***INSPIRE - Infrastructure for Spatial Information in Europe***

**Re3gistry software v0.4**

|  |  |
| --- | --- |
| **Title** | Re3gistry software v0.4 |
| **Creator** | European Commission, Joint Research Centre |
| **Creation Date** | 2014-09-29 |
| **Subject** | Re3gistry software |
| **Status** | Version 0.4 |
| **Publisher** | European Commission |
| **Type** | Text |
| **Description** | This document describes the Re3gistry software’s details and installation instruction |
| **Format** | MS Word (docx) |
| **Source** | Existing Re3gistry software documentation 0.3 |
| **Rights** | Reuse is authorised, provided the source is acknowledged. The reuse policy of the European Commission is implemented by a Commission [Decision of 12 December 2011 on the reuse of commisstion documents (2011/833/EU)](http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2011:330:0039:0042:EN:PDF) |
| **Identifier** | Re3gistry-v0.4 |
| **Language** | EN |

Disclaimer

The software is provided "as is", without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement. In no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the software or the use or other dealings in the software.

Table of Contents

[1 Background 7](#_Toc399838910)

[2 Introduction 8](#_Toc399838911)

[3 Information Model 9](#_Toc399838912)

[3.1 Registry 9](#_Toc399838913)

[3.2 Register 9](#_Toc399838914)

[3.3 Item Class 10](#_Toc399838915)

[3.4 Item 10](#_Toc399838916)

[3.5 Status 10](#_Toc399838917)

[3.6 Standard and Custom Attribute Fields 11](#_Toc399838918)

[3.7 Language representations 11](#_Toc399838919)

[4 System description 11](#_Toc399838920)

[4.1 Registry core 13](#_Toc399838921)

[4.2 Data module 13](#_Toc399838922)

[4.3 Data file structure 15](#_Toc399838923)

[4.4 Data actions and CSV formats 16](#_Toc399838924)

[4.4.1 Addition 16](#_Toc399838925)

[4.4.2 Clarification 18](#_Toc399838926)

[4.4.3 Supersession 20](#_Toc399838927)

[4.4.4 Invalidation 21](#_Toc399838928)

[4.4.5 Retirement 22](#_Toc399838929)

[4.5 Data analyser 23](#_Toc399838930)

[4.6 Data store 26](#_Toc399838931)

[4.7 Staticization 26](#_Toc399838932)

[4.8 Transformation configurations 32](#_Toc399838933)

[4.9 Specific XML transformation 32](#_Toc399838934)

[5 Serving the registry contents 33](#_Toc399838935)

[5.1 Web Service 33](#_Toc399838936)

[5.2 RESTful web service with content-negotiation 34](#_Toc399838937)

[5.3 Standard web service 35](#_Toc399838938)

[5.4 Customizing the web pages. 36](#_Toc399838939)

[6 Technical description 36](#_Toc399838940)

[6.1 Technology 36](#_Toc399838941)

[6.1.1 Web Server 36](#_Toc399838942)

[6.1.2 Database 36](#_Toc399838943)

[6.2 System structure 37](#_Toc399838944)

[6.2.1 Module concept 37](#_Toc399838945)

[6.2.2 Re3gistryCommon module 37](#_Toc399838946)

[6.2.3 Re3gistry webapp 38](#_Toc399838947)

[7 System installation and first run 41](#_Toc399838948)

[7.1 Common installation steps 41](#_Toc399838949)

[7.2 Using the precompiled webapp 42](#_Toc399838950)

[7.3 Using the source code 43](#_Toc399838951)

[7.4 Initialize the database with your data 47](#_Toc399838952)

[7.5 Running the Re3gistry software 47](#_Toc399838953)

[7.6 Setting up the web service 48](#_Toc399838954)

[8 Creating new modules 48](#_Toc399838955)

[9 Annex A 59](#_Toc399838956)

Table of tables

[Table 1 - CSV fields for the addition data file 17](#_Toc399838957)

[Table 2 - CSV fields for the clarification data file 19](#_Toc399838958)

[Table 3 - CSV fields for the supersession data file 20](#_Toc399838959)

[Table 4 - CSV fields for the invalidation data file 21](#_Toc399838960)

[Table 5 - CSV fields for the retirement data file 22](#_Toc399838961)

[Table 6 – Analyzer check and report message type 23](#_Toc399838962)

[Table 7 - RESTful URL example 34](#_Toc399838963)

Table of figures

[Figure 1 - System model 9](#_Toc399838964)

[Figure 2 - Schematic system description 12](#_Toc399838965)

[Figure 3- System description 12](#_Toc399838966)

[Figure 4- Data module, file selection screenshot 14](#_Toc399838967)

[Figure 5 - Data module, warning list 25](#_Toc399838968)

[Figure 6 - Data module, error list 25](#_Toc399838969)

[Figure 7- Staticization module – last staticized data 27](#_Toc399838970)

[Figure 8 - Staticization module - checking if a new data import has been added 27](#_Toc399838971)

[Figure 9- File system versioning, on the left the CodeList “ServiceTypeValue”, on the right the item CodeListValue “outpatientService” 29](#_Toc399838972)

[Figure 10 - XSL folder structure 30](#_Toc399838973)

[Figure 11- XSLT trasformations files 30](#_Toc399838974)

[Figure 12 - Staticization module, list of staticizer with the status 32](#_Toc399838975)

[Figure 13 - A view of the Re3gistryCommon module from an IDE 38](#_Toc399838976)

[Figure 14 - A view of the Re3gistry webapp module from an IDE 40](#_Toc399838977)

[Figure 15 - Load projects with NetBeans IDE 44](#_Toc399838978)

[Figure 16 - Creating new project from the project browser 48](#_Toc399838979)

[Figure 17 - Project type selection 49](#_Toc399838980)

[Figure 18 - Module name 50](#_Toc399838981)

[Figure 19- Creation of the module's folder 52](#_Toc399838982)

[Figure 20 - The new folder in the Re3gistry webapp project 52](#_Toc399838983)

[Figure 21 - Module's properties file 53](#_Toc399838984)

[Figure 22 - Localization folder for the new module 54](#_Toc399838985)

[Figure 23 - Localization properties folder 55](#_Toc399838986)

[Figure 24 - Module's localization properties file 56](#_Toc399838987)

Table of example

[Exmaple 1- ZIP data file 15](#_Toc399838988)

[Exmaple 2 - Mandatory field property 16](#_Toc399838989)

[Exmaple 3 - Addition CSV example 18](#_Toc399838990)

[Exmaple 4 - Clarification CSV example 19](#_Toc399838991)

[Exmaple 5 - Supersession CSV example 21](#_Toc399838992)

[Exmaple 6 - Example 6 - Supersession CSV example 22](#_Toc399838993)

[Exmaple 7 - Retirement CSV example 23](#_Toc399838994)

[Exmaple 8 – Staticizer configurations 29](#_Toc399838995)

[Exmaple 9 - Xslt transformation for xml 30](#_Toc399838996)

[Exmaple 10 - Gui-languages folder, French file structure 31](#_Toc399838997)

[Exmaple 11 – Staticizer configurations 33](#_Toc399838998)

[Exmaple 12 - Example of Content Negotiation Parameters 34](#_Toc399838999)

[Exmaple 13 - .var file example 35](#_Toc399839000)

[Exmaple 14 - Direct URL example 35](#_Toc399839001)

[Exmaple 15 - Example POM file 46](#_Toc399839002)

[Exmaple 16 - POM edit example 50](#_Toc399839003)

[Exmaple 17 – Constants.java example 51](#_Toc399839004)

[Exmaple 18 – Logger configurations 57](#_Toc399839005)

Acknowledgements

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.

