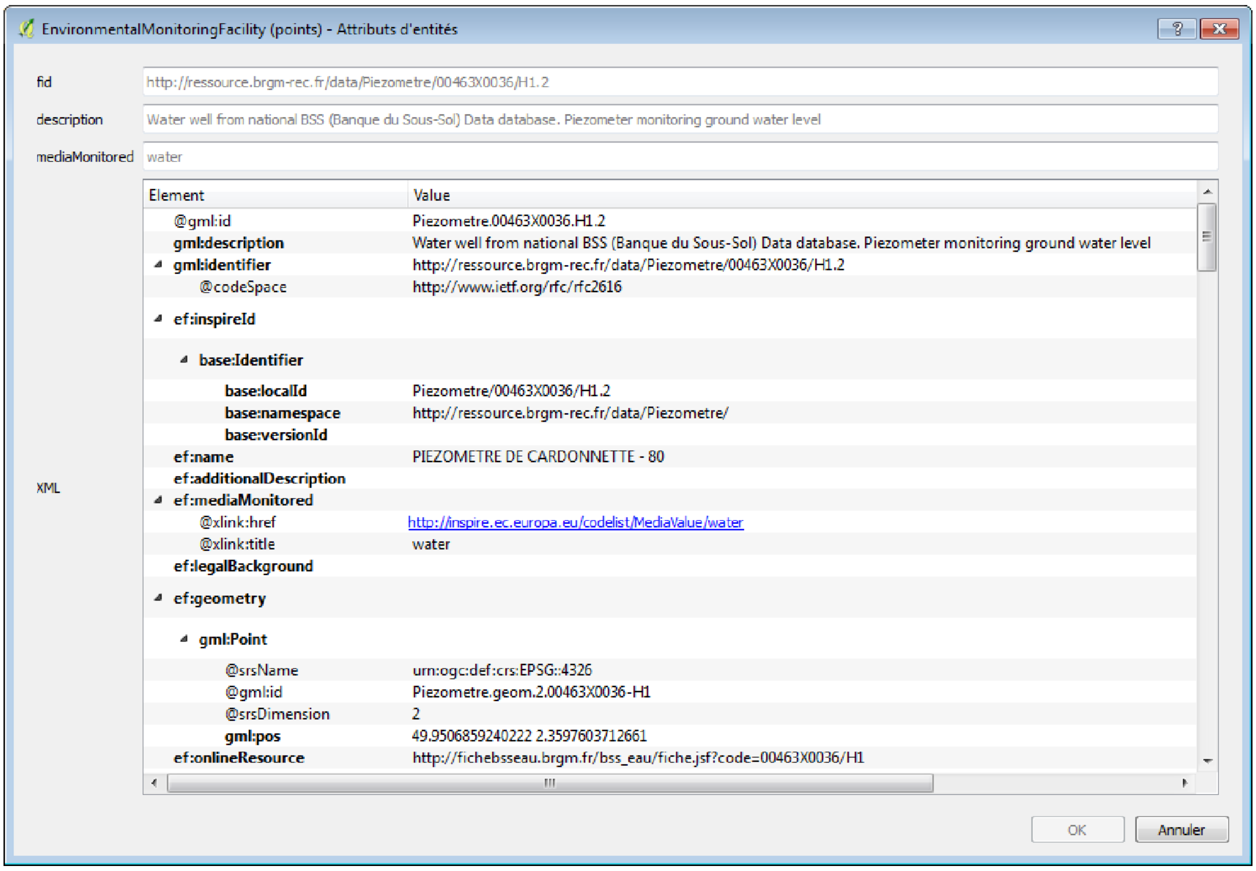
Python Init function: Provide code in this dialog

Function name: my_form_open

Id	Name	Edit widget	Alias	Type	Type name	Length	Precision
0	id	Hidden		qlonglong	int	0	0
1	fid	Text Edit		QString	text	0	0
2	_xml_	Hidden		QString		0	0
3	description	Text Edit		QString	qstring	0	0
4	mediaMonitored	Text Edit		QString	qstring	0	0

Illustration 2: Fields of an XML layer where the _xml_ field is declared as hidden and a custom Python init code allows to overload the default form

Given these declarations and a custom Python code that allows to overload the standard feature form generation of QGIS, the user can use standard tools of QGIS for identification and have access to the XML tree of the feature.



EnvironmentalMonitoringFacility (points) - Attributs d'entités

fid	http://ressource.brgm-rec.fr/data/Piezometre/00463X0036/H1.2
description	Water well from national BSS (Banque du Sous-Sol) Data database. Piezometer monitoring ground water level
mediaMonitored	water

XML

Element	Value
@gml:id	Piezometre.00463X0036.H1.2
gml:description	Water well from national BSS (Banque du Sous-Sol) Data database. Piezometer monitoring ground water level
gml:identifier	http://ressource.brgm-rec.fr/data/Piezometre/00463X0036/H1.2
@codeSpace	http://www.ietf.org/rfc/rfc2616
ef:inspireId	
base:Identifier	
base:localId	Piezometre/00463X0036/H1.2
base:namespace	http://ressource.brgm-rec.fr/data/Piezometre/
base:versionId	
ef:name	PIEZOMETRE DE CARDONNETTE - 80
ef:additionalDescription	
ef:mediaMonitored	
@xlink:href	http://inspire.ec.europa.eu/codelist/MediaValue/water
@xlink:title	water
ef:legalBackground	
ef:geometry	
gml:Point	
@srsName	urn:ogc:def:crs:EPSG::4326
@gml:id	Piezometre.geom.2.00463X0036-H1
@srsDimension	2
gml:pos	49.9506859240222 2.3597603712661
ef:onlineResource	http://fichebsseau.brgm.fr/bss_eau/fiche.jsf?code=00463X0036/H1

Illustration 3: Example of the form displayed when the standard identification tool of QGIS is used on an XML vector layer

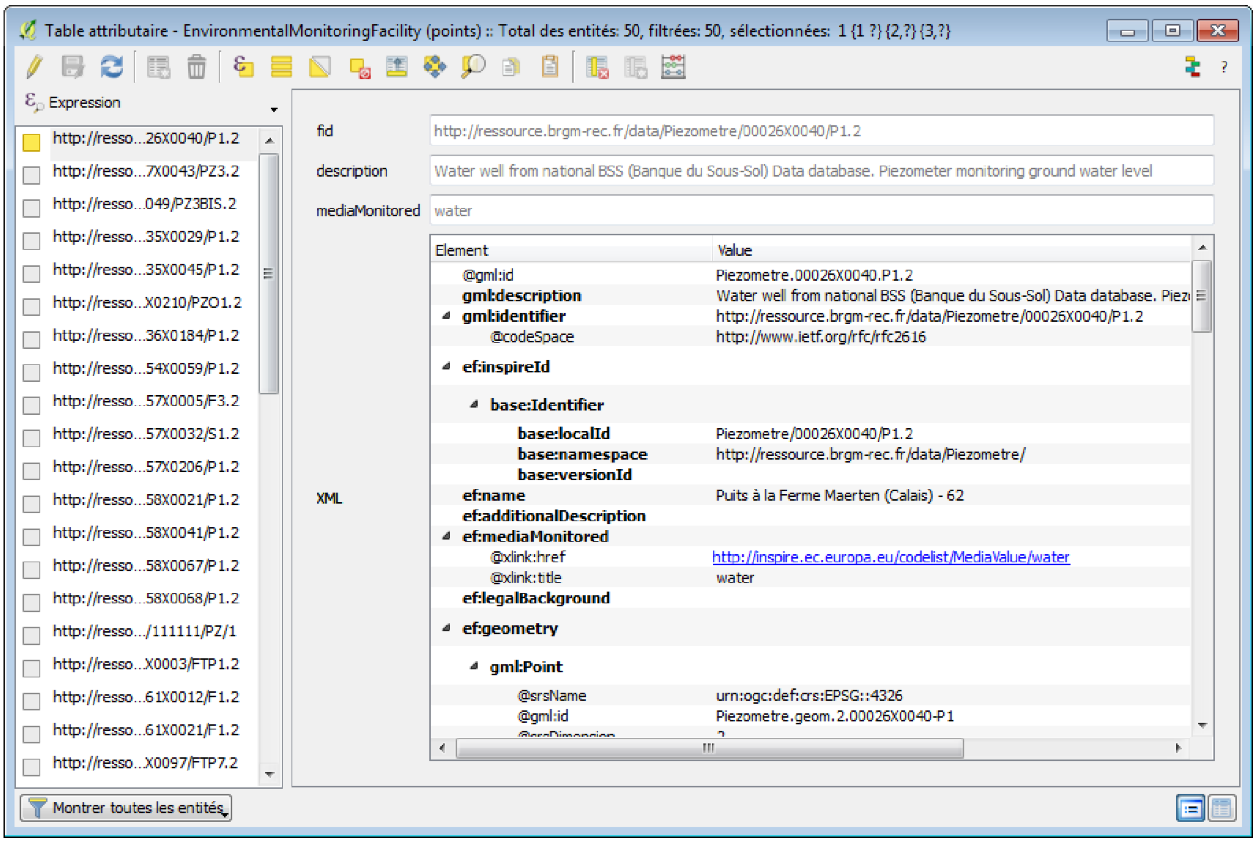


Table attributaire - EnvironmentalMonitoringFacility (points) :: Total des entités: 50, filtrées: 50, sélectionnées: 1 {1 ?} {2,?} {3,?}

- http://resso...26X0040/P1.2
- http://resso...7X0043/PZ3.2
- http://resso...049/PZ3BIS.2
- http://resso...35X0029/P1.2
- http://resso...35X0045/P1.2
- http://resso...X0210/PZO1.2
- http://resso...36X0184/P1.2
- http://resso...54X0059/P1.2
- http://resso...57X0005/F3.2
- http://resso...57X0032/S1.2
- http://resso...57X0206/P1.2
- http://resso...58X0021/P1.2
- http://resso...58X0041/P1.2
- http://resso...58X0067/P1.2
- http://resso...58X0068/P1.2
- http://resso.../111111/PZ/1
- http://resso...X0003/FTP1.2
- http://resso...61X0012/F1.2
- http://resso...61X0021/F1.2
- http://resso...X0097/FTP7.2

fid	http://ressource.brgm-rec.fr/data/Piezometre/00026X0040/P1.2
description	Water well from national BSS (Banque du Sous-Sol) Data database. Piezometer monitoring ground water level
mediaMonitored	water
XML	

