

*Prestudy*

# **Automagic aggregation of Reportnet deliveries**

**Prepared by**  
Søren Roug

**November 2008**  
**Version 0.4**

European Environment Agency



## Version management

<b>No</b>	<b>Date</b>	<b>Changes</b>	<b>Author</b>
0.1	21.04.2008	Initial release	Søren Roug
0.2	29.04-2008	Traceability, references and data types	Søren Roug
0.3	27.06-2008	Description of new workflow	Søren Roug
0.4	02.07-2008	Description on how to dereference country code	Søren Roug
0.5	14/11/08	Explanation of trust	Søren Roug

# Contents

<b>1. INTRODUCTION.....</b>	<b><a href="#">4</a></b>
1.1. GOALS.....	<a href="#">4</a>
1.2. CONSTRAINTS.....	<a href="#">4</a>
1.3. DEFINITIONS.....	<a href="#">4</a>
<b>2. THE WORKFLOW IN A DATAFLOW.....</b>	<b><a href="#">5</a></b>
2.1. WORKFLOW FOR XML DELIVERIES.....	<a href="#">5</a>
2.2. NEW WORKFLOW FOR XML DELIVERIES.....	<a href="#">6</a>
<b>3. IMPORTING THE FILES INTO THE AGGREGATION DATABASE.....</b>	<b><a href="#">6</a></b>
3.1. WHAT DOES CONVERTING TO RDF MEAN?.....	<a href="#">7</a>
3.2. QUERYING THE DATABASE.....	<a href="#">8</a>
3.3. UPDATING A STATION.....	<a href="#">9</a>
3.5. HANDLING CHANGES IN FORMATS.....	<a href="#">10</a>
<b>4. TRACEABILITY.....</b>	<b><a href="#">10</a></b>
4.1. EXAMPLE.....	<a href="#">10</a>
4.2. TRUST.....	<a href="#">11</a>
<b>5. USER INTERFACE TECHNOLOGY.....</b>	<b><a href="#">11</a></b>
5.1. DOING THE QA.....	<a href="#">12</a>
<b>6. DATATYPES.....</b>	<b><a href="#">12</a></b>
<b>7. REFERENTIAL RELATIONS (JOINING TABLES).....</b>	<b><a href="#">13</a></b>
7.1. DEREFERENCING.....	<a href="#">13</a>
7.2. EXAMPLE.....	<a href="#">14</a>
7.3. SETTING UP RDBMS REFERENCES.....	<a href="#">14</a>
<b>8. HOW DOES IT FIT IN WITH SEIS?.....</b>	<b><a href="#">15</a></b>
8.1. A NEW ROLE FOR DATA DICTIONARY.....	<a href="#">16</a>
<b>9. CONCLUSION.....</b>	<b><a href="#">16</a></b>

# 1. Introduction

The issue of aggregating the national deliveries to Reportnet is a particular tough nut to crack. In the old days (actually also today) the deliveries come in as MS-Excel files, and someone has to copy-paste all the spreadsheet rows into one large spreadsheet – avoiding column headings – and ensuring that each row is still properly associated to a country and a reporting period.

When reporting XML, the problem gets even worse as there are no native tools to do this operation. As Hermann can attest to, we're back to scripting with general purpose programming languages such as *awk*.

Another complication is that in periodical dataflows, what is collected can vary over time. More data can be collected or the methodology to calculate the values can change.

I intend to show *a possible method* by which XML-data in a hierarchical structure is sent into Reportnet's Content Registry (CR), gap-filling is done and the data is extracted in table-structured format ready for relational databases.

## 1.1. Goals

- We want to be able to look at all data without data types getting in the way for QA. E.g. sometimes the reporter writes n/a in a numeric field.
- We want that the data types are used when the final conversion to relational tables is done.
- We want to be able to trace back all values to the source.

## 1.2. Constraints

Risk: To be able to use the Semantic Web for relational data we might have to constrain the data structures to fit the model of relational data. This means that fields inside a table can occur only once. I.e. a predicate can not be repeated for the same subject.

## 1.3. Definitions

CDR	Central Data Repository
DD	Data Dictionary
QAW	Quality Assessment Workbench A future Reportnet module and website.
SEIS	Shared Environmental Information System – is also a development project

### 1.3.1. What is aggregation?

The process by which data values are collected with the intent to manage the collection as a single unit. Example: The combination of data for the same area extracted from multiple sources.