The work on the Re3gistry addresses 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.

This software has been developed by the ISA founded developers Daniele Francioli and Emanuela Epure.

We are also grateful for the review of this document to Robin S. Smith.



<http://ec.europa.eu/isa/actions/01-trusted-information-exchange/1-17action_en.htm>

C:\Users\epuread\Desktop\arena.png

<https://joinup.ec.europa.eu/community/are3na/description>

Re3gistry software: <https://joinup.ec.europa.eu/software/re3gistry/home>

INSPIRE registry service: <http://inspire.ec.europa.eu/registry/>

To provide feedback on the software/service we kindly invite you to use this service:

<https://joinup.ec.europa.eu/software/re3gistry/home>

**Contact information**

Michael Lutz

European Commission – Joint Research Centre

Institute for Environment and Sustainability

Spatial Data Infrastructures Unit

E-mail: inspire-registry-dev@jrc.ec.europa.eu

<http://ies.jrc.ec.europa.eu>

<http://www.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). Data module and Data Staticization (Section 4) 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).

Changes in this version

This is the fourth release of the Re3gistry software. Changes:

* Adoption of new module-based development allowing easier extension/customisation;
* Incremental data import added (supporting new data import, saving and editing previous imports);
* ISO 19135-1 functionalities added: addition, clarification, supersession, invalidation and retirement;
* Simplified data files (no more xml descriptor);
* Complete data consistency check included in the data procedure;
* Multilingual support allows others to reuse the software in their own language;
* New edits/changes can be exported separately (using an incremental static-file approach);
* Simpler customisation of the exported file using the XSLT files);
* (Bug fixing)

# 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.

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 outputs of this work will also appear on the ISA programme collaborative platform, JoinUp, to aid wide re-use.

# Introduction

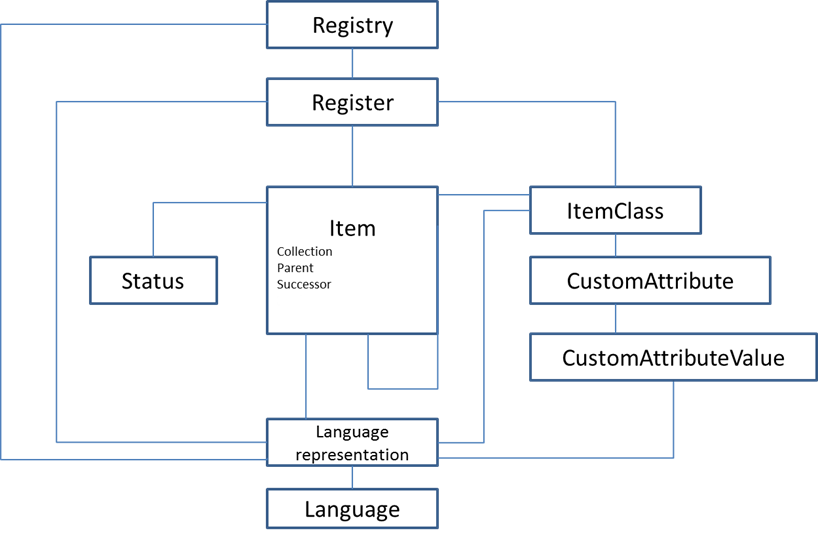
The Re3gistry software is designed to manage registers of items in different languages. The Re3gistry is a system to import the registry data, organize and manage them, and to export these data in different formats (such as html, xml, json). The exported files could be used for example to set up a web service in order to provide the access to the registers’ items.

The following documentation explains the details related to this software, including the install instructions and the user guide. There is also a guide to extend the software by creating new modules.

# Information Model

A simple representation of the system’s information model is shown below (fig. 1), with the following paragraphs describing each entity that can be found in the system[[4]](#footnote-4).

Figure - System model



## Registry

A registry is an information system on which a register is maintained.

## Register

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

The type of item 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 items 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 class can have parent/child relations to identify if the items are part of a collection. For example, in the INSPIRE Registry, the “codelist register” has two item class: CodeList and CodeValue. The CodeList item class is the father of CodeValue. The CodeValue item class contains the items that are part of the “Code List” collection.

## Item

Items are elements that can be entered in a register. An item can contain other child items. For example, in the INSPIRE registry, the item class “CodeList” could contain other items (items with item class “CodeValue”). The contained item (the “codevalue item”), has an indication of the collection within the collection\_id field (for detailed information please see the complete information model in [Annex A](#_Annex_A)).

Another type of relation between items of the same item class is the parent/child relation. This kind of relation is provided by the parent\_id field in the CSV data file. This field represents the reference to the local id of the parent item.

Note: The parent item and the child item shall be in the same item class. If an item is part of a collection, it can have a parent contained in the same collection.

## Status

Each item has a status as defined in ISO 19135-1 draft:

* Valid: the item has been accepted, is recommended for use, and has not been superseded or retired.
* Invalid: a decision has been made that a previously valid register item contains a substantial error and is invalid, and will normally have been replaced by a corrected item.
* Submitted: the item has been entered into the register, but the control body has not accepted the proposal to add it.
* Superseded: the item has been superseded by another item and is no longer recommended for use.
* Retired: A decision has been made that the item is no longer recommended for use. It has not been superseded by another item.

## Standard and Custom Attribute Fields

Each item has a standard list of fields common to all items. “Custom attribute” fields are designed to extend the standard item fields. By using a custom attribute, the item can be extended to include any item-specific fields. The current standard list of fields is described at [paragraph 4.4.1](#_Addition), Table 1.

## Language representations

Each component in the system (except for the status) can be represented in several languages. The localisation table in the database contains all the languages related to the various elements in the system and currently covers 23 official languages of the European Union.