Element	Value
@gml:id	Piezometre.00026X0040.P1.2
gml:description	Water well from national BSS (Banque du Sous-Sol) Data database. Piez
gml:identifier	http://ressource.brgm-rec.fr/data/Piezometre/00026X0040/P1.2
@codeSpace	http://www.ietf.org/rfc/rfc2616
ef:inspireId	
base:Identifier	
base:localId	Piezometre/00026X0040/P1.2
base:namespace	http://ressource.brgm-rec.fr/data/Piezometre/
base:versionId	
ef:name	Puits à la Ferme Maerten (Calais) - 62
ef:additionalDescription	
ef:mediaMonitored	
@xlink:href	http://inspire.ec.europa.eu/codelist/MediaValue/water
@xlink:title	water
ef:legalBackground	
ef:geometry	
gml:Point	
@srsName	urn:ogc:def:crs:EPSG::4326
@gml:id	Piezometre.geom.2.00026X0040-P1

Illustration 4: Example of the attribute table of an XML vector layer in "form" mode. The XML attribute is here interpreted as an interactive tree of elements

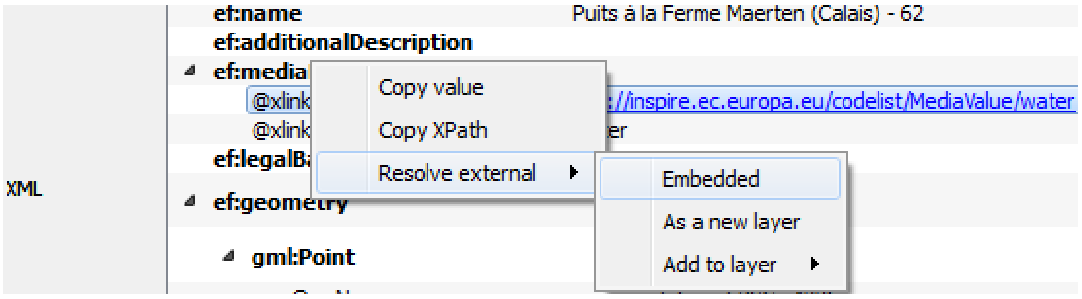
The plugin also allows to resolve external resources. When right-clicking on a xlink:href attribute, the user may choose to download it and embed the result in the current XML tree or to populate another layer.

When dereferencing the corresponding href, the http content negociation uses:

- the current user locale as the language (HTTP header “Accept-Language”)
- “application/xml” for the HTTP “Accept” field

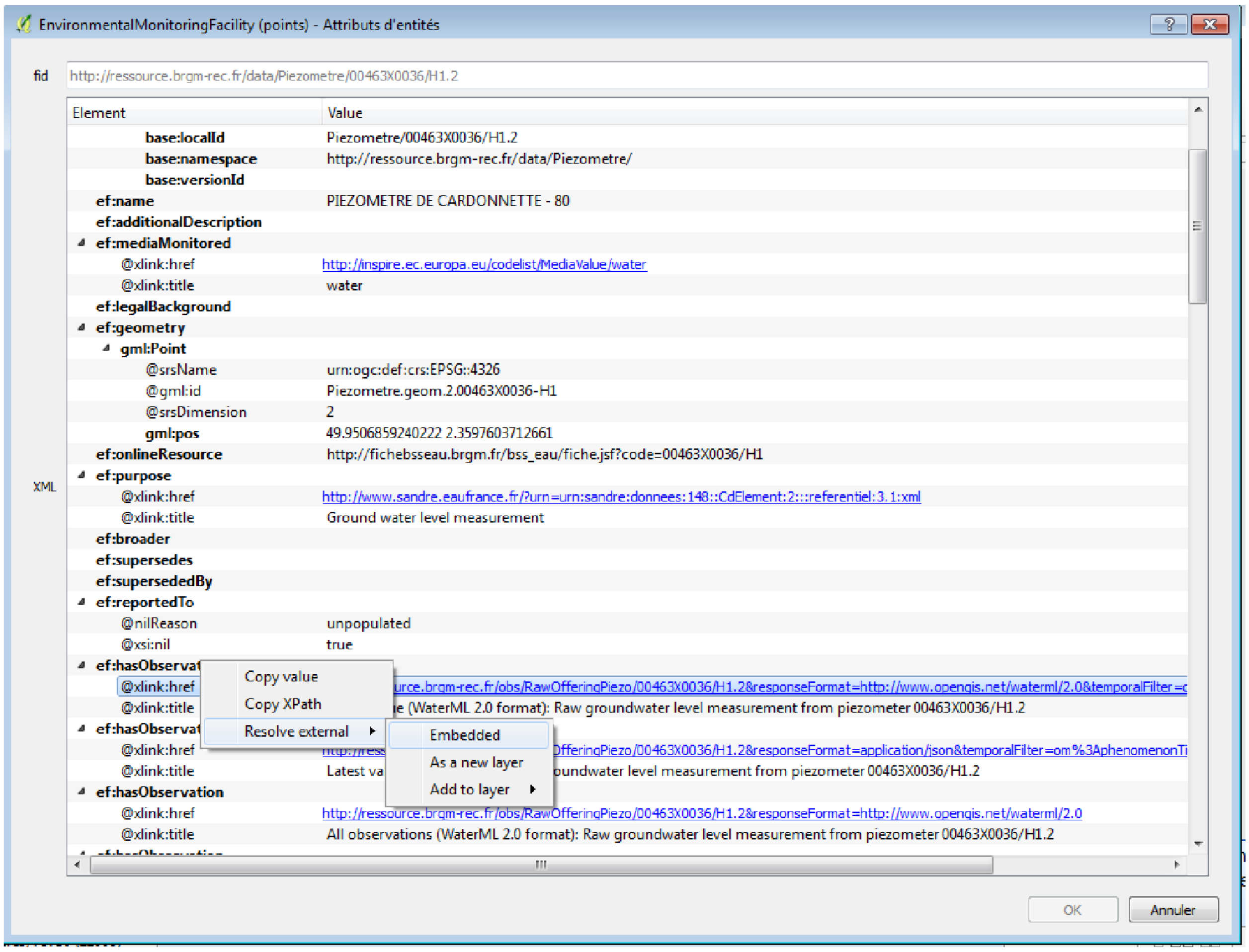
This approach is useful to retrieve information coming from :

- codeList registry : see example below on the Inspire registry



- or to retrieve a representation of a feature linked by reference:

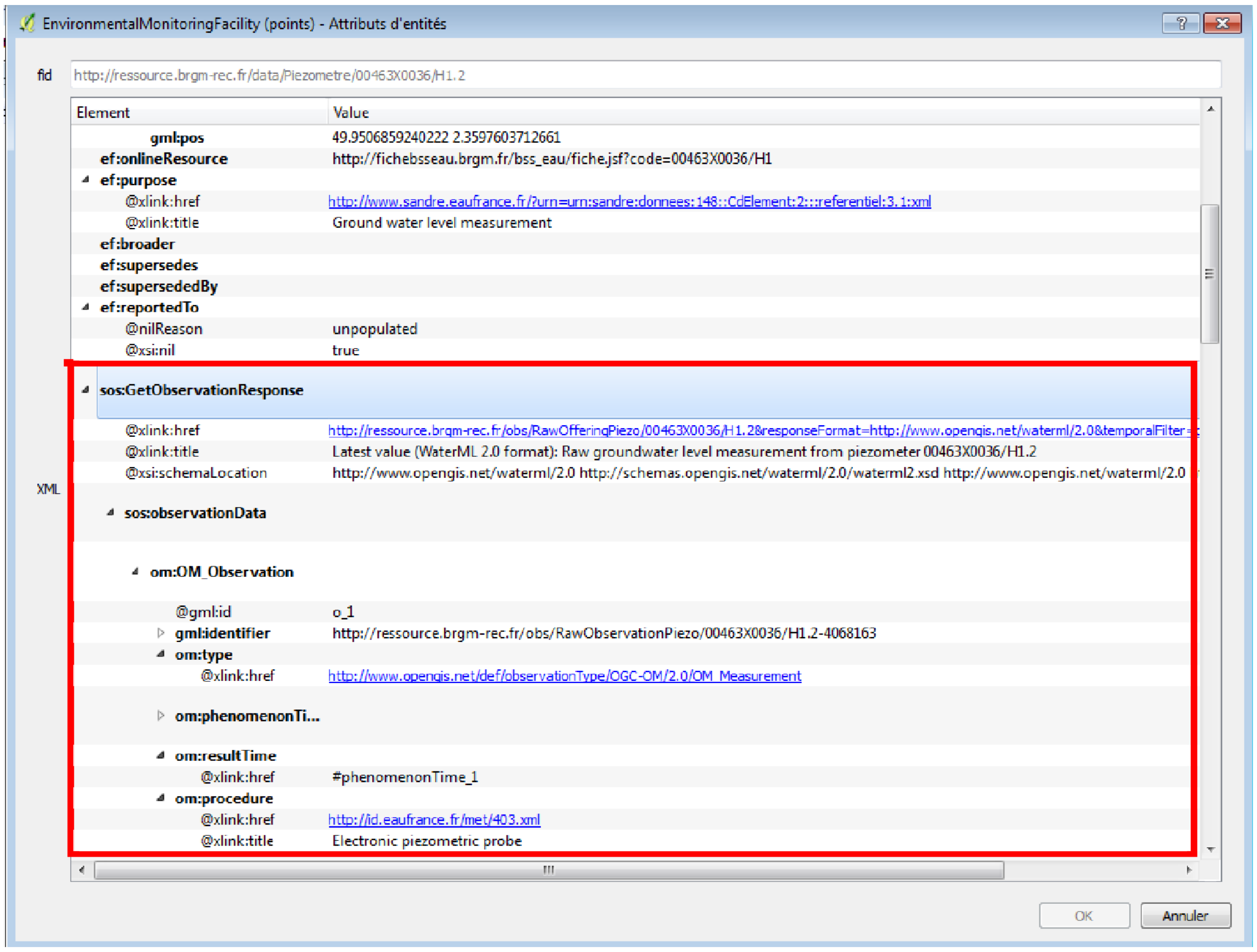
see the example below between a monitoring facility and an observation linked to it (depending on the application schema this could also be to another facility, network, aquifer,,..)