### 1.3.2. What is gap-filling?

When data for a variable is missing for one or several countries in a certain period, the requester uses a method for filling the data gap, which is based on the assumption that the growth of the variable from a period for which data exists has been the same as the average growth for those other countries in the same regional or alternative grouping, where data exists for both periods.

### 1.3.3. What is DMM?

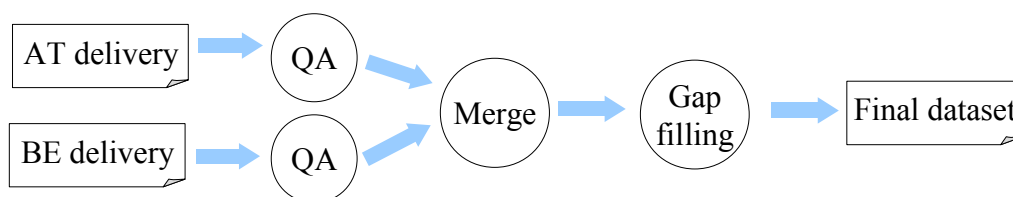
The data merging module (DMM) was designed and developed as a mechanism to aggregate XML files from a delivery. The user could in an interactive session query CDR for deliveries for a dataflow, period and country. He would get a list of results. In case a country has delivered twice, he must decide which one is the correct one. Then he downloads and stores the result in a relational database.

### 1.3.4. What is a Reportnet/SEIS-node?

The SEIS project's purpose is to collect data that is stored as close as possible to its source. To that end the Reportnet project has made specifications for the participating organisations describing four types of repositories and what features they must have. A SEIS node is a repository for files that follows the requirements for one of those types.

## 2. The workflow in a dataflow

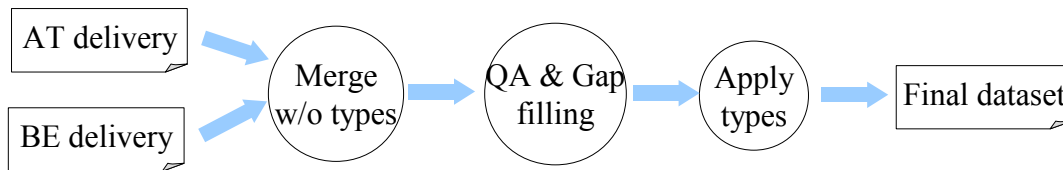
It is my observation that the manual QA work on deliveries tend to follow the same patterns. For a dataflow that is delivered as spreadsheet or Access database, this is what happens:



The QA operation is typically also a touch-up of the values, such as moving columns around, changing “n/a” to NULL etc. Then when the spreadsheets have become “normalised” they are manually merged with cut-and-paste to a fresh spreadsheet and the gap-filling starts.

### 2.1. Workflow for XML deliveries

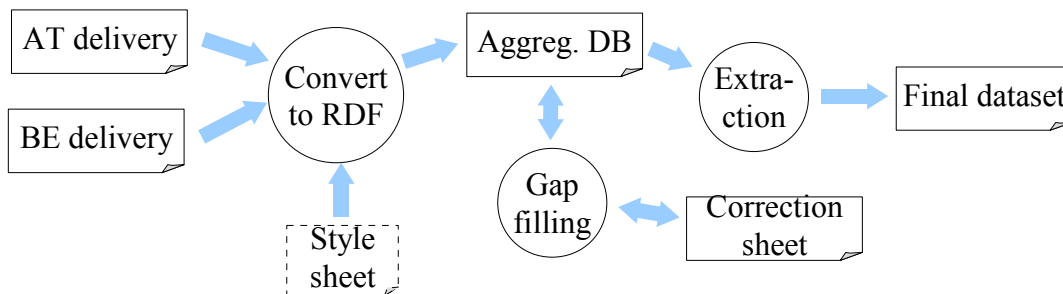
The typical workflow for XML deliveries is somewhat different. We download all XML files from all relevant deliveries. Then we write a script that converts the XML files to a single aggregated SQL database (Art. 17) or a comma separated file (Water Framework Directive). From that database we do our QA and gap filling and produce the final dataset.



Some dataflows are somewhere in between. Reportnet typically makes it possible to convert an XML file to spreadsheet, and if the requester does this, the dataflow will follow the first flowchart.