# System description

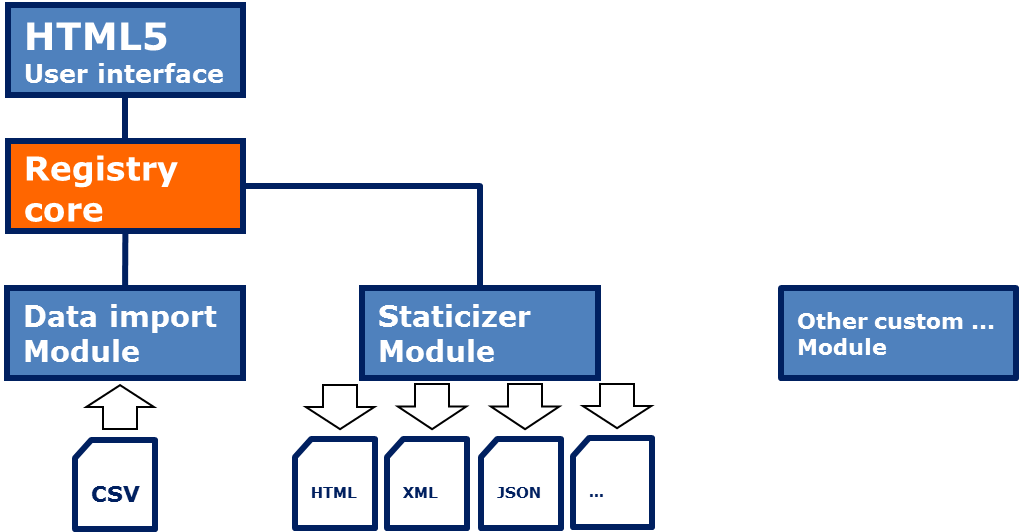
The Re3gistry software handles the entire data flow process for register items management, from data import and edit to the staticization of specific items in different customizable formats.

The system is composed by modules: there is a common module, implementing all the basic features, and several other modules implementing each specific functionality.

This package is bundled with the data management module (responsible for importing and editing data) and the data staticization module (responsible for saving the data as static files in different formats).

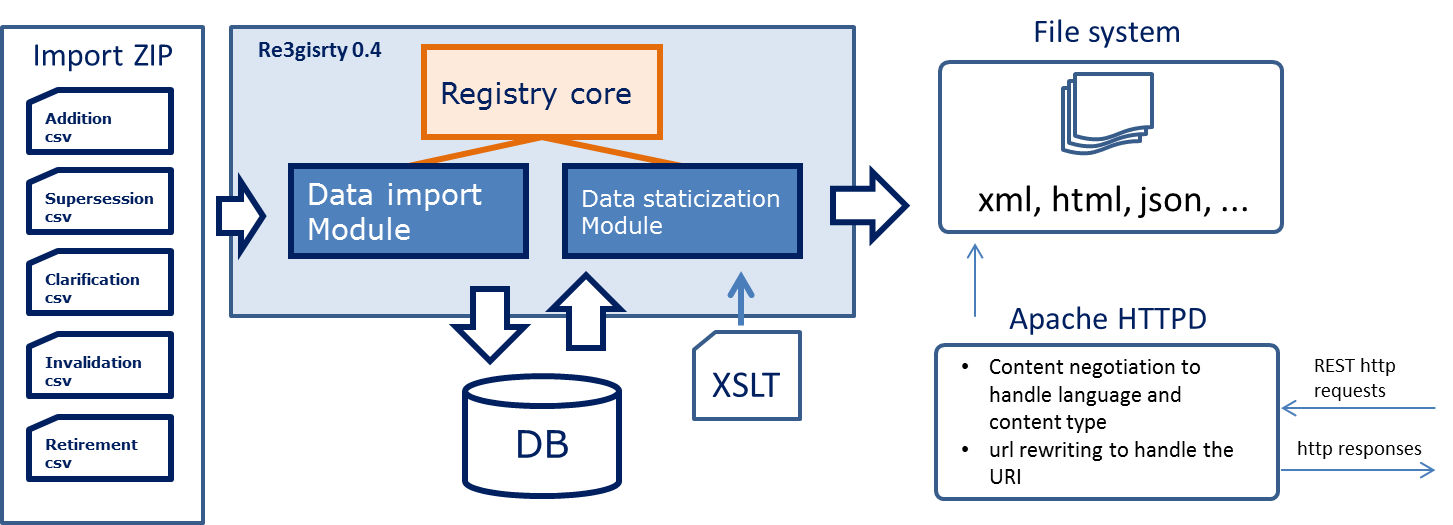
The picture below (fig. 2) shows a schematic representation of the system.

Figure - Schematic system description



The schema below gives a complete overview of the Re3gistry software and all the other components required in order to provide a RESTful web service (fig. 3).

Figure - System description



The following paragraphs give an overview of the system. A detailed description of each module and how to extend or create a new module is provided at [chapter 6](#_Technical_description) and [chapter 8](#_Creating_new_modules).

## Registry core

The registry core module contains all the basic object model and database operation required by the system.

## Data module

The data module is responsible for all the data operation that can be performed on the registers.

The input for this module is the zip data file (described at [paragraph 4.3](#_Data_file_structure)), containing one CSV file for each of the operations, for each register.

The supported operations are taken from the ISO19135-1[[5]](#footnote-5) and are listed below (detailed at [paragraph 3.5](#_Status)):

* Addition
* Clarification
* Supersession
* Retirement
* Invalidation

The CSV file description for each of these actions is provided at [paragraph 4.4](#_Data_actions).

The data module performs different check to ensure the data consistency in the register.

The data module procedure steps are listed below:

* **Format check**: a check on the csv files format (described at [paragraph 4.4](#_Data_actions));
* **Data analysis**: a semantic check trough the data, by verifying links between data and data consistency (described at [paragraph 4.5](#_Data_analyzer))
* **Data store**: save the data contained in the CSV files (described at [paragraph 4.6](#_Data_store)).

The Re3gistry’s web interface used to start the data procedure is very simple. It requires choosing the *data file*, a *short description* for the procedure and the optional “*Ignore warning*” flag.

