|  |  |
| --- | --- |
| logo_ec_17_colors_300dpi | EUROPEAN COMMISSION  JOINT RESEARCH CENTRE  Institute for Environment and Sustainability (Ispra)  Digital Earth & Reference Data Unit |

Ispra, 27/06/2013

Re3gistry documentation

Contact: inspire-registry-dev@jrc.ec.europa.eu

## Purpose

This document provides details about the Re3gistry, an open source tool for the management of codelists and other key terms within the scope of the Infrastructure for Spatial Information for Europe (INSPIRE) Directive 2007/2/EC. A background to this work is provided (Section 2), followed by a description of the information model (Section 3) and a description of the overall system (Section 4). Import (Section 5) and Export (Section 6) functions are then described before focussing on the web service (Section 7). The last two sections list the technical components used in the Re3gistry (Section 8) and the installation instructions (Section 9).

## Background

The European Union (EU) Member States are currently implementing the INSPIRE Directive and related regulations. Technical guidelines for INSPIRE’s implementation, based on existing international standards, have been developed or are currently under development. Interoperability between systems is, however, being limited by varying ways of implementing standards, the regular evolution of standards and challenges in coordinating changes between standards, alongside varying choices in the technologies being adopted.

As part of the Interoperability Solutions for European Public Administrations (ISA) Programme[[1]](#footnote-1), the European Commission’s (EC) Joint Research Centre (JRC) is establishing a Reusable INSPIRE Reference Platform (ARE3NA) which is identifying and developing common components for the successful implementation of the INSPIRE Directive. To further address these interoperability issues and provide support to the Member States, the platform will provide guidance, collaboration, sharing of best practices and approaches and a reference implementation of common components through the following activities:

* Inventory of
  + Existing INSPIRE components from the Open Source community;
  + Components used within the Member States to implement INSPIRE;
  + Missing components;
* Selection of other policies and initiatives from other sectors (such as INSPIRE, Water Framework Directive[[2]](#footnote-2), Digital Agenda for Europe, open data, Shared Environmental Information System (SEIS)[[3]](#footnote-3) etc.;) requiring exchange and sharing and maintenance of spatial data sets and services.
* Selection of the missing components and/or functionalities. Multilingual support is envisioned where required;
* Support Open Source projects to develop the missing items and produce the related documentation (installation guides and technical documentation in several languages);
* Selection and development where required of conformance test suites;
* Set up a collaborative platform to share and maintain the components

The work on the Re3gistry contributes to the third point by addressing a missing component of INSPIRE as an open source solution for use in other contexts, including those who want to manage multilingual code lists in various levels of public administration in Europe. The outputs of this work will also appear on the ISA programme collaborative platform, JoinUp, to aid wide re-use.

## Information Model

The Re3gistry is a software containing registers of items in different languages and tools to manage register items, their characteristics and their status. A simple representation of the system’s model is shown below (see Fig. 1), with the following paragraphs describing each entity that can be found in the system.