## 2.2. New workflow for XML deliveries

The task here is to figure out what we can do about the manual merge-step. Here is my vision. The way it is anticipated it will work is that as part of designing the dataflow, a stylesheet that converts the XML delivery format to a table structured RDF format is also implemented. Then when a delivery is made to CDR or another Reportnet node is made the system will automatically run the conversion and import all the values into the Aggregation Database, which will be a new role for the Content Registry (CR). Every value will have meta-data attached to it. One wellknown metadatum you know from spreadsheets is the datatype. But in addition, this system will also record the URL of the file the value originates from.



Gap-filling will happen like this: The user creates a correction sheet for the gap filling operation. He will look at the data by querying the Aggregation database, and when he sees something that needs to be changed he will do it via the user-interface, but the system will write the change into the correction sheet. The correction sheet will have a URL and will be loaded into the Aggregation database as if it was a real delivery.

The extraction part will then be a manually initiated procedure. In the user-interface, the AT delivery, the BE delivery and the correction sheet will be seen as sources to merge from. The user will be able to preview the result, and can turn some of the sources on/off. For instance, if there are two correction sheets, the user can choose one, two or none of them. Then he decides what format he wants and runs the extraction.

## 3. Importing the files into the aggregation database

The real magic is in the conversion of the XML files into RDF and the subsequent import into the database. I'll walk you through an example of how it is done. This example is so

simple that you might wonder why we go to such length to get essentially the same out as we put in. But I'm keeping this first example simple to focus on the transitions.

Let's imagine that Austria reports a new Airbase station. They will upload an XML file to CDR that looks like this:

```
<stations xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://air-
climate.eionet.europa.eu/databases/o3excess/stations.xsd">
  <station>
    <local_code>32301</local_code>
    <name>St. Pölten - Eybnerstraße</name>
    <longitude_decimal>15.63166</longitude_decimal>
    <latitude_decimal>48.21139</latitude_decimal>
    <altitude>270</altitude>
    <station_type_eoi>Industrial</station_type_eoi>
    <area_type_eoi>urban</area_type_eoi>
  </station>
</stations>
```

There is no EoI code as that one is assigned centrally later on.

Keep in mind that some XML files are a lot more complex than this example. We would like to import this data into a relational database. To do this operation someone has written a XSLT stylesheet that flattens the hierarchical XML into RDF and also uses common element names for those fields that share a concept with other dataflows – such as latitude/longitude/altitude coordinates. A stylesheet will have to be written once for every XML format that Reportnet knows about. We do something like this already for CDR to show a delivery as a web page. It is simple, but it requires domain knowledge.

Hence the above XML becomes:

```
<air:Station
rdf:about="http://cdr.eionet.europa.eu/at/eea/envxq1/stations.xml#32301">
  <air:local_code>32301</air:local_code>
  <air:name>St. Pölten - Eybnerstraße</air:name>
  <geo:long>15.63166</geo:long>>
  <geo:lat>48.21139</geo:lat>>
  <geo:alt>270</geo:alt>
  <air:station_type_eoi>Industrial</air:station_type_eoi>
  <air:area_type_eoi>urban</air:area_type_eoi>
</air:Station>
```

### 3.1. What does converting to RDF mean?

Every record gets a *type*. This one is called <air:Station>. It will become important when we insert the record into CR. Every record gets a *globally unique* identifier. The purpose is to generate an anchor point at which all the attributes are attached. In this case we generate it from the filename of the delivery and local code (32301) to be <http://cdr.eionet.europa.eu/at/eea/envxq1/stations.xml#32301>, but for the brevity of the example we'll show it as #32301.

Why use common element names? The <geo:long>, <geo:lat> and <geo:alt> elements are as you can see different from <longitude\_decimal>, <latitude\_decimal> and <altitude> used in the XML. The idea is that by giving all forms of lat/long values the same element name, then we can add business logic to CR to give special treatment to lat/long values. We can for instance develop a query that finds all records in the vicinity of a location. It just so

happens that the element names decimal degrees are already standardised by another organisation. The W3C.

Now that we have the data on RDF format, we import it into Content Registry. CR operates on a database structure known as *triples*. You can add any conceivable table structure into triples without declaring a table structure first.