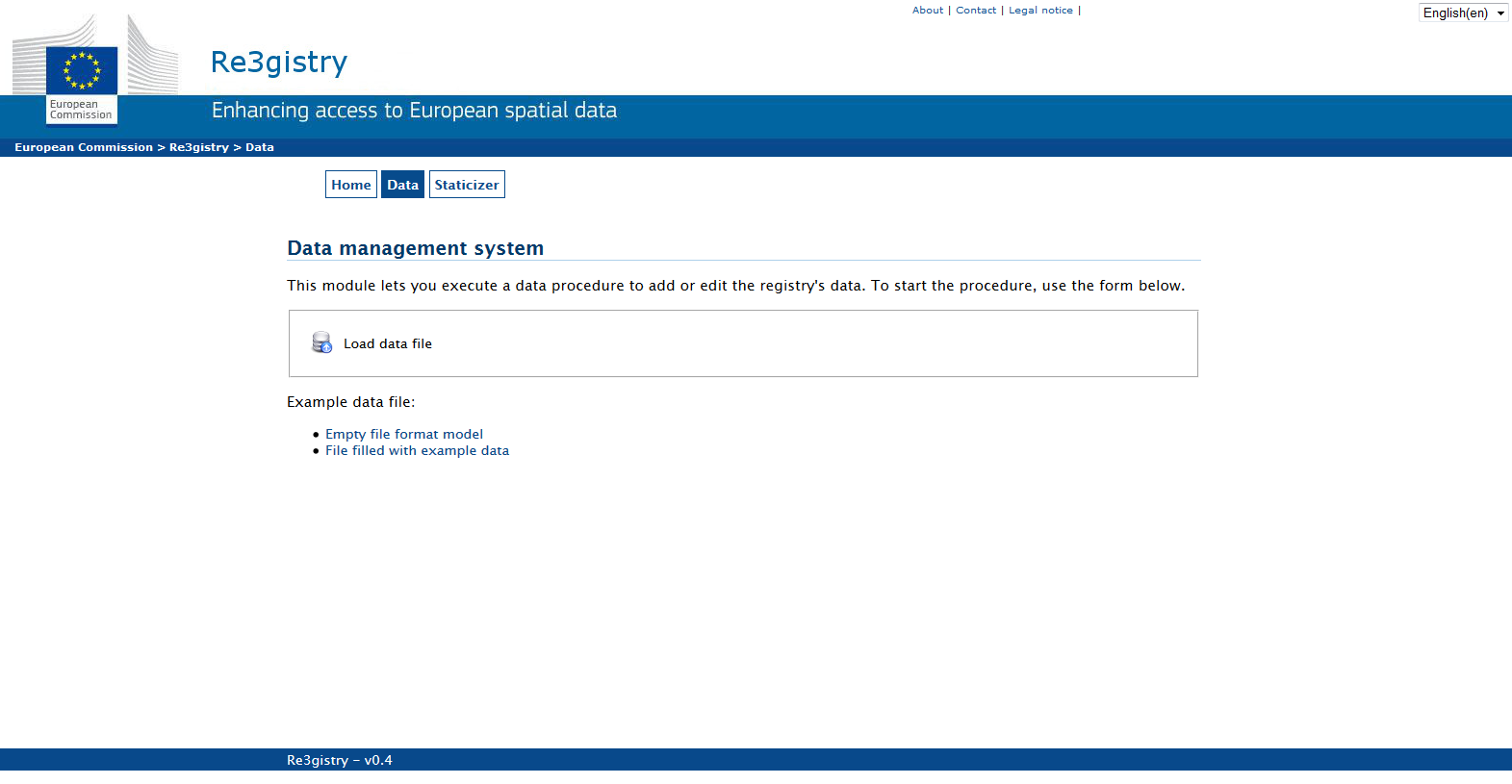
The procedure’s description field is mandatory and it is useful to later identify the procedure in the list of the procedure completed.

The “Ignore warnings” checkbox force the system to perform the data store even if the analyzer finds any warnings in the data files. If the analyzer finds some errors, the procedure is locked even with the “force warnings” flag set. The difference between errors and warnings is explained at [paragraph 4.5](#_Data_analyzer).

After filling all the required information, click on the “Start procedure button”. The procedure may take a considerable amount of time, depending on the number of items to be processed.

The browser can be closed the system will continue to run; an email notification will be sent once the procedure ends. If it ends with some problems, the list of problems will be attached to the email.

Figure - Data module, file selection screenshot



In the data module interface there are also 2 example data files: one is an empty file containing only the structure for each register, the other file contains some sample INSPIRE Registry data. They could be a starting point for testing the system and getting familiar with it.

An important concept is the “**operating language**”: this property identifies the language representation that shall be always available. You cannot load an item without this language representation.

For example, if the operating language is set to English, and a new item has to be added in the French language, the addition file shall also contain the item in the English language.

The operating language can be configured in the system following the documentation at [chapter 7](#_System_installation_and).

## Data file structure

Since the first versions of the Re3gistry software, the data file had a XML file descriptor. Now, the system can understand all the information needed in order to perform the actions by analysing the ZIP folders and files.

The zip file structure is available below:

* The root of the ZIP file contains one folder for each of the *item class* (for the definition of item class, refer to [paragraph 3.3](#_Item_Class)); on each of these folders, the system is going to perform an action. The name of the folder has to be exactly the same name of the item class, including the case (the reference to the register is taken through the item class, which is linked to the register in the information model – [chapter 3](#_Information_Model)).
* Each of these folders contains the CSV files related to the action to be performed on the specific item class. The file is named with the action’s name in lowercase.

An example ZIP structure:

Exmaple 1- ZIP data file

**import.zip**

ApplicationSchema/addition.csv ApplicationSchema/clarification.csv Theme/supersession.csv

Theme/retirement.csv

CodeList/invalidation.csv

The ZIP data files can contain only the files and folders needed to perform the related actions.

For example, the ZIP data file provided above (example 1) will perform:

* addition(s) and clarification(s) for the ApplicationSchema item class;
* supersession(s) and retirement(s) for the Theme item class.

## Data actions and CSV formats

For each of the actions supported by the system there is a specific CSV format for the data file, detailed below.

### Addition

This action adds the items specified in the related register.

The CSV for this action is composed by two different types of fields: *main fields* and *additional fields*.

The main fields must be provided even if they have null values; if one or more of the main field values are not available, the specific field must be left blank (the csv separator without any character or space. Example: value1|||value2|value3|||value4||…).

There are some main fields that are required for each action. These fields are highlighted in table 1 and can be configured in the data module properties file.

The property to customize the required fields is: data.mantatoryfields.<action\_name>=<field>.<field>. …

The example below shows an example of mandatory fields for the addition action.

Exmaple - Mandatory field property

**Data file header:**

**LocalId**|ParentLocalId|CollectionLocalId|**Language**|**Label**|Definition|Description|**Status**|Comment|\*Theme[t,f,f,t]|\*UMLName[t,f,f,f]

…

**Property in the properties file:**

**…**

data.mantatoryfields.addition=0.3.4.7

List the index of the field as in the property above to set a common field mandatory.

The additional fields (called *custom attributes*) represent some extra fields that users can add in order to customize the data model. The properties of these fields are described in the table 1.

Table - CSV fields for the addition data file

|  |  |  |
| --- | --- | --- |
| Type | Code | Description |
| **Main fields** | **LocalId (mandatory)** | The local identifier (id) is used to identify an item inside the register and to create the URI of the item. If an item has a local identifier such as “localId”, the URI will be composed as follow: “http://site.ext/register/localId”; |
| 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. |
| **Language (mandatory)** | The language of this item. If there is multiple language for each item, the row has to be duplicated, hanging the language related fields. |
| **Label (mandatory)** | This contains the label of the items |
| Definition | This contains the definition of the system |
| Description | This contains the description of the items |
| **Status (mandatory)** | This contains the status related to the current item. More details about the code to be used in this field can be found in Table 2. |
| Comment | A textual comment for the change log stored by the system for each action. |
| **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:  **\*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].**  The meaning for each of the properties set in the square parenthesis is described in the following rows. |
| Required | This indicates that the custom attributes is required. The whole data rows in the CSV must have this value in order to be imported. |
| Multivalue | This indicates that this custom attribute could have more than one value. |
| Coded | If this property is set to true, the custom attribute takes its values from a list of values defined in a specific database table. More details about the information model can be found in [Annex A](#_Annex_A) (This feature is not implemented but is foreseen in the future). |
| Foreignkey | If this property is set to true, the value in this field is a *localId* of another item in the register. The name of the custom attribute has to be the name of the item class referenced. |

Exmaple - Addition CSV example

**Import.zip -> ApplicationSchema/addition.csv**

LocalId|ParentLocalId|CollectionLocalId|Language|Label|Definition|Description|Status|Comment|\*Theme[t,f,f,t]|\*UMLname[t,f,f,f]

sd|||en|Species Distribution|||valid||sd|SpeciesDistribution

sd|||fr|Répartition Des Espèces|||valid||sd|SpeciesDistribution

This example contains an addition for the ApplicationSchema item class (linked to the ApplicationSchema register). There is the mandatory header line and one item in two languages.