fid http://ressource.brgm-rec.fr/data/Piezometre/00463X0036/H1.2

Element	Value
base:localId	Piezometre/00463X0036/H1.2
base:namespace	http://ressource.brgm-rec.fr/data/Piezometre/
base:versionId	
ef:name	PIEZOMETRE DE CARDONNETTE - 80
ef:additionalDescription	
ef:mediaMonitored	
@xlink:href	http://inspire.ec.europa.eu/codelist/MediaValue/water
@xlink:title	water
ef:legalBackground	
ef:geometry	
gml:Point	
@srsName	urn:ogc:def:crs:EPSG::4326
@gml:id	Piezometre.geom.2.00463X0036-H1
@srsDimension	2
gml:pos	49.9506859240222 2.3597603712661
ef:onlineResource	http://fichebsseau.brgm.fr/bss_eau/fiche.jsf?code=00463X0036/H1
ef:purpose	
@xlink:href	http://www.sandre.eaufrance.fr/?urn=urn:sandre:donnees:148::CdElement:2:::referentiel:3.1:xml
@xlink:title	Ground water level measurement
ef:broader	
ef:supersedes	
ef:supersededBy	
ef:reportedTo	
@nilReason	unpopulated
@xsi:nil	true
ef:hasObservat	
@xlink:href	urce.brgm-rec.fr/obs/RawOfferingPiezo/00463X0036/H1.2&responseFormat=http://www.opengis.net/waterml/2.0&temporalFilter=c
@xlink:title	e (WaterML 2.0 format): Raw groundwater level measurement from piezometer 00463X0036/H1.2
ef:hasObservat	
@xlink:href	http://ress OfferingPiezo/00463X0036/H1.2&responseFormat=application/json&temporalFilter=om%3AphenomenonTi
@xlink:title	Latest va oundwater level measurement from piezometer 00463X0036/H1.2
ef:hasObservation	
@xlink:href	http://ressource.brgm-rec.fr/obs/RawOfferingPiezo/00463X0036/H1.2&responseFormat=http://www.opengis.net/waterml/2.0
@xlink:title	All observations (WaterML 2.0 format): Raw groundwater level measurement from piezometer 00463X0036/H1.2

Illustration 5: External resource resolving in "embedded" mode


EnvironmentalMonitoringFacility (points) - Attributs d'entités

fid: http://ressource.brgm-rec.fr/data/Piezometre/00463X0036/H1.2

XML

Element	Value
gml:pos	49.9506859240222 2.3597603712661
ef:onlineResource	http://fichebsseau.brgm.fr/bss_eau/fiche.jsf?code=00463X0036/H1
ef:purpose	
@xlink:href	http://www.sandre.eaufrance.fr/?urn=urn:sandre:donnees:148::CdElement:2:::referentiel:3.1:xml
@xlink:title	Ground water level measurement
ef:broader	
ef:supersedes	
ef:supersededBy	
ef:reportedTo	
@nilReason	unpopulated
@xsi:nil	true
sos:GetObservationResponse	
@xlink:href	http://ressource.brgm-rec.fr/obs/RawOfferingPiezo/00463X0036/H1.2&responseFormat=http://www.opengis.net/waterml/2.0&temporalFilter=
@xlink:title	Latest value (WaterML 2.0 format): Raw groundwater level measurement from piezometer 00463X0036/H1.2
@xsi:schemaLocation	http://www.opengis.net/waterml/2.0 http://schemas.opengis.net/waterml/2.0/waterml2.xsd http://www.opengis.net/waterml/2.0
sos:observationData	
om:OM_Observation	
@gml:id	o_1
gml:identifier	http://ressource.brgm-rec.fr/obs/RawObservationPiezo/00463X0036/H1.2-4068163
om:type	
@xlink:href	http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_Measurement
om:phenomenonTi...	
om:resultTime	
@xlink:href	#phenomenonTime_1
om:procedure	
@xlink:href	http://id.eaufrance.fr/met/403.xml
@xlink:title	Electronic piezometric probe

Illustration 6: Result of the external resource embedding

Relational approach

In a relational approach, the main problem is to **convert from an XML flow of Complex Features instances to a relational database representation**. The approach we followed consists in:

- analysing the XSD schema(s) declared by the XML instance in order to retrieve the underlying object model and determine the type of each element and attribute as well as links between elements ;
- converting links between elements into relations between database tables;
- inserting values in the database corresponding to the values found in the XML instance;
- generating a QGIS project configured to easily manipulate the relational model.

XSD schema analysis

The analysis task of an XSD schema requires a software brick able to offer an API for an easy manipulation of this schema in memory and to associate a **strong type** to each element and attribute of an XML document.

Although numerous libraries allow to validate an XML document against a given XSD schema, namely to return “valid” or “invalid” when giving an XML and a XSD, there are very few software libraries that allow to **access the XSD meta model**.

The possible ways identified so far are:

- the Xerces C++ library which supports “Schema API” and implements in particular “post-schema-validation infoset” which allows to access schema information (and then types) of an element or attribute
- using a conversion tool that is designed to convert an XSD model to an object model. This includes: Code Synthesis in C++ and PyXB in Python

For this prototype, we decided to **rely on PyXB** for the schema analysis.

Indeed, this library shows a high level of conformity with complex XSD. It is shipped for instance with “bundles” of Python object models converted from many OpenGIS schemas.

Moreover, the Python programming language seemed to be a good way to explore features in a QGIS plugin. Xerces could still be used from Python, but we would have to find or write good Python bindings and deal with packaging issues.

We thus had used PyXB in a way that is not directly design for: before generating Python files from an XSD schema, we had to find a way to access the internal object model and generate a database structure out of it.

XSD downloading and caching

The very first step in this approach is to collect every XSD documents to properly define the whole application schema.

The starting point is the XML file passed as input. It contains namespace resolution directives that can be used to download XSD documents describing the schema. These documents may in turn have references to other XSD documents (through <include> or <import>).

Because the number of XSD files to download for a given application schema may be important, they are cached in a local directory. Before trying to download XSD documents, the cache is requested.

By default the cache directory is set to a temporary directory.

XSD → SQL conversion

Converting an XSD model, and more generally an object model into a relational model requires choices to be done, since the relational model is, in essence, poorer than a complete object model.

The basic rules used for the conversion are the following:

1. each attribute is represented by a column
2. an element with maxOccurs = “unbounded” is considered as a 1-N link. A new table will be created with a foreign key constraint
3. an element with maxOccurs = 1 is represented as a column, but only if its type is simple. If its type is complex, a new table is created with a foreign key constraint
4. elements that inherit from AbstractGeometryType are represented as a geometry column
5. if minOccurs = 0 or nillable = true, the link or the column is declared as optional (may be NULL)
6. substitution groups are processed as follows: for two types A and B of a substitution group SG, three tables SG, A and B are created. In SG, two foreign keys “a_id” and “b_id” pointing to A and B are added with a CHECK constraint that express the “exclusive or” : an SG can be either an A or B, but not both. This mechanism is generalized when there are more than two types in the substitution group.
7. If an element has an “id” attribute, it means it may be referenced by another element, through an href link. A new table is then created in that case.

Naively applying this set of rules to an XSD schema gives most of the time a huge number of tables that is very hard for the end user to figure out and manipulate.

This is due to the fact that OpenGIS schemas are very rich and contains lots of optional possibilities.

We spent some time on the conversion process in order to simplify the resulting relational model.

This leads us to the following heuristics rules.

Table merging

The rule number 3 is modified so that if the compound type of an element is composed itself of unitary elements (maxOccurs=1), then a new table is not necessary and members of the element can be represented as columns by concatenation.

Example:

From the following XML:

```
<a>

  <b a1="1" color="green"/>

</a>
```

The naive rules would give two tables:

Table A
Id
b_id : foreign key to B

Table B
id
a1
color

With the modified rule 3, this would result in only one table:

Table A
id
b_a1
b_color

We then introduced an option to set the maximum number of tables than can be "merged" this way under the attribute 'Maximum table merging depth' (see Illustration 7 below).

Conversion driven by the instance

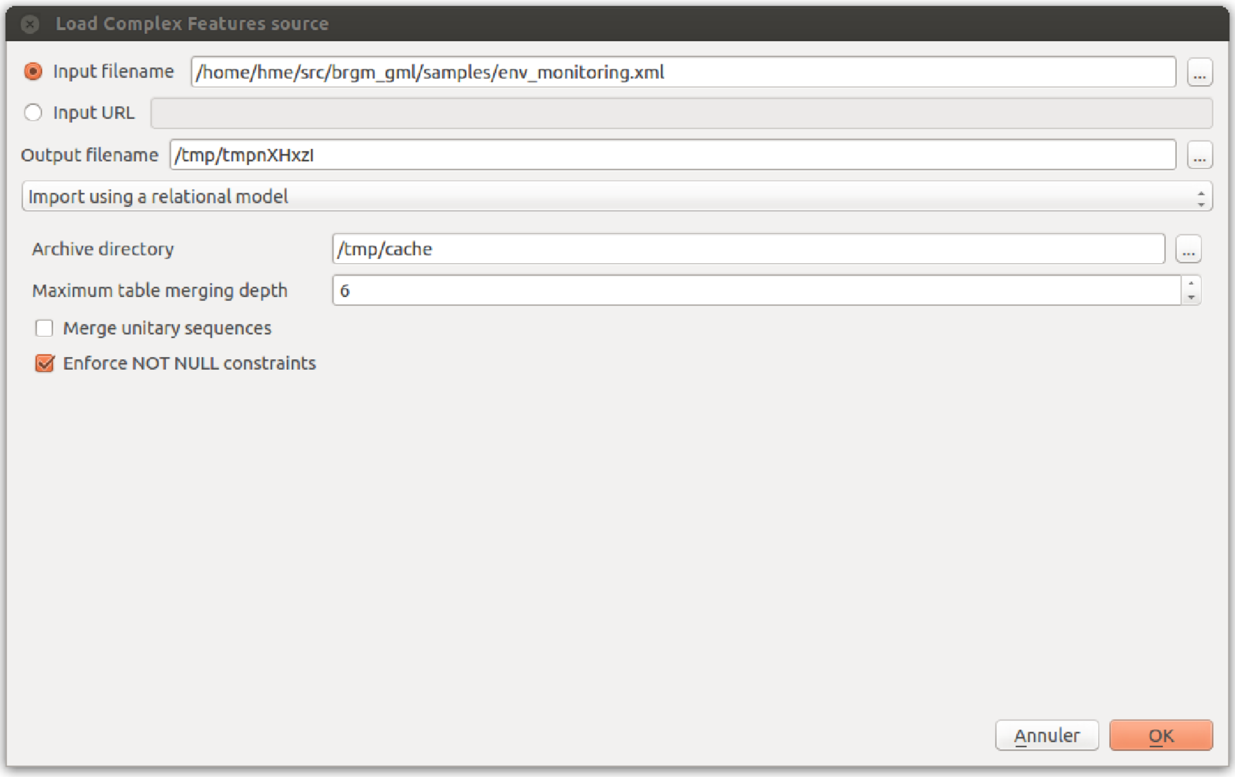
We also decided to drive the conversion based on the XML instance and not the XSD schema. It prevents the number of empty tables to grow if nothing fills them in the instance. The conversion of each complex feature may modify the current relational model by adding new columns or new tables. Tables that have been merged in a first step may also be "unmerged" if the merge is no longer possible.