The reformatting that is occurring internally in CR is that it takes every value linked to the record's key and adds it as a subject-predicate-object row, where the subject is the record's key, the predicate is the column name and the object is the field value.

*"Flattened" station structure*

Identifier	air:local_code	air:name	...
#32301	32301	St. Pölten - Eybnerstraße	...
#32302	32302	Europaplatz	...

*Triples*

Subject	Predicate	Object	Source
#32301	type	air:Station	stations.xml
#32301	air:local_code	32301	stations.xml
#32301	air:name	St. Pölten - Eybnerstraße	stations.xml
#32302	type	air:Station	stations.xml
#32302	air:local_code	32302	stations.xml
#32302	air:name	Europaplatz	stations.xml

With a little bit of development work, CR can be changed to do this automatically: When a new file is uploaded to CDR, CR gets notified and it would then run the conversion to RDF and import it. It will be a continuous process. CR will have all past and present data in its database.

OK, so the triples database doesn't have but three columns. Well, the source column is a meta-data column, and in the example it is the most significant. There are a few more, such as whether the object field is a resource or a literal.

### 3.2. Querying the database

The question is how do we make use of it? Here we must implement a query language in CR. If we start by something simple, such as a query on all records of type air:Station, CR will then find all rows with the predicate="type" and object="air:Station". It would then for all the subjects it has found look up which predicates exist and show a table.

Identifier	Local code	Name	...
#32301	32301	St. Pölten - Eybnerstraße	...
#32302	32302	Europaplatz	...
...	...	...	...

The data could then be exported as MS-Excel, MS-Access or whatever else is convenient for the user.

If the amount of data is too large, the query language should be able to add additional filtering. For instance, if the topic centre has already processed new stations up to 1<sup>st</sup> of May, it can decide to only see the rows where `released > 2008-05-01`.

Many other operations could be made. For instance if the dataflow allows the country to only report later modifications to stations, you could see a row for 2008-06-29, where only the altitude is reported because they made a better measurement. Then it could be possible to instruct CR to merge two records, making the newer information override the older.

Let's try that.

### 3.3. Updating a station

Let's assume that the station is moved to Mühlweg. The Austrian NFP will upload an XML file to CDR with the new information:

```
<stations xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://air-
climate.eionet.europa.eu/databases/o3excess/stations.xsd">
  <station>
    <local_code>32301</local_code>
    <name>St. Pölten - Mühlweg</name>
    <longitude_decimal>15.62880</longitude_decimal>
    <latitude_decimal>48.21194</latitude_decimal>
    <altitude?/>
    <station_type_eoi/>
    <area_type_eoi/>
  </station>
</stations>
```

When the new file is harvested by CR we now have two records:

Identifier	Updated	Local code	Name	Area	Long	Lat
#32301	2008-04-30	32301	St. Pölten - Eybnerstraße	urban	15.63166	48.21139
	2008-05-28	32301	St. Pölten - Mühlweg		15.62880	48.21194

The uploaded file will have a different URL and a different upload date than the first file. What we have to do is to tell CR to discard all fields with an update that is newer when we extract the data.

Now what with the `<area_type_eoi>`? In the first delivery it was 'urban'; in the second update it was the empty string. How will CR know not to update the record with the empty string?

The answer is that it doesn't, but that isn't CR's fault. The delivery format for this particular dataflow is ambiguous. It doesn't distinguish between 'unanswered' and

‘unchanged’. Both are the empty string. It is then up to the RDF-producing stylesheet to indicate in the RDF what is the intention. A complication that requires domain knowledge when constructing the RDF stylesheet.

### 3.5. Handling changes in formats

Over time, the delivery formats for periodic dataflows change. It could be that a new chemical is being monitored, or the methodology changes. In the first case, a new predicate will appear in the output. In the latter case, to avoid conflation between the two ways of measuring, the RDF stylesheet would be changed to use a different global predicate for the new values.

## 4. Traceability

The system will have traceability built in. As you saw above the Triple-table has an additional meta-data column called source. It contains the URL of the file the value comes from. You will therefore for *every value* in your CR queries be able to know which file the value comes from.

### 4.1. Example

To illustrate the concept I use colours to show the source of origin of the values.

Original declaration of new Ozone stations:

Identifier	Local code	Name	Area	Long	Lat
#32301	32301	St. Pölten - Eybnerstraße	urban	15.63166	48.21139
#32302	32302	Europaplatz	urban	15.61111	48.20999

The EoI code is provided from central authority.