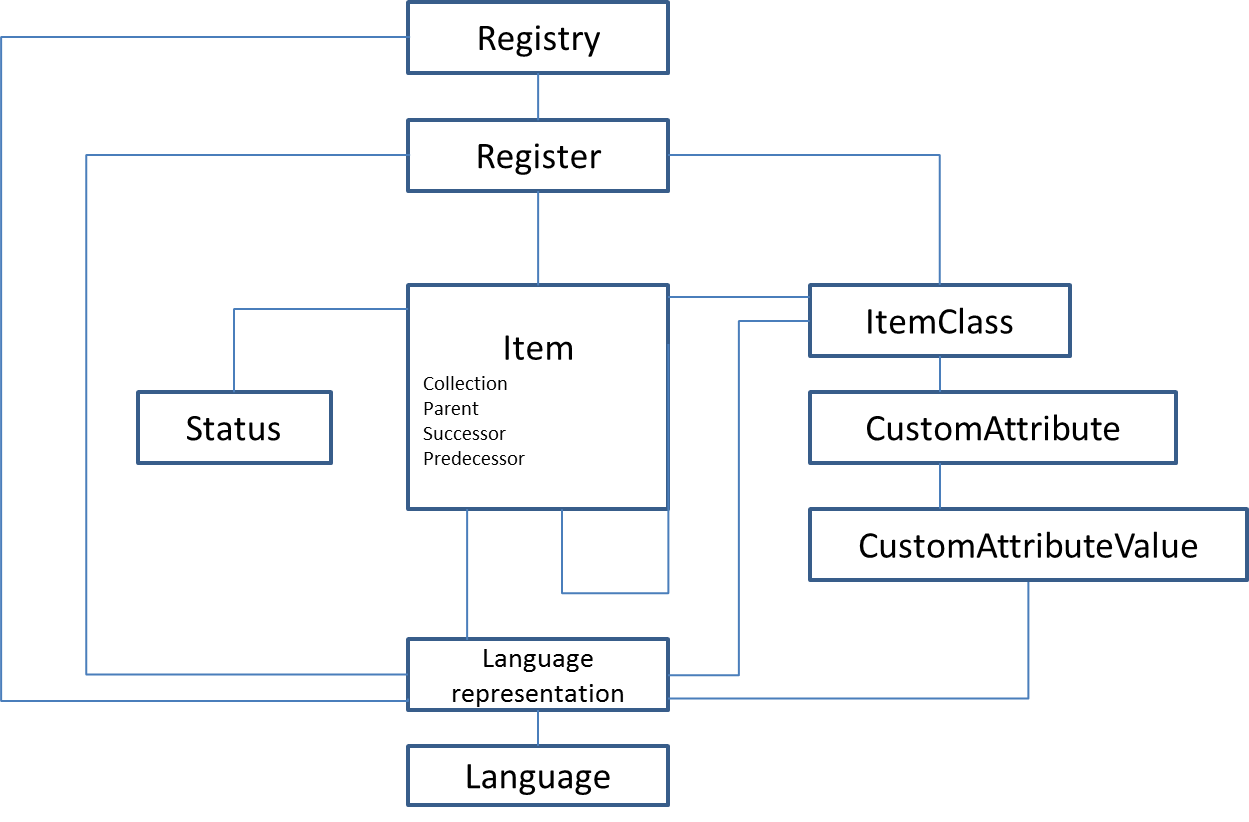


Figure 1 - System model[[4]](#footnote-4)

### Registry

The registry is the information system on which a register is maintained.

### Register

A register is a single controlled collection (list) of elements. Each register is operated on behalf of some owner organisation which provides the authority for the collection.

The type of element that can be entered in a register is completely open. For example, anything which can be given a Uniform Resource Identifier (URI) can be registered.

The register elements are called **items**. Each item has a specific **item class** that describes the type of the items.

### Item Class

The item class defines a group of items contained in the register. The item class is a description of a class of information items specified in a technical standard.

For example, in a “theme register”, such as a register for an INSPIRE Theme, the item class of the items contained can be referred to as “theme items”.

### Item

Items are elements that can be entered in a register. An item can contain other child items. For example, where the item class “codelist item”: each item with that class could contain other items. The contained item, the “codevalue item”, has the indication of the container in the collection\_id field (for detailed information refer to the complete information model in annex A). In this way there is a link between items in the same register.

### Status

Each item has a status; the statuses are taken from the ISO 19135-1 draft. Below there is the list of the available statuses:

* valid
* Invalid
* Submitted
* Superseded
* Retired

### Standard and Custom Attribute Fields

Each item has a standard list of fields common to all the items. The “custom attribute” fields are designed to extend the standard item fields. Using these custom attribute, the item can be extended to include any item-specific fields. The current standard list of fields is described in section 5.2.