Note: in the example above the custom attribute (name: Theme) is defined as foreign key. In this case, the name of the custom attribute (in the descriptor line –the first line of the csv) shall be exactly the name of the itemclass related to the item specified as value. The value of the custom attribute shall be the LocalId of the referenced element. Only items that are not part of a collection (with the CollectionId field empty) can be pointed by a custom attribute.

### Clarification

This action allows editing the items. Every time a clarification action is performed, the version of the edited item is increased.

This action data file shall be filled only with the fields that need to be edited (plus the LocalId and the language to identify the item). Only the filled fields will be updated; the field left blank in the action file, will not be updated, and remain as is in the register.

The clarification action cannot change the collection of an item. Nevertheless the clarification CSV file contains the *CollectionLocalId* field in order to unambiguously identify the item. In fact, in the same data file, different items that are part of a collection may have the same *LocalId* but different CollectionLocalId. In the clarification (but also in the supersession, retirement and invalidation) the item has to be identified in the data file by its *LocalId* and *CollectionLocalId* (if available).

The clarification’s CSV file is like the addition CSV file but without the status field. The CSV format is described in table2 (for the detailed fields description, refer to [table 1](#_Addition)).

Table - CSV fields for the clarification data file

|  |  |  |
| --- | --- | --- |
| Type | Code | Description |
| **Main fields** | **LocalId (mandatory)** | The local identifier (id) is used to identify the item to be updated |
| ParentLocalId |  |
| CollectionLocalId |  |
| **Language (mandatory)** | The local identifier (id) is used to identify the language representation of item to be updated |
| Label |  |
| Definition |  |
| Description |  |
| Status |  |
| Comment |  |
| **Additional fields** | (custom attribute/ header descriptor) | In the clarification data file, all the custom attribute related to the current item class shall be specified in the first row as the addition CSV. |

Exmaple - Clarification CSV example

**Import.zip -> ApplicationSchema/clarification.csv**

LocalId|ParentLocalId|CollectionLocalId|Language|Label|Definition|Description|Comment|\*Theme[t,f,f,t]|\*UMLname[t,f,f,f]

sd|||en|Species Distribution changed|||||changed custom attribute

sd|||fr|| Ajouter la définition en français ||||

This example contains a clarification CVS file for the ApplicationSchema item class (linked to the ApplicationSchema register). The file contains only the “sd”. Item, to be modified in the 2 different language:

* English: the label and the “UMLname” custom attribute are edited;
* French: the definition is added

### Supersession

This action allows superseding an item. The supersession CSV file format is quite different from the previous two, because it only requires the reference to the item to be superseded and to the successor item.

The successor item shall be available in the database or shall be present in the addition csv of same item class data file containing the supersession csv.

Table - CSV fields for the supersession data file

|  |  |
| --- | --- |
| Code | Description |
| **SupersededLocalId (mandatory)** | The local identifier (id) is used to identify the item to be superseded |
| SupersededCollectionLocalId | The optional collection local identifier (id) is used to identify the item to be superseded (to be provided if the item is part of a collection) |
| **NewLocalId (mandatory)** | The local identifier (id) is used to identify the successor item |
| NewCollectionLocalId | The optional collection local identifier (id) is used to identify the successor item (to be provided if the item is part of a collection) |
| Comment | A textual comment for the change log stored by the system for each action. |

Exmaple - Supersession CSV example

**Import.zip -> ApplicationSchema/supersession.csv**

SupersededLocalId|SupersededCollectionLocalId|NewLocalId|NewCollectionLocalId|Comment

sd||ad||this is a comment

This example contains a supersession CVS file for the ApplicationSchema item class (linked to the ApplicationSchema register). The file contains only the “sd” item, to be superseded by the “ad” item. This supersession data file contains also a comment.

### Invalidation

This action allows invalidating an item. The invalidation CSV file format is the same such as the supersession CSV file, in addition there is the recursive flag (explained in the table 4), because it only requires the reference to the item to be superseded and to the successor item.

The successor item shall be available in the database or shall be present in the addition csv of same item class data file containing the supersession csv.

Table - CSV fields for the invalidation data file

|  |  |
| --- | --- |
| Code | Description |
| **LocalId (mandatory)** | The local identifier (id) is used to identify the item to be invalidated |
| CollectionLocalId | The optional collection local identifier (id) is used to identify the item to be invalidated (to be provided if the item is part of a collection) |
| **SuccessorLocalId (mandatory)** | The local identifier (id) is used to identify the successor item |
| SuccessorCollectionLocalId | The optional collection local identifier (id) is used to identify the successor item (to be provided if the item is part of a collection) |
| Recursive | This field tell the system to recursively invalidate any children items or linked items to the invalidated item. To set this flag put the word “true” in the field, otherwise leave it blank. |
| Comment | A textual comment for the change log stored by the system for each action. |

Exmaple - Example 6 - Supersession CSV example

**Import.zip -> ApplicationSchema/invalidation.csv**

LocalId|CollectionLocalId|SuccessorLocalId|SuccessorCollectionLocalId|Recursive|Comment

sd||ad|||this is a comment

so||ad||true|this is a comment

This example contains an invalidation CVS file for the ApplicationSchema item class (linked to the ApplicationSchema register). The file contains two invalidations:

* the “sd” item, with the successor item “ad” item. This is not recursive and contains also a comment.
* the “so” item, with the successor item “ad” item. This is recursive and contains a comment.

### Retirement

This action allows retiring an item. The supersession CSV file format is the simpler one. It only requires the reference to the item to be retired plus a recursive flag (explained at table 5).

Table - CSV fields for the retirement data file

|  |  |
| --- | --- |
| Code | Description |
| **LocalId (mandatory)** | The local identifier (id) is used to identify the item to be retired |
| CollectionLocalId | The optional collection local identifier (id) is used to identify the item to be retired (to be provided if the item is part of a collection) |
| Recursive | This field tell the system to recursively retire any children items or linked items to the retired item. To set this flag put the word “true” in the field, otherwise leave it blank. |
| Comment | A textual comment for the change log stored by the system for each action. |

Exmaple - Retirement CSV example

**Import.zip -> ApplicationSchema/retirement.csv**

LocalId|CollectionLocalId|Recursive|Comment

sd|||this is a retirement

so||true|this is a retirement

This example contains a retirement CVS file for the ApplicationSchema item class (linked to the ApplicationSchema register). The file contains two item, to be retired:

* the “sd” item. This is not recursive and contains also a comment.
* the “so” item. This is recursive and contains a comment.