Identifier	Eol	Updated
#32301	AT44	2008-05-03
#32302	AT220	2008-05-03

The 32301 station is moved to another location – Mühlweg:

Identifier	Updated	Local code	Name	Area	Long	Lat
#32301	2008-05-28	32301	St. Pölten - Mühlweg		15.62880	48.21194

Merging of all records based on update timestamp.

Identifier	Eol	Updated	Local code	Name	Long	Lat
#32301	AT44	2008-05-28	32301	St. Pölten - Mühlweg	15.62880	48.21194
#32302	AT220	2007-05-31	32302	Europaplatz	15.61111	48.20999
...		...	...	...	...	

The mechanism is similar to the layers graphic designers have in their tools such as Photoshop. Whenever they want to add something to an image, they first create a new transparent layer, then draw on it. They can then make the layer invisible or discard it if they are unsatisfied with the result.

The same mechanism could be used by the Data Requesters when they clean up data and fill in gaps. We would have to think about the look and feel, but one scenario is like this: The requester sees the merged table like the one above. When he edits a field, it is then automatically saved to another file, and is visible. Let's assume he wants to specify clearer where Europaplatz is located:

Identifier	Eol	Updated	Local code	Name	Long	Lat
#32301	AT44	2008-05-28	32301	St. Pölten - Mühlweg	15.62880	48.21194
#32302	AT220	2007-05-31	32302	St. Pölten - Europaplatz	15.61111	48.20999
...		...	...	...	...	

The system would then allow the user to turn the grey layer on/off. Off: *Europaplatz*, On: *St. Pölten – Europaplatz*.

## 4.2. Trust

We have all the data from every source in CR. How do we trust what we see? In principle anyone can get a file harvested that claims to be the latest version of CO<sub>2</sub> emissions. It turns out that this is a derivative of traceability. If you have the source of every data element, then all you need is for someone to evaluate the source's reliability.

In the example 4.1 above you would then be able to check the source for the blue fields. You'd be able to read what the reviewers have written about the source, and who the reviewers are. If you don't like what you see, you turn the blue layer off in the user interface, and the application redraws the display.

## 5. User interface technology

Currently the vision is a spreadsheet-like interface with all sheet visible at the same time, and with transparent cells where there are no values. Each source will show as a sheet, which the user can turn on/off. They will all be read-only. If the user wants to modify he can create one or more writeable correction sheets that will be stored at the user's work area in QAW. He will be able to add some meta data to the sheet, such as description and methodology.

The sheets will be made available to other users as read-only sheets.

I envision the technology to be either Java running in the webbrowser or Flash. In any case, the application must only load the segment of the dataset the user looks at. I.e.:

- If the user has chosen to see a data set with 100.000 records in it, he can only see 100 at a time in the webbrowser, and the application should only load from CR the 100 records he's viewing, but it must be possible to scroll in the data set. The missing values should be loaded in the background, like with AJAX.
- If he does a search then the full dataset is searched.

- If he does a search-and-replace, the replaced values are written to the writeable sheet. The search and replace should ask if he wants to write to an existing writeable sheet or create a new one.
- It must be possible to highlight a segment of values and copy them to another column. Other typical spreadsheet operations should be possible.
- It should be possible to turn datatypes on/off per column and when turning them on get a warning about loss of information.

## 5.1. Doing the QA

The requester will begin his work by going to QAW at <http://qaw.eionet.europa.eu>. Here he will log in and create a QA project area. The project creation will ask two questions, the ROD obligation and the period. It will also create a default correction sheet.

The first operation will be to search all deliveries for the obligation and a first verification of the delivered file formats is done. If a country has made a delivery in an incompatible format, the requester will be able to convert the file with whatever tools he has available and upload the converted files to the project area and use them for the following operations. It will be possible to add the necessary meta-data to the files – i.e. country codes.

The system will show a list of RDF classes used in any of the deliveries. He will choose one and get a list of records having the class. As described in section 7.1 he will be able to dereference the sources and join their meta-data to the records.

Next step is the gap-filling described in section 2.2.

And finally an export to a product database with datatypes applied.