### Language representations

Each component in the system (except for the status) can be represented in several languages. The localisation table contains all the languages related to the various elements in the system.

## Description of the system

The Re3gistry handles the entire data flow for code list management, from data import to the export of specific items.

The system has two main parts: a data - handling system and a web service.

The schematic details of the data’s import system, the export system and the RESTful web service that provides direct access to the registry are shown below (see Fig. 2).

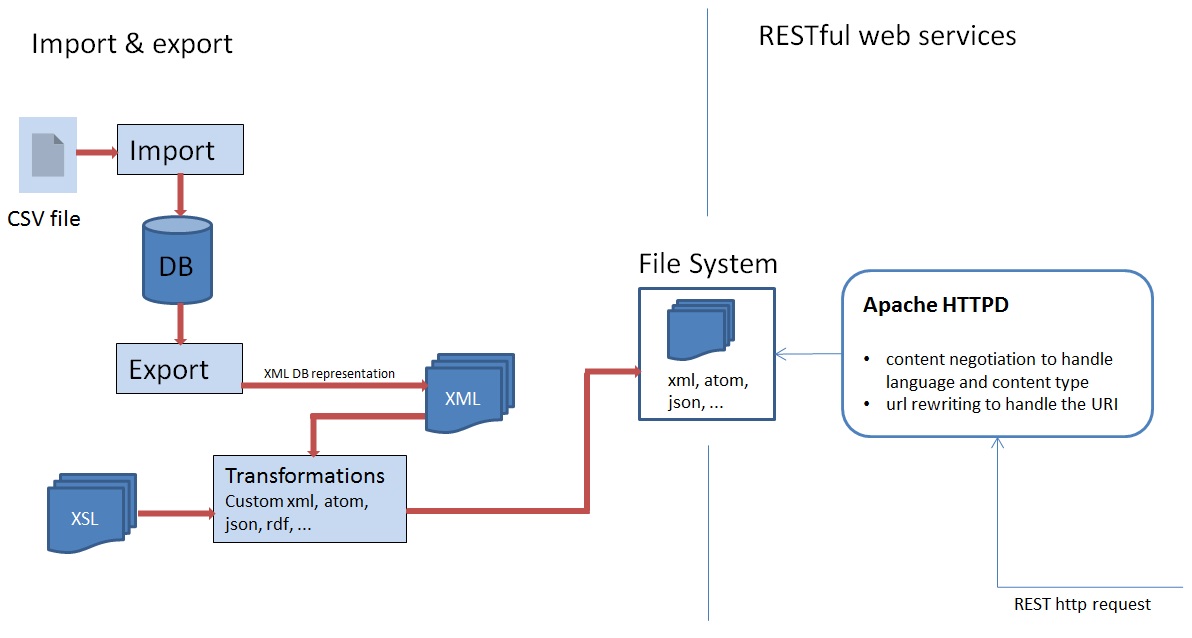


Figure 2 - System description

The description of the system is provided in the following paragraphs.

### Data handling

The data handling part is responsible for several operations described below:

* Parsing a comma separated value (CSV) file: based on the type of import, the parser reads and organises the information to be imported in the database. The import and the CSV format are described in section 5.2.
* Importing data in the database: once the data from CSV files are organized, the system stores them in the database.
* Exporting data: this function exports the data stored in the database in a generic XML format (as database representation).
* Transforming exported data: the generic XML format is then translated to other specific formats using some XSL files. In this way, the system can provide custom XML files (starting from the general XML format), as well as other format (either starting from the general XML or the custom XML produced by the first transformation).

### Web service

The Re3gistry web service part is responsible for serving the data to the INSPIRE community and other potential users. An important characteristic of the service is the possibility to provide the same information in multiple formats and multiple languages. The service is implemented using the content-negotiation approach. The server can provide the correct file language/format based on the http header sent in the http request.

If a user prefers to use the classic way for accessing a specific file, the system supports this approach, too.