## Data analyser

The data analyser component perform some checks in order to understand if the data file contains consistent data and also if the database keep its consistency after current data file is processed.

There are two type of message raised by the analyser:

* Errors: these are locking problems; the data file shall be corrected before the system can continue its work.
* Warnings: these are non-locking problems; the user is notified about the problem but the import can continue by setting the “force warning” flag available in the Data Module web interface.

Below there is a list of the checks performed by the analyser.

Table – Analyzer check and report message type

Legend: errors - warnings

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Actions** | | | | |
| **Situations** | **Addition** | **Clarification** | **Supersession** | **Invalidation** | **Retirement** |
| Item already in the database |  | - | - | - | - |
| Item not available in the database | - |  |  |  |  |
| Custom attribute (with foreign key flag true) pointing to an item not available in the database nor in the current addition file |  |  | - | - | - |
| Custom attribute (with foreign key flag true) pointing to an item not valid |  |  | - | - | - |
| Item pointing to a parent not available in the database nor in the current addition file |  |  | - | - | - |
| Item pointing to a parent not valid |  |  | - | - | - |
| Item pointing to a collection not available in the database nor in the current addition file |  | - | - | - | - |
| Item pointing to a collection not valid |  | - | - | - | - |
| Item pointing to a successor not available in the database nor in the current addition file | - | - |  |  | - |
| Item pointing to a successor not valid | - | - |  |  | - |
| The item’s operating language is not available |  | - | - | - | - |
| The item’s language is not in the list of supported language |  | - | - | - | - |
| The item has collection | - | - |  | \* |  |
| The item has child | - | - |  |  |  |
| The item has links (pointed by a custom attribute as foreign key) | - | - |  | \* |  |

\* The system will throw an error only if the recursive flag has not been set.

If the system finds some errors or warnings, it stops the procedure and shows a list containing all the problems found, with a short description. Below there is a screenshot related to the two different types of problems (warnings and errors).

Figure - Data module, warning list

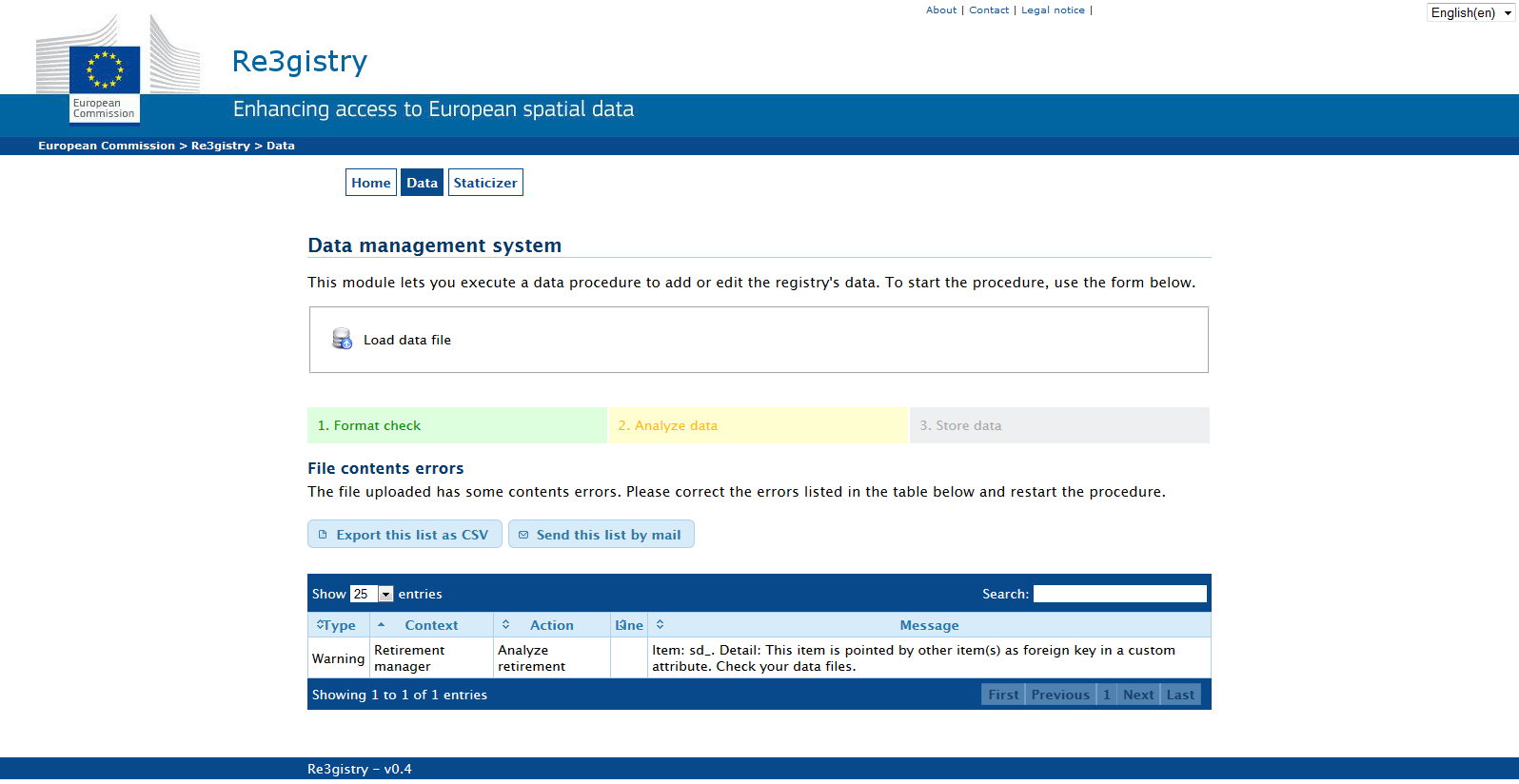
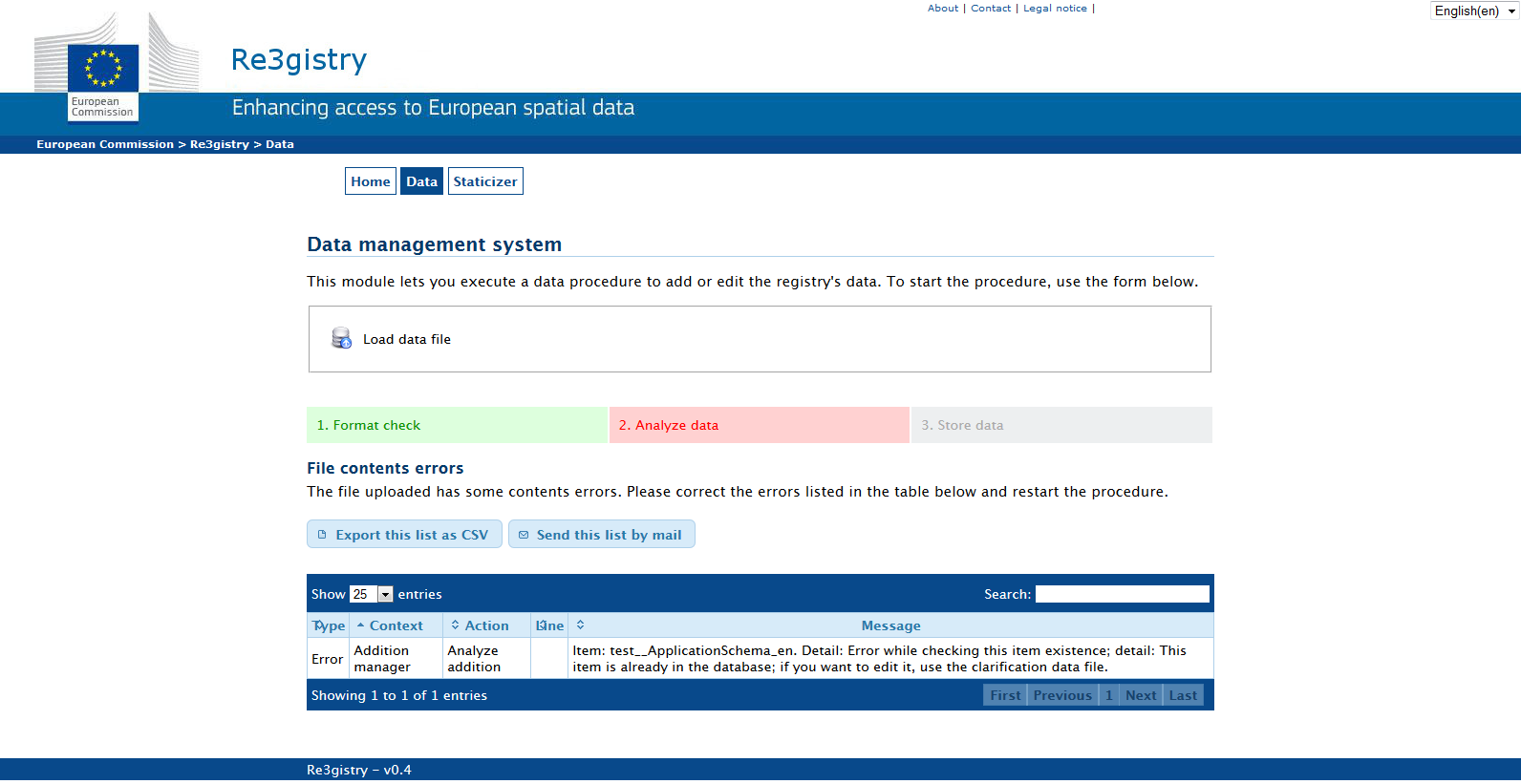