Merge of unitary sequences

Some elements may be declared as sequences, resulting in the creation of a table with a foreign key. But there are lots of cases where sequences are most of the time populated with only one element. An option then allows to consider by default sequences as unitary elements (and thus merge-able).

When a feature actually has more than one member in the sequence, the model is modified and a new table is created.

These options are presented to the user on the main creation dialog:


Load Complex Features source

[● Input filename] /home/hme/src/brgm_gml/samples/env_monitoring.xml

○ Input URL

Output filename /tmp/tmpnXHxzI

Import using a relational model

Archive directory /tmp/cache

Maximum table merging depth 6

☐ Merge unitary sequences

☑ Enforce NOT NULL constraints

Illustration 7: Complex Feature source loading interface

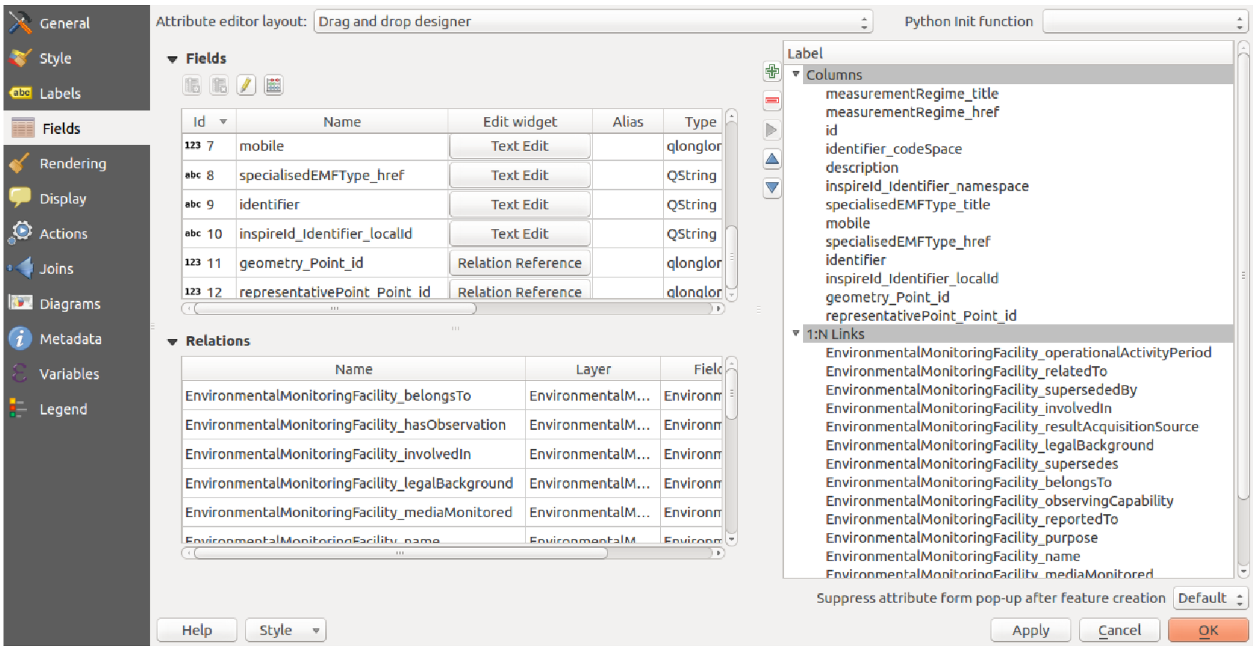
QGIS project generation

Converting an XML instance into a relational SQL model generates numerous tables and relations between them.

In order to expose the entire relational model inside QGIS, all the tables of the database must be loaded (including tables without geometries) and links between them must be declared.

QGIS has a concept of **“relations”** that allows to declare 1:N relations between vector layers. These relations can then be used in the form view of layers to navigate the model.

The plugin developed automatically generates a QGIS project with all the layers loaded and all the known relations declared.



Attribute editor layout: Drag and drop designer

Python Init function

Fields

Id	Name	Edit widget	Alias	Type
7	mobile	Text Edit		qlonglor
8	specialisedEMFType_href	Text Edit		QString
9	identifier	Text Edit		QString
10	inspireId_Identifier_localId	Text Edit		QString
11	geometry_Point_id	Relation Reference		qlonglor
12	representativePoint_Point_id	Relation Reference		qlonglor

Relations

Name	Layer	Fielc
EnvironmentalMonitoringFacility_belongsTo	EnvironmentalM...	Environm
EnvironmentalMonitoringFacility_hasObservation	EnvironmentalM...	Environm
EnvironmentalMonitoringFacility_involvedIn	EnvironmentalM...	Environm
EnvironmentalMonitoringFacility_legalBackground	EnvironmentalM...	Environm
EnvironmentalMonitoringFacility_mediaMonitored	EnvironmentalM...	Environm
EnvironmentalMonitoringFacility_name	EnvironmentalM...	Environm

Label
- Columns
  - measurementRegime_title
  - measurementRegime_href
  - id
  - identifier_codeSpace
  - description
  - inspireId_Identifier_namespace
  - specialisedEMFType_title
  - mobile
  - specialisedEMFType_href
  - identifier
  - inspireId_Identifier_localId
  - geometry_Point_id
  - representativePoint_Point_id
- 1:N Links
  - EnvironmentalMonitoringFacility_operationalActivityPeriod
  - EnvironmentalMonitoringFacility_relatedTo
  - EnvironmentalMonitoringFacility_supersededBy
  - EnvironmentalMonitoringFacility_involvedIn
  - EnvironmentalMonitoringFacility_resultAcquisitionSource
  - EnvironmentalMonitoringFacility_legalBackground
  - EnvironmentalMonitoringFacility_supersedes
  - EnvironmentalMonitoringFacility_belongsTo
  - EnvironmentalMonitoringFacility_observingCapability
  - EnvironmentalMonitoringFacility_reportedTo
  - EnvironmentalMonitoringFacility_purpose
  - EnvironmentalMonitoringFacility_name
  - EnvironmentalMonitoringFacility_mediaMonitored

Suppress attribute form pop-up after feature creation: Default

Illustration 8: Example of fields declared by the plugin on a table of the relational model. Foreign key columns are declared as relation references

Layer forms are represented with three tabs:

- a “column” tab that contains all the regular columns of the layer
- an optional “1:N links” tab that contains (recursively) linked layers
- an optional “Backlinks” tab that contains layers that point to the current layer