Detailed information about the web service and its configuration are provided in section 7.

## Re3gistry - Import procedure

The Re3gistry - import procedure is an automatic process to import data from a source data file in CSV format.

The import system can be accessed by a web interface that allows users to specify the requested parameter, including the location of the source data file.

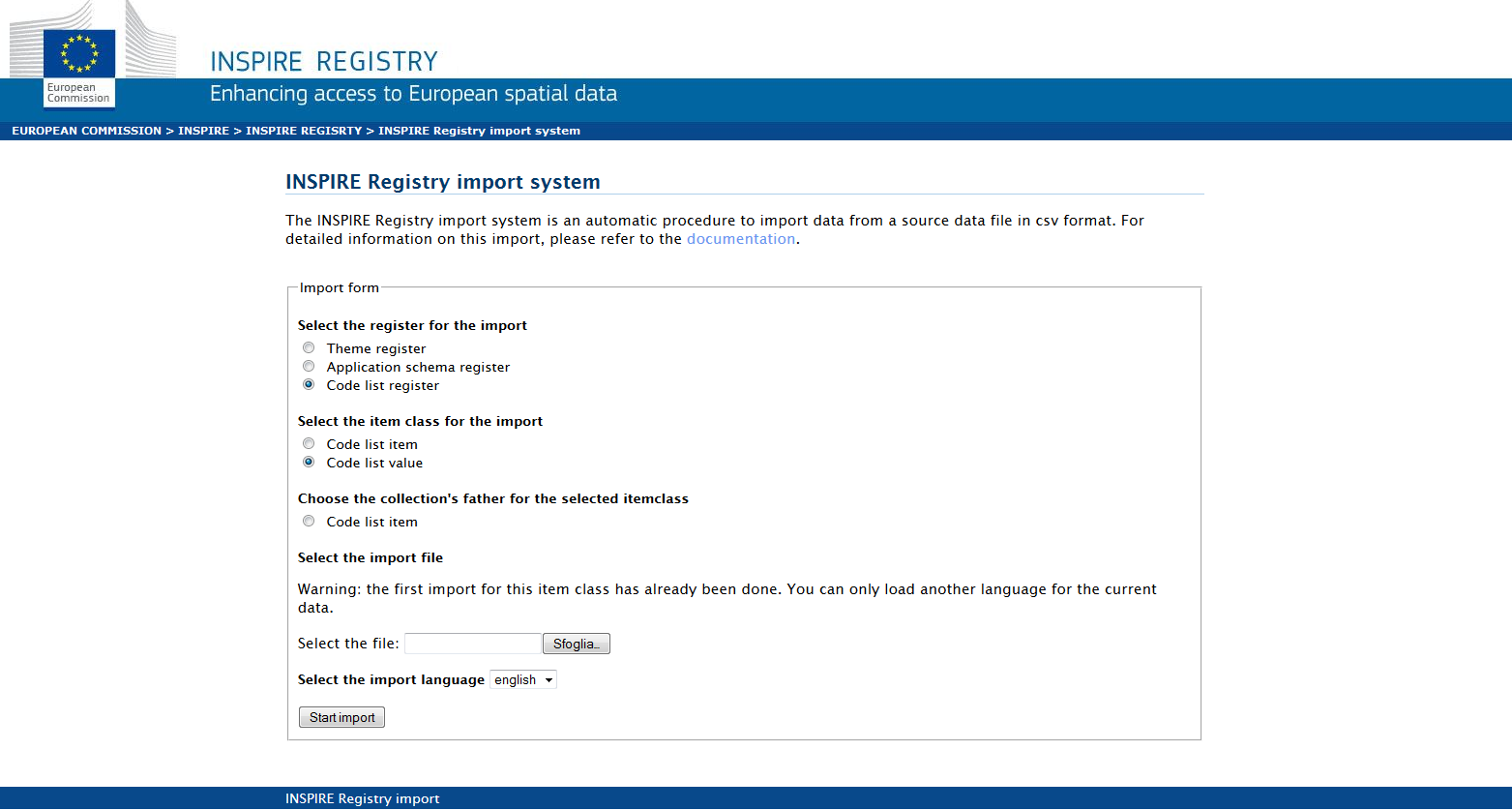


Figure 3 – Web interface for the Import

The web interface needs the following information in order to start the import procedure[[5]](#footnote-5):