## 6. Datatypes

As explained, a triples database can load almost any data, but it will just end up in a sort of *dissaggregated* form. How do we apply some structure to it? Here RDF has answer too. We can write an RDF file called a schema that ties together the various predicates into a class-structure which you can then work on as a unit. Semantically it borrows a lot from object-oriented programming. Having defined the classes we can then follow section 2.9 of <http://www.w3.org/TR/rdf-syntax-grammar/> that specifies how to declare types on the object node of a predicate. We will only declare datatypes when we declare the properties. Essentially RDF points to a declaration in an XML schema, with all the ability to add restrictions. We would probably need a schema parser to extract the necessary information for a SQL CREATE statement.

For example the datatype of a predicate called countrycode could point to

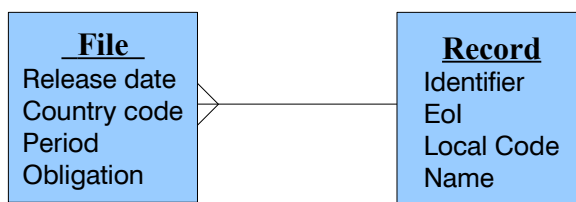
```
<xsd:simpleType name="cctype">
  <xsd:restriction base="xsd:string">
    <xsd:length value="2"/>
  </xsd:restriction>
</xsd:simpleType>
```

meaning the cctype type can be exactly two characters long. It would translated to the following SQL declaration:

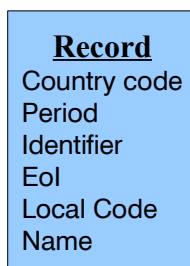
## 7. Referential relations (joining tables)

### 7.1. Dereferencing

The first problem you'll encounter when you try to merge two countries' deliveries is that the records typically don't have the country code on every record. This was the main problem that the DMM (Data Merge Module) solved; You would first query CDR on obligation. The result would be a list of deliveries for which you selected some. Then when merging the DMM would automatically add country, delivery date and period columns to every table. Unfortunately DMM merging only worked when the file format was defined in DD, and only when querying CDR via XML-RPC. In a SEIS environment you can't assume the repository has the necessary functionality. For some of the repository types, the only requirement is that the file is located at a persistent URL. But the country code, period and obligation information are also stored as meta-data for the file. All the metadata about the delivery is registered at the QAW either indirectly or entered manually and discovered by CR.



We can model the records contained in the file as a one-to-many relationship, where the “meta-data” on the file is seen as data in a “File” table. This can then be joined with the records to create a new view:



The new columns that amend the records would go into a correction sheet.

What would be required is that when the conversion to RDF is done of the file content, one field per record is added pointing to the file URL with an `rdf:resource` attribute. When the user looks at a list of the aggregated records he'll be able to follow the pointers to access the values of the File object. This is called *dereferencing* in computer science. He will have the ability to decide which fields in the File object to import and the system will do it for all records at once. One such implicit pointer to the file URL is the source-field in the triples database. The source actually exists on every field, and can be different from field to

field. But if we choose a field that is not likely to be corrected, such as the `<rdf:type>`, then it would work.

## 7.2. Example

Meta data on the Ozone files:

Filename	CC	Release date	Title
envxq1/stations.xml	AT	2008-04-30	Ozone stations 2008
envywp/stations.xml	AT	2007-05-31	Ozone stations in 2007

Original declaration of new Ozone stations:

Identifier	Source	Local code	Name	Area	Long	Lat
#32301	envxq1/stations.xml	32301	St. Pölten - Eybnerstraße	urban	15.63166	48.21139
#32302	envxq1/stations.xml	32302	Europaplatz	urban	15.61111	48.20999

Dereferencing will show that the `isDefinedBy` is defined as an `rdf:resource`, and will give the user the option to join the CC, Release date and Title columns to the stations, giving this result:

Identifier	CC	Release date	Local code	Name	Area	Long	Lat
#32301	AT	2008-04-30	32301	St. Pölten - Eybnerstraße	urban	15.63166	48.21139
#32302	AT	2007-05-31	32302	Europaplatz	urban	15.61111	48.20999

The blue fields are added to the user's correction sheet, making it possible to discard the values again by turning off the sheet.

## 7.3. Setting up RDBMS references

When defining the `rdf:Classes` for the objects, we can also define how they relate relationally to other classes.