Figure - Data module, error list



In case the procedure produces both errors and warnings, the system will lock even if there is the “Force warning” flag set. In this case the web interface will present a mixed error and warning rows; the type of the problem is simply understandable by the specific “problem type” column.

In both case of errors or warnings, the system will send an email (to the configured admin email) containing an attachment with the list of errors.

When the procedure ends correctly, the admin user is notified by email.

## Data store

The data store component performs all the action described in each csv contained in the ZIP data file.

## Staticization

The “Staticization” component is responsible for saving the data stored by the “Data” module to the file system.

The Data “Staticization” procedure is described below:

### Analyzer

The analyzer procedure finds all the data modifications (performed by the Data module), that have not been staticized yet.

The first table (fig. 7) shows the components of staticizer that has been completed, are in progress/queued/failed.

The second table (fig. 8) lists all the data import completed since the last staticizer procedure has completed.

The third table (fig. 8) shows in detail, the item modified by the last data imports (listed in the second table).

Figure - Staticization module – last staticized data

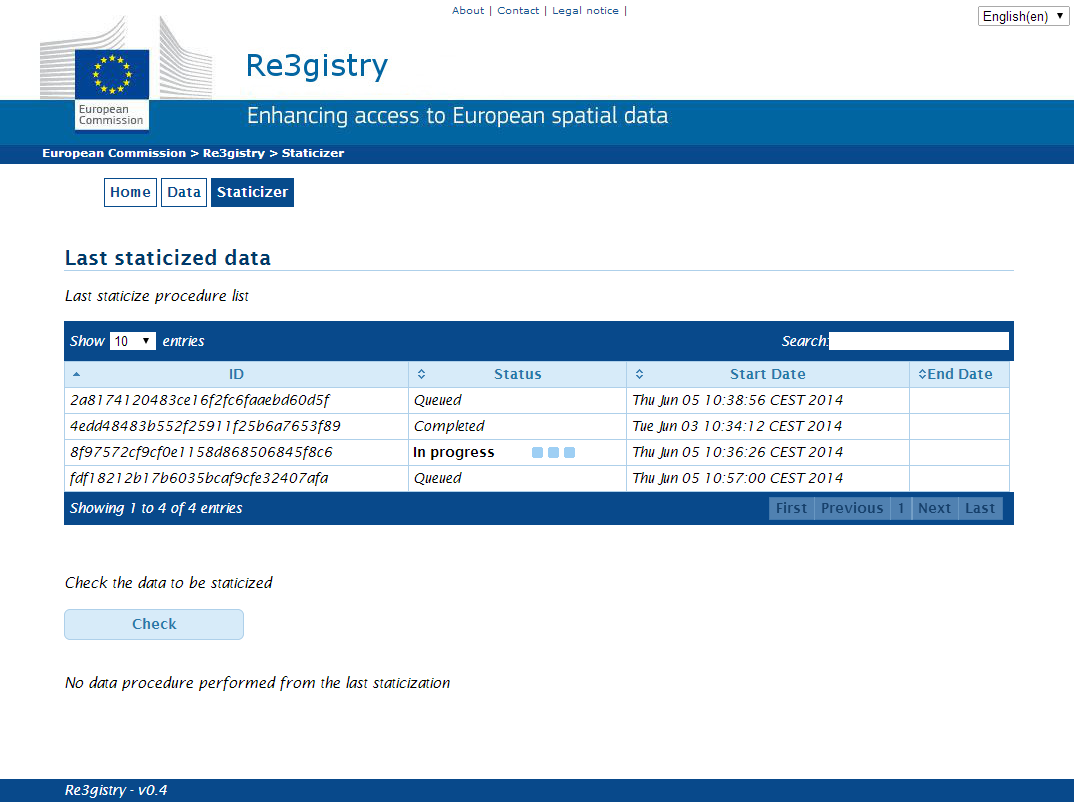
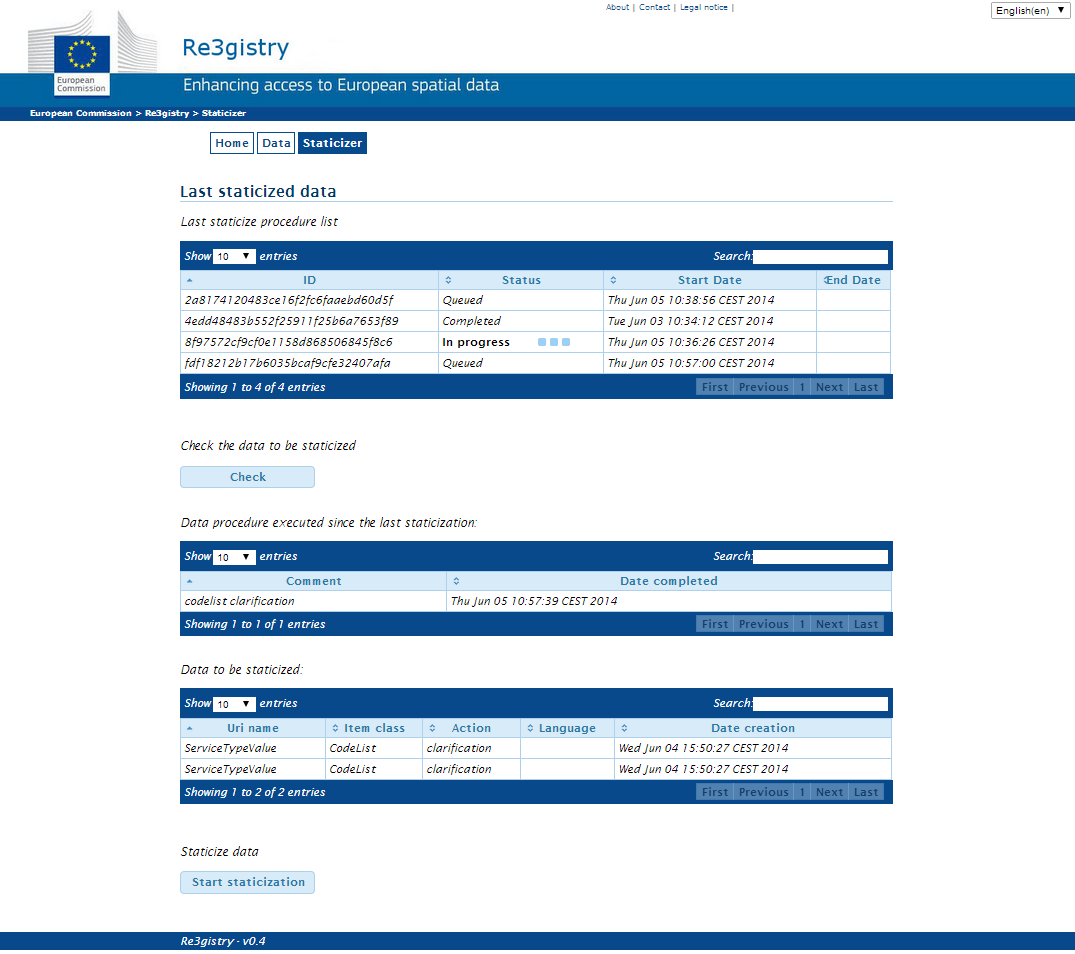


Figure - Staticization module - checking if a new data import has been added



### Store

The store procedure saves the items to the file system in the formats specified by the user. It runs the XSLT trasformations described in the following paragraphs.

Once the procedure has started, the system runs even if the browser is closed. The system will notify the user by email once the procedure ends. The user is notified even if the procedure has been queued (Only one staticization can run per time).

The procedure may take a considerable amount of time, depending on the number of items to be processed (for example, the complete INSPIRE registry data, in all the 23 languages, in the 4 formats provided, will take about 5 hours).

The store procedure could be spitted in different steps:

The first step, checks if there is another staticizer in progress. Only one staticizer can run per time. In case another staticization process is in progress, any new staticization requests will be queued.