* The register in which the data will be imported;
* The item class related to the chosen register;
* If the selected item’s class is part of a collection (a list of items contained within another item), the system ask also the collection parent;
* The file location;
* The language that the data is in.

### First run

After the installation of the system (section 9), the database has to be initialised with the base data that is needed to run the import. The SQL script to initialise the database is provided in the project package. This includes an example SQL file for the creation of the required tables; an example SQL file to load example data, as well as some registers and their required and related information[[6]](#footnote-6).

### Import file

Only csv files can be used for file input. There are two parts to the csv file: mandatory fields and additional fields (described below). The mandatory fields must be provided even if they are null values. The additional fields represent custom attributes that users can add.

The first line of the csv always contains the descriptor/header of the fields. This is important, because the custom attribute properties are specified in this header (as you can see from the examples below).

|  |  |  |
| --- | --- | --- |
| Type | Code | Description |
| **Mandatory fields** | LocalId | The local id is used to identify an element inside the register and to create the URI of the item. If an item has a LocalID such as “aas”, the URI will be composed as follow: “http://site.ext/register/aas”; |
| ParentLocalId | This is the reference to the parent item |
| CollectionLocalId | This is the reference to the collection parent. If this field is set then an item is part of a collection. |
| PredecessorLocalId | This is the reference to the predecessor item. This field is designed to support item versioning. |
| SuccessorLocalId | This is the reference to the successor item. This field also supports item versioning. |
| Label | This contains the label of the items |
| Definition | This contains the definition of the system |
| Description | This contains the description of the items |
| Status | This contains the status related to the current item. More details about the code to be used in this field can be found in the following table. |
| **Additional fields** | (custom attribute/ header descriptor) | The custom attribute headers define the properties of the additional fields.  Custom attributes must be added to the csv file after the standard fields and have the following format and are defined below:  **\*custom\_attribute\_name[required,multivalue,coded,foreignkey]**  The value in the square parenthesis is a Boolean value, and can be “t” (for true) of “f” (for false): e.g. **\*extensibility[t,f,f,f]** |
|  | Required | This indicates that these custom attributes are required: the whole row in the csv must have that value in order to be imported. |
|  | Multivalue | This indicates that this custom attribute could have more than one value (it is possible to have more than one custom attribute with the same name but different values) |
|  | Coded: | If this property is set to true, the custom attribute takes its values from a list of values stored in the “customattributecode” table. More information about the information model can be found in Annex A (This feature is not implemented in this version and is foreseen in the next versions). |
|  | Foreignkey | If this property is set to true, the value in this field is a reference to another element in the register (this field is usually filled with a LocalId of another item or with a URI). |

A complete example of a CSV file can be found in the project package.

**Status code reference table**

This table shows the code to be used in the “status” field of the import CSV file.

|  |  |
| --- | --- |
| **Status code** | **Status label** |
| **1** | valid |
| **2** | Invalid |
| **3** | Submitted |
| **4** | Superseded |
| **5** | Retired |

## Re3gistry - Export procedure

The export system produces a representation of the database creating static files in different formats.

In this system there are 2 export systems and a configuration file export. These export functions are described below (see Fig. 4).