The purpose is to not just work with tables, but with datasets. By describing the relationships, when we look at one table in the webbrowser we can follow foreign keys to other tables. One example is to go from a measurement to the station that made the measurement.

The semantic web already has a mechanism for relations. But it relies on direct references from a property to a resource, which is a URI, and will look somewhat alien to humans who are used to the mechanism used in SQL.

What we will do is model SQL relations in RDF by creating foreign key declarations in the `rdf:Class` declarations. It would then be possible to turn a `rdf:Class` declaration into SQL's `CREATE TABLE` syntax and continue using the dataset in a relational database.

### 7.3.1. An ontology<sup>1</sup> for references

Subject	Predicate	Object
Class C	has_index	Index I
Index I	has_key	Property X
Index I	has_key	Property Y
Index I	key_type_is	Unique
Class D	has_index	Index J
Index J	has_key	Property X
Index J	has_key	Property Y
Index J	references	Index I

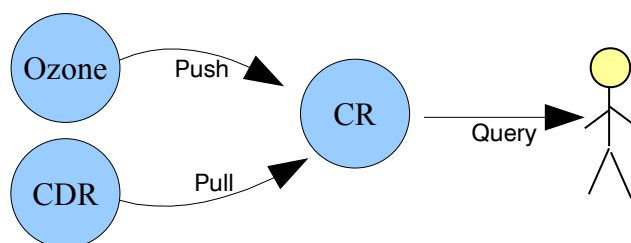
Assuming properties X and Y are integers, the SQL create statement would like this:

```
CREATE TABLE C (
X INTEGER,
Y INTEGER,
UNIQUE INDEX I (X,Y));

CREATE TABLE D (
X INTEGER,
Y INTEGER,
FOREIGN KEY J (X,Y) REFERENCES C (X,Y));
```

## 8. How does it fit in with SEIS?

SEIS operates with several types of dataflows. There are the dataflows that happen frequently – like once an hour. They can be either pushed to EEA or pulled by EEA. The canonical example is Hourly ozone. These fall outside Reportnet, as they have no human interaction and need a very lightweight handling. Then there is Reportnet dataflows. Slow periodical frequency of one month or longer. Finally there are on-demand use. The system at EEA knows the data is there, but hasn't collected it. This is a direct analogy to the GIS Web Feature Service. The aggregation database can handle the two first types of dataflows.



The reason is that CR is actually operating outside the Reportnet workflow. Updating CR doesn't have a ripple-effect as when releasing an envelope on CDR. CR is agnostic. It doesn't care if the data is clean or not, only that it can be represented as RDF.

<sup>1</sup> Ontology: An explicit formal specification of how to represent the objects, concepts and other entities that are assumed to exist in some area of interest and the relationships that hold among them. It is a common term in Semantic Web, and therefore used here also.

## 8.1. A new role for DataDictionary

The reason DD isn't more popular is that it is table-oriented and this creates delivery formats that can be very inconvenient to deliver in. For instance, for Article 17 we created 3 XML delivery formats, one for species, one for habitat types and one for the general report. When we then converted the XML formats to tables, we ended up with ~60 tables.

We *do* eventually need to get the data into tables for the Final dataset, and here is where DD can have a new role. Part of the conversion to RDF described in section 2.2 is a declaration of the table structure that is imported into the Aggregation database. It is actually a declaration of the *Final dataset*. DD could be used to define the *Final dataset* and then make the declaration available for the Aggregation database. It would help a lot in the writing of the stylesheet. In those cases where the delivery *is* in DD-generated format, the stylesheet could be auto-generated.

The main modification to DD is that elements must have *generic parent element*. For instance, the reason there is more than one country code element in DD is that they are implemented as a list of legal values. Some dataflows only allow the EU27 countries, others all European countries. This necessitates two different elements in DD. By making a generic country code element, and then subclassing the two elements with code lists from the generic, it is possible to use the subclassed elements for reporting, and the generic element for post-processing.

## 9. Conclusion

The core ideas expressed in this document can be summarised as follows:

1. Make a conversion of tree-structured XML data into a dissaggregated collection of values
2. Load those values into a semi-structured database
3. Apply a schema to describe of the structure to the database
4. Ensure all values can be traced back to their sources
5. Working with the data initially without datatypes then apply types in later phase
6. Write gap-filling corrections into a special correction document
7. Extract the data from the database as dataset of linked tables using the schema