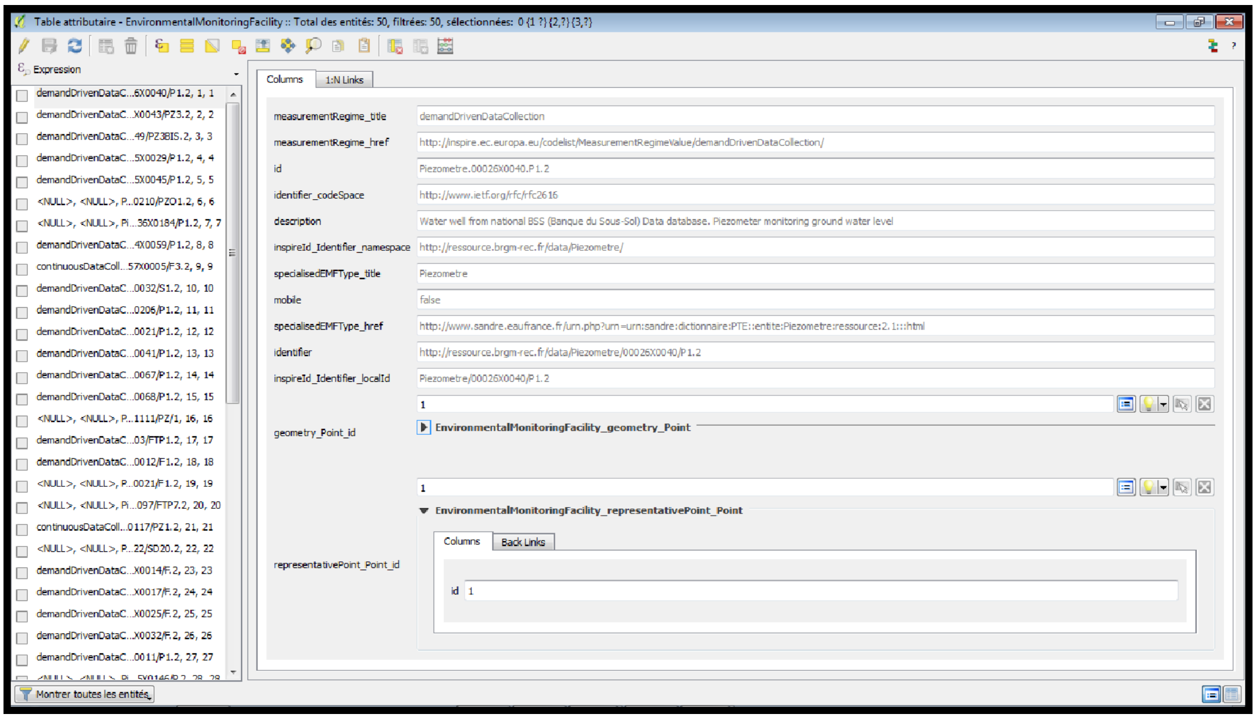
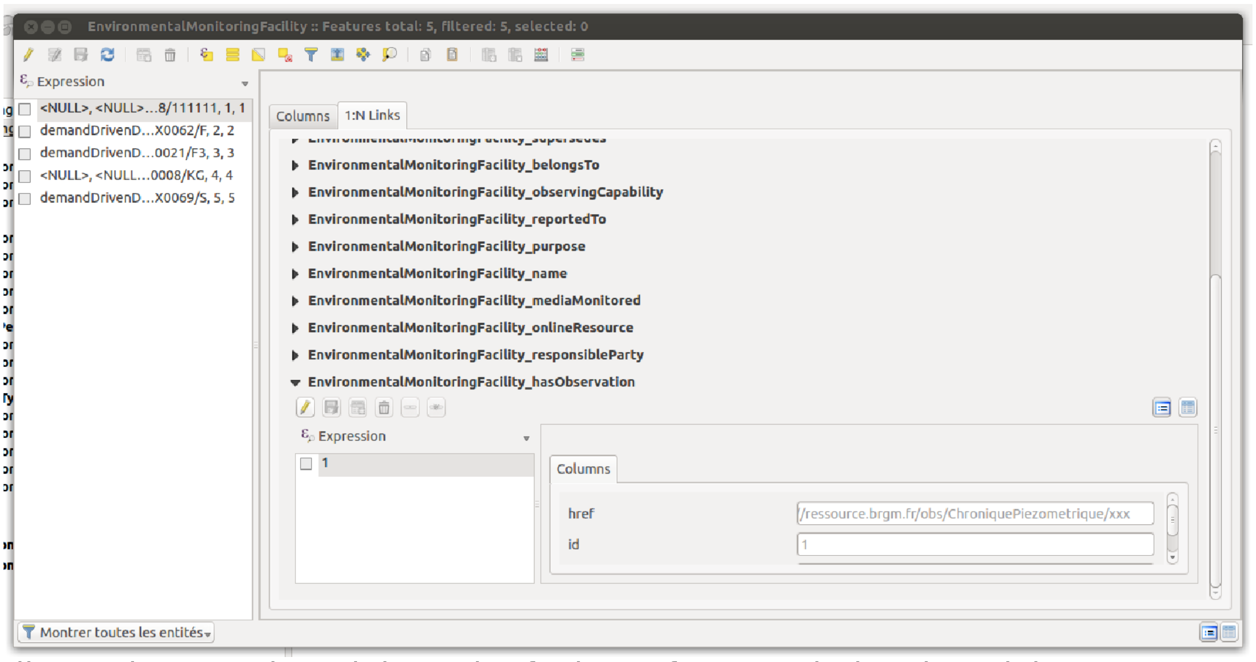


Table attributaire - EnvironmentalMonitoringFacility :: Total des entités: 50, filtrées: 50, sélectionnées: 0 {1 ?} {2,?} {3,?}

demandDrivenDataC...6X0040/P1.2, 1, 1
demandDrivenDataC...X0043/PZ3.2, 2, 2
demandDrivenDataC...49/PZ3BIS.2, 3, 3
demandDrivenDataC...5X0029/P1.2, 4, 4
demandDrivenDataC...5X0045/P1.2, 5, 5
<NULL>, <NULL>, P...0210/PZO1.2, 6, 6
<NULL>, <NULL>, Pi...36X0184/P1.2, 7, 7
demandDrivenDataC...4X0059/P1.2, 8, 8
continuousDataColl...57X0005/F3.2, 9, 9
demandDrivenDataC...0032/S1.2, 10, 10
demandDrivenDataC...0206/P1.2, 11, 11
demandDrivenDataC...0021/P1.2, 12, 12
demandDrivenDataC...0041/P1.2, 13, 13
demandDrivenDataC...0067/P1.2, 14, 14
demandDrivenDataC...0068/P1.2, 15, 15
<NULL>, <NULL>, P...1111/PZ/1, 16, 16
demandDrivenDataC...03/FTP1.2, 17, 17
demandDrivenDataC...0012/F1.2, 18, 18
<NULL>, <NULL>, P...0021/F1.2, 19, 19
<NULL>, <NULL>, Pi...097/FTP7.2, 20, 20
continuousDataColl...0117/PZ1.2, 21, 21
<NULL>, <NULL>, P...22/SD20.2, 22, 22
demandDrivenDataC...X0014/F.2, 23, 23
demandDrivenDataC...X0017/F.2, 24, 24
demandDrivenDataC...X0025/F.2, 25, 25
demandDrivenDataC...X0032/F.2, 26, 26
demandDrivenDataC...0011/P1.2, 27, 27

measurementRegime_title	demandDrivenDataCollection
measurementRegime_href	http://inspire.ec.europa.eu/codelist/MeasurementRegimeValue/demandDrivenDataCollection/
id	Piezometre.00026X0040.P1.2
identifier_codeSpace	http://www.ietf.org/rfc/rfc2616
description	Water well from national BSS (Banque du Sous-Sol) Data database. Piezometer monitoring ground water level
inspireId_Identifier_namespace	http://ressource.brgm-rec.fr/data/Piezometre/
specialisedEMFType_title	Piezometre
mobile	false
specialisedEMFType_href	http://www.sandre.eaufrance.fr/urn.php?urn=urn:sandre:dictionnaire:PTE::entite:Piezometre:ressource:2.1:::html
identifier	http://ressource.brgm-rec.fr/data/Piezometre/00026X0040/P1.2
inspireId_Identifier_localId	Piezometre/00026X0040/P1.2
geometry_Point_id	1 — EnvironmentalMonitoringFacility_geometry_Point
representativePoint_Point_id	1 — EnvironmentalMonitoringFacility_representativePoint_Point — id 1

Illustration 9: The "columns" tab of a layer from the relational model



EnvironmentalMonitoringFacility :: Features total: 5, filtered: 5, selected: 0

Expression

- <NULL>, <NULL>...8/111111, 1, 1
- demandDrivenD...X0062/F, 2, 2
- demandDrivenD...0021/F3, 3, 3
- <NULL>, <NULL...0008/KG, 4, 4
- demandDrivenD...X0069/S, 5, 5

Columns | 1:N Links

- EnvironmentalMonitoringFacility_supersedes
- EnvironmentalMonitoringFacility_belongsTo
- EnvironmentalMonitoringFacility_observingCapability
- EnvironmentalMonitoringFacility_reportedTo
- EnvironmentalMonitoringFacility_purpose
- EnvironmentalMonitoringFacility_name
- EnvironmentalMonitoringFacility_mediaMonitored
- EnvironmentalMonitoringFacility_onlineResource
- EnvironmentalMonitoringFacility_responsibleParty
- EnvironmentalMonitoringFacility_hasObservation

Expression

- 1

Columns

href	/ressource.brgm.fr/obs/ChroniquePiezometrique/xxx
id	1

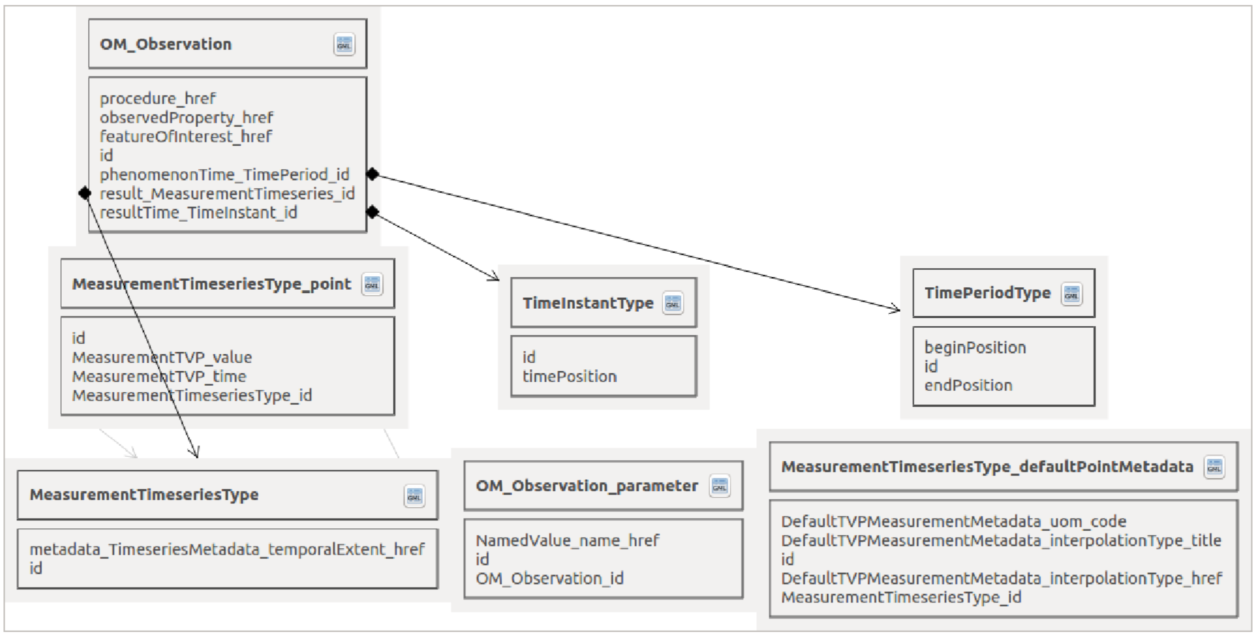
Montrer toutes les entités

Illustration 10: The "Links" tab of a layer from a relational model

Note that there is no specific code to implement this form view, only proper configurations of the QGIS project.

Relational schema visualisation

The plugin also exposes a dialog that allows to display the relational model created. It is graphically represented by the different tables with links between them represented as arrows.