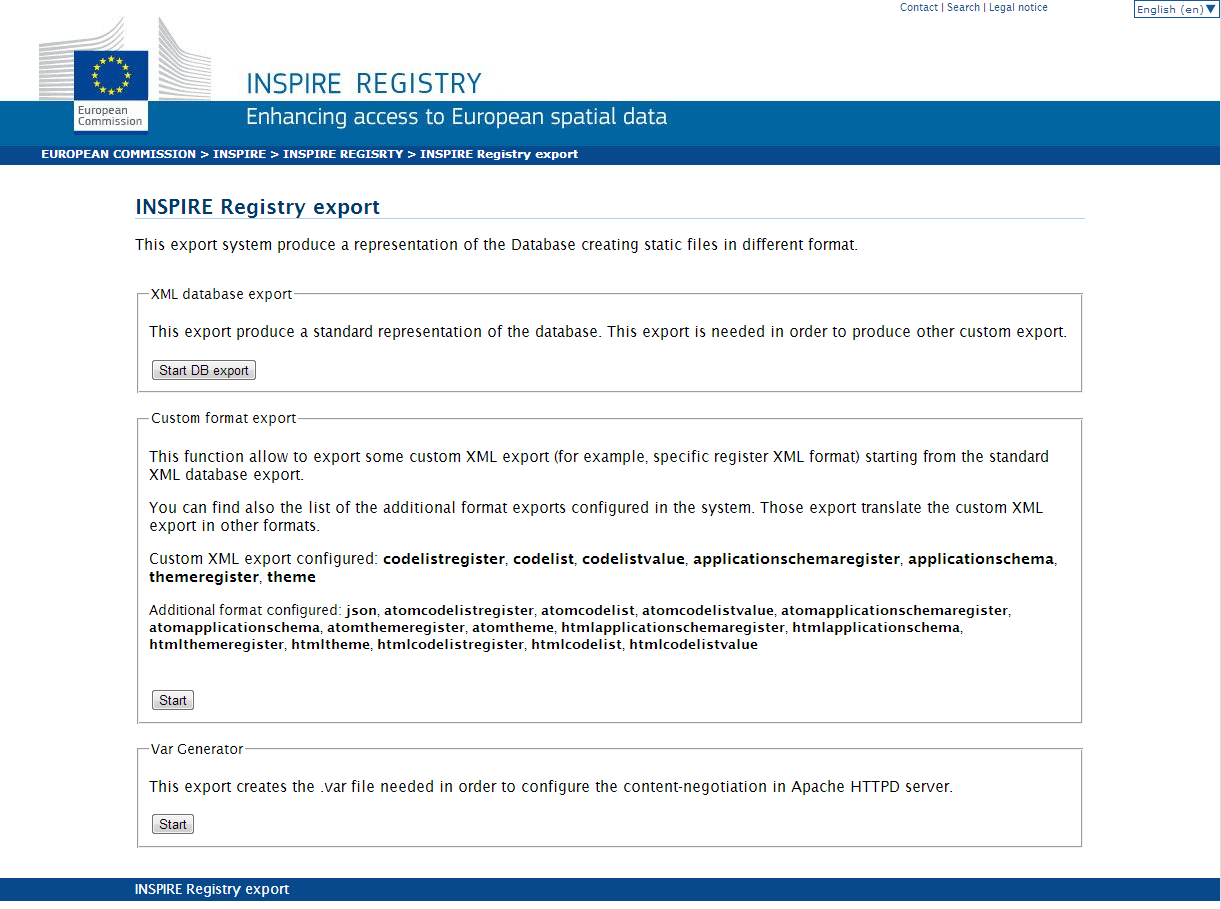


Figure 4 – Web interface for the Export

The **XML database export** will produce standard database representation xml files with the following structure.

The root folder for the export could be configured using the properties file (webapp.properties) in the project. If the custom export folder is not configured, the servlet context is taken instead.

The **Custom format export** allows users to export some custom XML files (e.g. a specific register XML format) starting from the standard XML database export.

The system can also be configured (via webapp.properties file) to produce some additional formats. Each export process could be configured to start from either standard or customer xml exports and to translate them into other formats.

All the export process and configurations are explained in section 6.2.

The **Var Generator** generates the .var index configuration files needed to set up the Apache content-negotiation feature. This allows the user to use the content-negotiation approach explained in section 7.1.

# File System Structure

Each entity has a different folder/filename labelled with the “uri\_name” database field. Each folder contains a complete register file and a list of folders, one for each item. Inside the item’s folder, there is a file describing that item.

## Transformation

### Transformation configurations

The transformation system allows the standard xml database export to be customised for a custom format export. This process uses XSLT files in order to convert the original xml files into other file formats.

The first transformation uses the standard xml database export as input and produces the specific xml for each register / item.

Each specific transformation has to be configured in the “webapp.properties” file. This file contains detailed comments for each property.

### Specific XML transformation

The first transformation performed is the xml specific representation transformation that uses the standard xml database export as an input and is configured as follows:

Each specific xml representation has to be listed in the “xsl.xml.list” property. This property tells the system how many transformations there are for the specific xml and how to find the related properties.

Example: xsl.xml.list=codelistregister-codelist

Each specific-format transformation also has a set of properties:

|  |  |  |
| --- | --- | --- |
| code | Description | Example |
| xsl.xml.filename.[transformation\_name] | The name (and the path starting from the xsl common path) of the xsl file to be used for the transformation | xsl.xml.filename.codelistvalue=xml/base\_to\_codelistvalue.xsl |
| xsl.xml.folderin.[transformation\_name] | The folder in which the transformations have to be performed. If this property is left blank, the root folder will be taken. The standard database export folder is organised in a subfolder. The first level represents the register, the following levels represent the items. | xsl.xml.folderin.codelistvalue=codelist |
| xsl.xml.recursive.[transformation\_name] | This property tells the system if the transformations have to run recursively inside the directory. If it is set to true, it takes all the subdirectories and the contained files but takes no file from the directories specified itself. If it is set to false takes only the files in the specified directory. If a fully recursive transformation is needed (also with file in the first folder) a recursive operation needs to be set=true and folderdepth=-1. | xsl.xml.recursive.codelistvalue=true |
| xsl.xml.folderdepth.[transformation\_name] | This property represents the level that a folder takes; example: if it is set to 2, then only the second level folder is taken | xsl.xml.folderdepth.codelistvalue=2 |

### Other custom transformation

In addition to the custom XML transformations, other format transformations can be configured.

The structure of the properties for the other custom transformations is similar to the ones described above.

There are 2 extra properties described here:

|  |  |
| --- | --- |
| Code | Description |
| transformation.xsl.startfrom.[transformation\_name] | This could be custom or standard and tells the system if the transformations have to start from the standard xml export or from the custom one. |
| transformation.xsl.outfileextension.[transformation\_name] | the file extension |

## Web Service

The Re3gistry has two way of accessing the contained resources: a RESTful web service which adopts a content-negotiation approach for serving the resource and a usual web service, that uses the resource name in order to access the specific resources. These are both described below.

### RESTful web service with content-negotiation

This web service is implemented using the Apache HTTPD web server and its content-negotiation capabilities.

The resource is accessed using the classic RESTful notation. Below you can find some examples:

|  |  |
| --- | --- |
| URL | Use |
| http://inspire.ec.europa.eu/[register LocalID]/[item LocalID]/[item LocalID] | - |
| http://inspire.ec.europa.eu/themes | Used to access given INSPIRE theme’s register |
| http://inspire.ec.europa.eu/Applicationschema/ad | Used to access a specific applicationschema |
| http://inspire.ec.europa.eu/codelist/AgeBy5YearsValue/41039 | Used to access a specific value in the code list |

To ask for a specific file format or language the HTTP request should have the proper http header set. If no header is set, the web service returns (as default) the xml format in English.

Below you can find some example:

Accept application/xml

Accept-Language en

Accept application/atom

Accept-Language fr

In order to enable the content-negotiation capabilities, the Apache HTTPD server has to be configured. Each content folder needs a configuration file that address the HTTPD to serve the right file; this files is a .var file (a no name file with the .var extension). The export system has an automatic procedure to produce this file after the execution of the export.