OM_Observation
procedure_href
observedProperty_href
featureOfInterest_href
id
phenomenonTime_TimePeriod_id
result_MeasurementTimeseries_id
resultTime_TimeInstant_id
MeasurementTimeseriesType_point
id
MeasurementTVP_value
MeasurementTVP_time
MeasurementTimeseriesType_id
TimeInstantType
id
timePosition
TimePeriodType
beginPosition
id
endPosition
MeasurementTimeseriesType
metadata_TimeseriesMetadata_temporalExtent_href
id
OM_Observation_parameter
NamedValue_name_href
id
OM_Observation_id
MeasurementTimeseriesType_defaultPointMetadata
DefaultTVPMeasurementMetadata_uom_code
DefaultTVPMeasurementMetadata_interpolationType_title
id
DefaultTVPMeasurementMetadata_interpolationType_href
MeasurementTimeseriesType_id

On each table, a button allows to directly open the attribute table of the given layer.

Common elements

List of features that are common to both native XML mode and relational mode.

Feature cutting up

The plugin is designed to extract a collection of Complex Features from different kinds of streams (either a local file or a remote URL):

- If the stream results from a WFS request, each subtree of the <member> (WFS 2.0) elements or every child of the <featureMembers> (WFS 1.1) element is considered as a feature
- Else, the document is considered to be an isolated feature

Besides, some application schemas also return collections of features. This is the case of streams resulting from requests to a Sensor Observation Service (each <observationData> is a feature)

Handling of identifiers

Different kinds of identifiers may be found in a Complex Features stream. The XML instance will contain a gml:id attribute and may contain a gml:identifier element.

Those two types of identifier serve different purposes :

- mandatory gml:id provides a unique identifier within a given flow (or file)
- gml:identifier is not mandatory per se but is more and more used to provide an externally unique and stable identifier for a given feature (linked open data)

Note that QGIS layers should have an integer unique identifier in order to be selectable by id. So “native” identifiers (gml:identifier or gml:id) are kept as is and the primary key is set on an autoincremented integer.

Multiple geometries

A Complex Features stream may contain more than one geometry per feature. And they can be of different types. The QGIS layer data model is more limited and one layer can only be associated to a geometry type.

In relational mode, the plugin will create different layers for each geometry type.

In XML mode, the plugin will use the first geometry found. If the user wants to see the other geometry types, the import has to be run again with a “geometry xpath”.

Custom element viewers

Accessing the information contained in XML elements, either thanks to the native XML approach or thanks to the relational model approach is not always easy.

Some elements may be visually displayed in a way that is more pleasant for the end user.

We have tested the idea here of **custom element viewers for certain known types**.

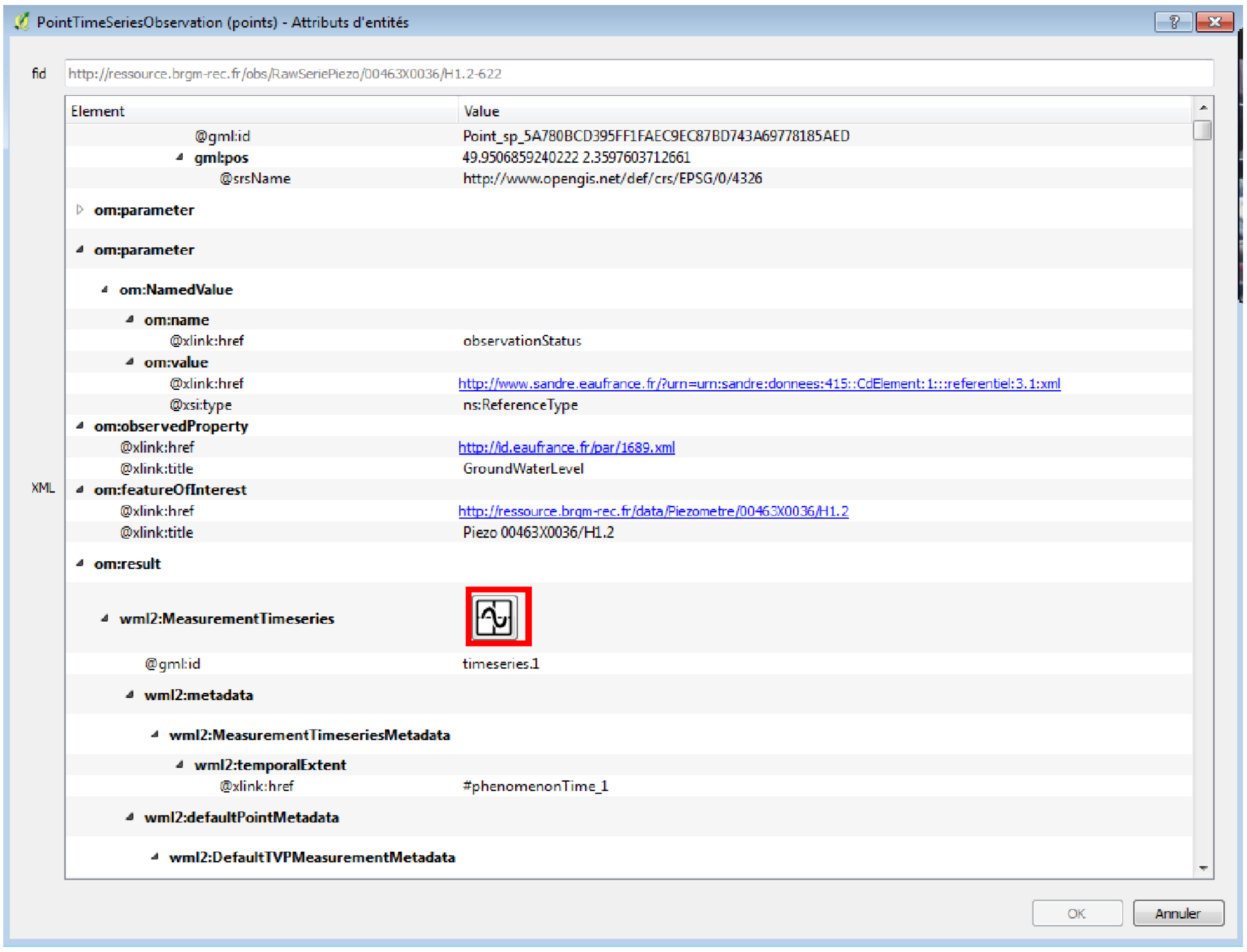
In particular, we have developed a custom viewer for the time series of measurements that can be found in the WaterML2 – part I : TimeSeries application schema.

Such time series are represented by XML sequences of values. Plotting them as a curve should be a better representation than a raw list of time instants and values.

The idea behind this custom viewer is to offer an API that could be used by third party developers to provide custom viewer widgets for some complex types.

This API is developed so that it works regardless of the value passed for the “Maximum table merging depth” (see Illustration 7 above)

The “time series” custom viewer is shipped with the current plugin. It is associated with “wml2:MeasurementTimeseries” elements. In both modes (native XML or relational), a button allows to display time series thanks to this custom viewer.



PointTimeSeriesObservation (points) - Attributs d'entités

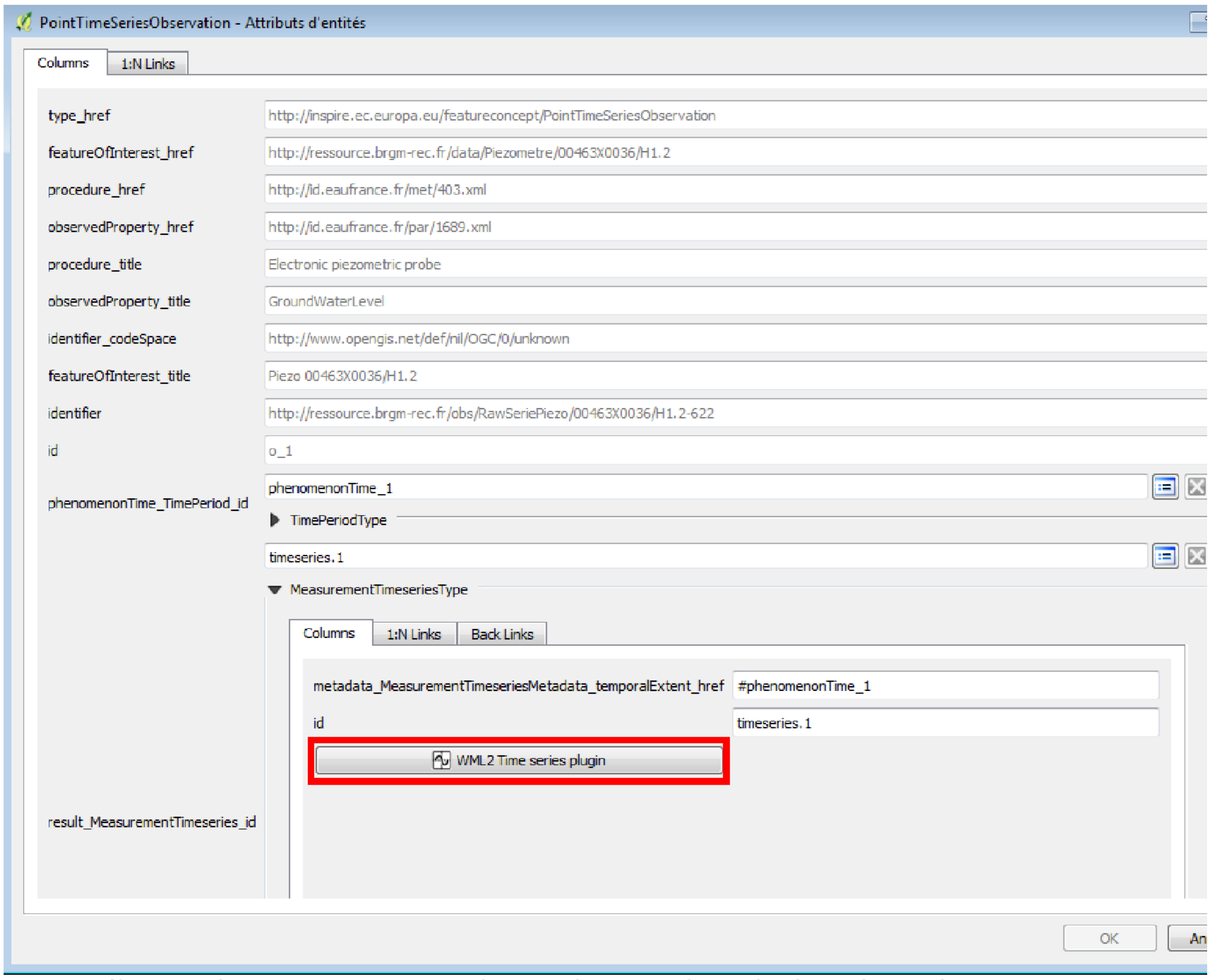
fid http://ressource.brgm-rec.fr/obs/RawSeriePiezo/00463X0036/H1.2-622