An example of the .var file is given below:

URI: ICDValue.en.atom

Content-language: en, en-GB, en-US, en-EN

Content-type: application/atom+xml

URI: ICDValue.en.html

Content-language: en,en-GB,en-US,en-AU,en-NZ; q=1.0

Content-type: text/html

URI: ICDValue.en.json

Content-language: en, en-GB, en-US, en-EN

Content-type: application/json

URI: ICDValue.en.xml

Content-language: en, en-GB, en-US, en-EN

Content-type: application/xml

For detailed information on how to configure Apache HTTPD and content-negotiation, refer to this url: <http://httpd.apache.org/docs/2.2/content-negotiation.html>

### Standard web service

To access the resources without using the content negotiation approach, it can be used the standard url-based request.

Below there are some examples of requests in different formats and languages:

<http://inspire.ec.europa.eu/codelist.en.html>

<http://inspire.ec.europa.eu/themes.de.atom>

<http://inspire.ec.europa.eu/Applicationschema/ad.fr.json>

<http://inspire.ec.europa.eu/codelist/AgeBy5YearsValue/41039.it.xml>

## Technical description

The following information describes the system’s architecture.

### Technology

The system is implemented using **Java 1.6** technology. The libraries used in the system are described in the POM (Project Object Model) file. This project is built using **Apache Maven** technology.

### Web Server

There are two different web servers: one used only to serve the RESTful web service, the other handles the system import and export. The two servers are listed below:

* **Apache HTTPD Version 2**
* **Apache Tomcat Version 6**

### Database

The database layer is handled by an EclipseLink library. All the databases supported by EclipseLink could be used for the system.

However, only the following database type/version has been tested for the current version of the system: **PostgreSQL 9.2**

## System installation and first run

The following six steps detail how to install and run the Re3gistry for the first time.

*Step 1*

Install the required software: Apache HTTPD, Apache Tomcat, Java 1.6, PostgreSQL 9.2.

*Step 2*

Execute the database initialisation using the SQL scripts available in the project’s package. (The create-table.sql must be run first, then the database-initialization.sql).

The database-initialization.sql file contains some sample data. For each new register to be added, the register table, the itemcalss table and the localisation table should all be filled in.

*Step 3*

Deploy the application “ARE3RegistryExport” and “ARE3RegistryImport” under the Tomcat instance.

*Step 4*

Run the data import using your csv file (the format of the file are specified in Section 5).

*Step 5*

Run the data export. The first export is the generic database export. After this export has been performed, the custom export can be started. The custom export has to be configured following the details in Section 6.2.3.

The last action to perform (if the content-negotiation feature under Apache is needed) is to start the generation of the .var files.

*Step 6*

Move the exported files under the Apache HTTPD document root (or under a specific, previously configured, folder).

If you want to use content-negotiation you have to configure the Apache HTTPD following the instruction in Section 7.2.

# Annex A

This information model which is also provided in the project package as an XSD and as an Enterprise Architect file.



1. See <http://ec.europa.eu/isa> [↑](#footnote-ref-1)
2. <http://ec.europa.eu/environment/water/water-framework> [↑](#footnote-ref-2)
3. <http://ec.europa.eu/environment/seis/> [↑](#footnote-ref-3)
4. For the detailed information model, please refer to annex A. The complete information schema could be found also as .eap (Enterprise Architect file) and .xsd (XML Schema Definition) in the project’s package. [↑](#footnote-ref-4)
5. Note: This system has no update procedure: if data need to be re-imported, then the original item must be deleted from the database. [↑](#footnote-ref-5)
6. Note: Each time the tables in the database is truncated (for example to re-import some data) this script will be re-executed in order to reinitialise the system. [↑](#footnote-ref-6)