XML

Element	Value
@gml:id	Point_sp_5A780BCD395FF1FAEC9EC87BD743A69778185AED
gml:pos	49.9506859240222 2.3597603712661
@srsName	http://www.opengis.net/def/crs/EPSG/0/4326
om:parameter	
om:parameter	
om:NamedValue	
om:name	
@xlink:href	observationStatus
om:value	
@xlink:href	http://www.sandre.eaufrance.fr/?urn=urn:sandre:donnees:415::CdElement:1:::referentiel:3.1:xml
@xsi:type	ns:ReferenceType
om:observedProperty	
@xlink:href	http://id.eaufrance.fr/par/1689.xml
@xlink:title	GroundWaterLevel
om:featureOfInterest	
@xlink:href	http://ressource.brgm-rec.fr/data/Piezometre/00463X0036/H1.2
@xlink:title	Piezo 00463X0036/H1.2
om:result	
wml2:MeasurementTimeseries	
@gml:id	timeseries.1
wml2:metadata	
wml2:MeasurementTimeseriesMetadata	
wml2:temporalExtent	
@xlink:href	#phenomenonTime_1
wml2:defaultPointMetadata	
wml2:DefaultTVPMeasurementMetadata	

OK Annuler

Illustration 11: Custom viewer button in native XML mode


PointTimeSeriesObservation - Attributs d'entités

Columns | 1:N Links

type_href	http://inspire.ec.europa.eu/featureconcept/PointTimeSeriesObservation
featureOfInterest_href	http://ressource.brgm-rec.fr/data/Piezometre/00463X0036/H1.2
procedure_href	http://id.eaufrance.fr/met/403.xml
observedProperty_href	http://id.eaufrance.fr/par/1689.xml
procedure_title	Electronic piezometric probe
observedProperty_title	GroundWaterLevel
identifier_codeSpace	http://www.opengis.net/def/nil/OGC/0/unknown
featureOfInterest_title	Piezo 00463X0036/H1.2
identifier	http://ressource.brgm-rec.fr/obs/RawSeriePiezo/00463X0036/H1.2-622
id	o_1
phenomenonTime_TimePeriod_id	phenomenonTime_1
result_MeasurementTimeseries_id	timeseries.1

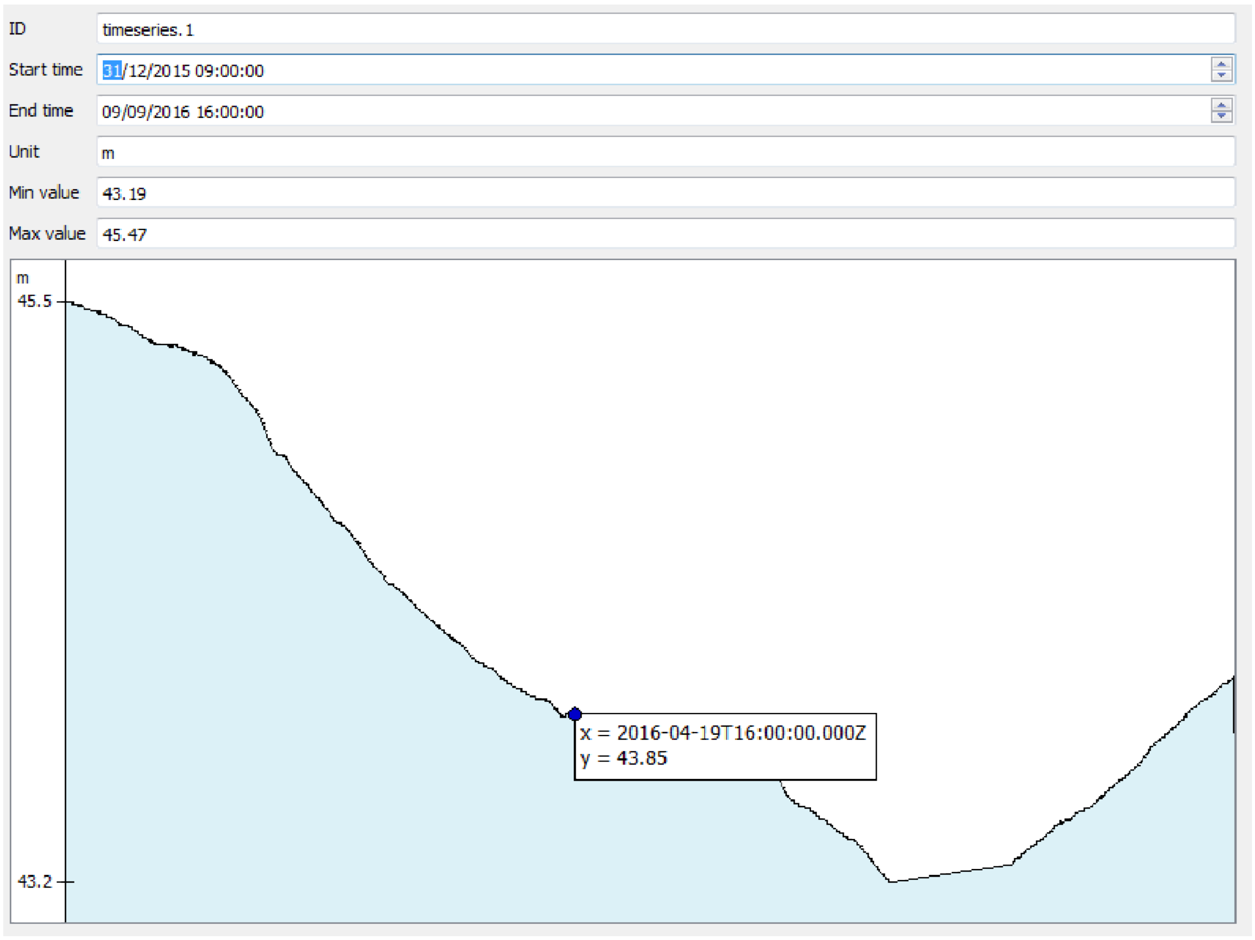
TimePeriodType

MeasurementTimeseriesType

Columns | 1:N Links | Back Links

metadata_MeasurementTimeseriesMetadata_temporalExtent_href	#phenomenonTime_1
id	timeseries.1

Illustration 12: Custom viewer button in relational mode



ID: timeseries.1
Start time: 31/12/2015 09:00:00
End time: 09/09/2016 16:00:00
Unit: m
Min value: 43.19
Max value: 45.47

Illustration 13: Example of the display given by the WaterML2 time series viewer

A custom viewer is made to be linked to a particular type (or table name). It receives in input a handle on the XML subtree or on the table and should return a Qt widget.

Use case – attribute-based symbology

We present here a use case where the user wants to extract useful information from the Complex Feature stream and use it to configure the symbology of the geometry features in QGIS.

Data are coming from a sample XML instance implementing the INSPIRE "EnvironmentalMonitoringFacility" theme (see BRGM_environmental_monitoring_facility_piezometer_50.xml example on the github).

The idea here is to extract all the titles of the "mediaMonitored" of each measurement and use them as a label.

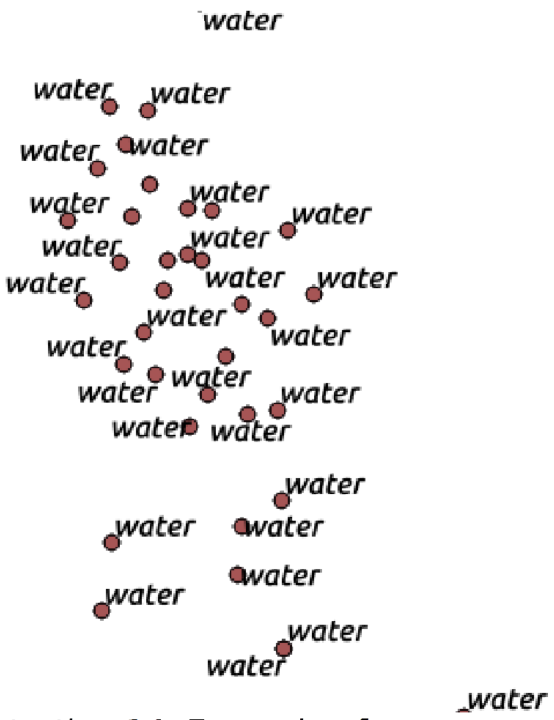


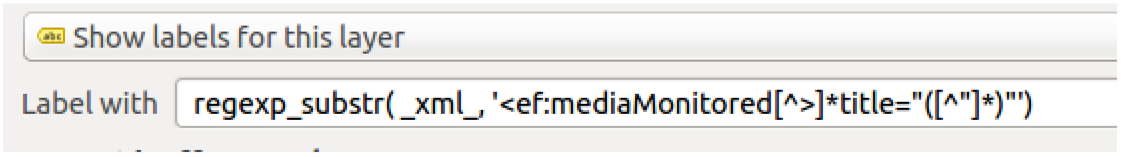
Illustration 14: Example of EnvironmentalMonitoringFacility features with labels extracted from the data

XML mode

In XML mode, the only currently possible way to extract data from the feature is to use QGIS to parse the "_xml_" attribute that contains the XML stream as a string.

Using a label based on an expression with something like the following would do the trick.

```
regexp_substr( _xml_, '<ef:mediaMonitored[^>]*title="([^"]*)"')
```



Label with regexp_substr(_xml_, '<ef:mediaMonitored[^>]*title="([^"]*)"')

The expression relies on regular expression parsing to extract useful data from the raw XML content.

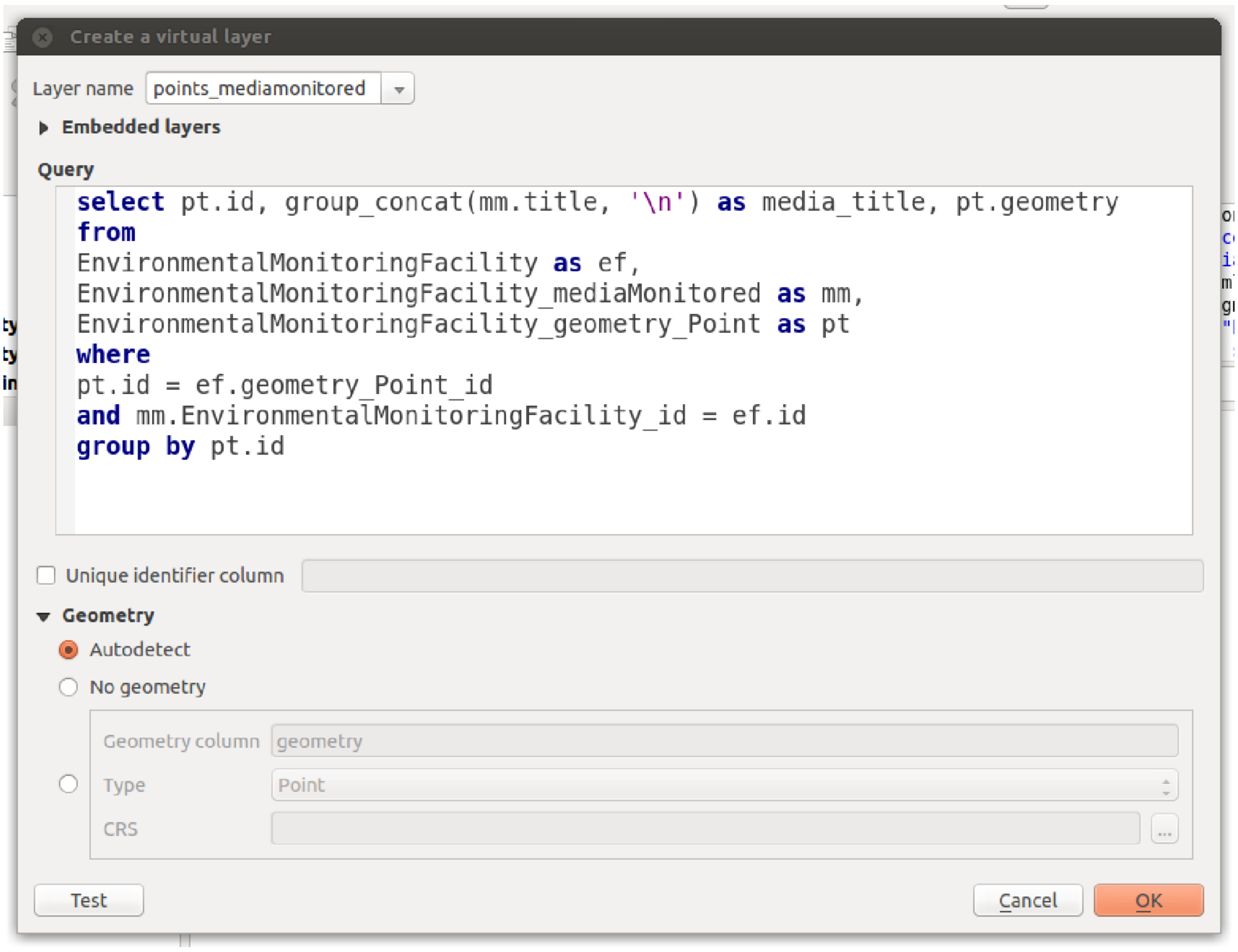
Regular expressions are not always easy to design and the text parsing required could lead to performance issues. **New functions to manipulate XML streams**, like XPATH resolution could be of great help here, especially since the XML tree widget allows the user to copy the Xpath of a given XML element, this can be reused easily in QGIS expressions.

Relational mode

In relational mode, accessing “media monitored” values of a feature requires joins on different tables. These joins may be implemented by the addition of a view to the Spatialite database.

This can also be done by using a virtual layer in QGIS, with the following query:

```
select pt.id, group_concat(mm.title, '\n') as media_title, pt.geometry from

EnvironmentalMonitoringFacility as ef,

EnvironmentalMonitoringFacility_mediaMonitored as mm,

EnvironmentalMonitoringFacility_geometry_Point as pt

where

pt.id = ef.geometry_Point_id

and mm.EnvironmentalMonitoringFacility_id = ef.id

group by pt.id
```



Layer name: points_mediamonitored

Query

```
select pt.id, group_concat(mm.title, '\n') as media_title, pt.geometry
from
EnvironmentalMonitoringFacility as ef,
EnvironmentalMonitoringFacility_mediaMonitored as mm,
EnvironmentalMonitoringFacility_geometry_Point as pt
where
pt.id = ef.geometry_Point_id
and mm.EnvironmentalMonitoringFacility_id = ef.id
group by pt.id
```

[ ] Unique identifier column

[✓ Autodetect] No geometry

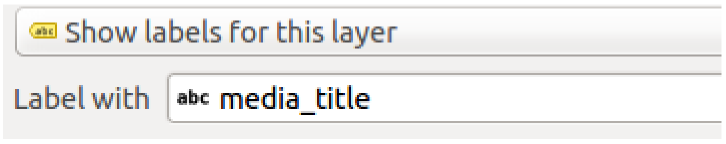
Geometry column: geometry

Type: Point

CRS:

This creates a new layer acting as a view with three columns: *id, media_title* (the useful information extracted) and the associated geometry.

On this new layer, we can use the *media_title* column as a label to get an equivalent result.



Show labels for this layer

Label with: abc media_title

Perspectives

Integration with existing open source applications

- The code that is responsible of the conversion to a relational database has been isolated in an independent Python library (called gml2relational). The conversion from GML to a relational model could be seen as a part of more low-level components like OGR/GDAL.
- For the native XML mode, the plugin creates a dedicated XML tree widget to view and interact with the XML stream. The regular “form” view of the attribute table is then overloaded by the plugin. But this way of doing is hacky and has no guarantee to work in future versions of QGIS. It may then be interesting to extend QGIS so that custom widget types may be registered by Python plugins.
- In a similar way, custom viewers for time series could be more integrated in QGIS if it was aware that the standard “form” of a layer may be overloaded by a custom widget provided by a Python plugin.
- QGIS expression functions could be extended to add native XML requests like queries based on Xpath (or XQuery)

PostGIS support

The conversion to a database model is only supported for the Spatialite format right now.

It should be extended to also be able to export the model in a PostGIS schema.

Since the part of the plugin code that creates the Spatialite database is an independent function reading a relational model and transforming it to SQL statements, adding PostGIS support has no identified problem so far.

Edit mode support

The plugin currently imports Complex Features as Spatialite layers. Those can of course be modified by the user as he wants. But without any guarantee on the validation against the original XSD schemas.

In XML mode, the XML elements and attributes cannot be modified. However, there would be no technical obstacles for an evolution in that direction. The XML tree widget would have to serialize the tree into a character string upon each modification.

In relational mode, nothing prevents the modification of the existing values. But there is no possibility to export the modified values back to its original XML form.

Allowing the edition of values would require the modifications to be tested against the original XSD schemas.

Moreover, adding a new feature to the set of existing Complex Features could lead the model to be modified, since the new feature may use elements or attributes that are optional and have not been set in the other features.

So adding a new feature may lead to remodel all the database structure in relational mode. Removal of a feature may also lead to a structure change if one wants to optimize the overall database structure.

In XML mode, insertion of new elements will require the plugin to be aware of the underlying application schema (which is not the case currently).

External resource resolution in relational mode

There is currently no way to resolve an href link when the Complex Features stream is imported in relational mode.

Supporting this feature would require the plugin to be able to restructure the database schema dynamically, something close to the requirements of the edit mode.

User interface

The relational mode pushes QGIS forms to their limits.

When the underlying data structure is too rich, nested forms become difficult to use. New data representation techniques should be investigated.

External resource resolution could be configurable by allowing to resolve them automatically until a certain depth. A network timeout should also be configurable by the end user.

The “content negociation” should also be based on user settings (MIME type to use for the content, language to force, etc.)

Links between the two modes

In relational mode, it is not always clear for the end user what XML elements correspond to a layer. The graphical representation of the relational model helps to recall the link from the relational model to a layer, but making the link with the XML stream is not simple.

Access to the attribute table of a vector layer coming from a relational model may be extended to include a visualisation of its XML counterpart.

The two modes are currently seen as two independent modes. We could imagine adding the possibility to switch between these two modes.

Following this idea, the user could manually drive the conversion process with an “on-demand” mode where he would choose which sub-tree of the XML stream has to be converted to a relational model.

The inverse operation, converting a relational model back to its XML form, could also be added in a future version.

Furthermore, the relational mode needs a valid XML to work properly (XML/XSD validation). When using a well-formed but not valid XML, better interlinkage between the two modes would allow the plugin to automatically switch to a 'degraded' mode where the XML mode will allow the user to at least start exploiting the file content instead of being stuck with an error message.

An open source project

The “GML application schema toolbox” plugin has been developed as an open source plugin under the terms of the GPL license.

Its source code is published on the github platform.

https://github.com/Oslandia/gml_application_schema_toolbox

The “issues” feature of the github project is used by developers and users to interact around issues, either bug declarations and fixes or requests for enhancements.

The plugin will be added to the official QGIS plugin repository and will then be available from the official list of plugins.

As an outcome of this prototyping exercise, a direct follow-up project financed by the EEA is starting during the summer 2016.

This new project aims at including the support for GML application schema compliant flows in OGR (simple, complex feature); porting the experience gained in developing the first release of QGIS GML Application Schema Toolbox.

The new OGC driver foreseen name is “OGR GMLAS driver” for OGR GML Application Schema driver.

The GML data handling part of the toolbox will have to be updated so that the interface continues to work properly.