Once the current staticization process has been completed, the next one (the oldest in the queue) is started.

The user will receive an email (to the configured admin email address) every time the procedure changes its status.

The staticization process takes all the items to be stored (all the items listed in table 3 – described at the previous paragraph), and starts the creation of the static files.

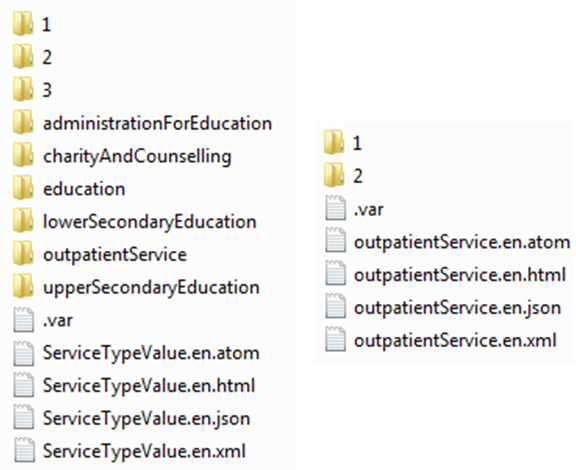
The system creates firstly a common xml file, containing all the information stored in the database (also called “standard export”).

Then using the XSLT files provided by the user, it creates the different file formats, starting from the standard xml produced. This export is also called “custom”.

The “custom” folder will contain all the files produced using the XSLT files.

The file system’s structure of the produced files, reflects the registry’s structure. The example below (fig. 9) shows this folder structure.

Figure - File system versioning, on the left the CodeList “ServiceTypeValue”, on the right the item CodeListValue “outpatientService”



The name of the folders related to each export can be customized in the Re3gistryStaticizer.properties file:

Exmaple – Staticizer configurations

staticizer.standard.folder=standard

staticizer.custom.folder=custom

staticizer.solr.folder=solr

staticizer.solrcustom.folder=solrcustom

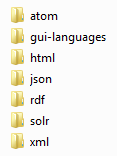
### XSLT

The XSLTs are responsible for the creation of the custom file formats defined by the user.

Each new format shall have a folder named with the same name of the format itself. For example, for the .html format, there is an html folder, containing all the XSLT required for this transformation.

An example related to the format used in the INSPIRE registry is provided below.

Figure - XSL folder structure



Each of the folders used for the XSLT transformation should contain one XSLT file for each registry, one for each register and one for each category of item:

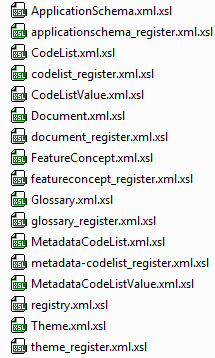
Exmaple - Xslt transformation for xml

<itemclass\_uriname>.<format>.xsl - items

<register\_uriname>.<format>.xsl - register

<registry\_uriname>.<format>.xsl - registry

Figure - XSLT trasformations files



Each format shall also be configured in the Re3gistryStaticizer.properties file. These configurations are described at [paragraph 4.9](#_Specific_XML_transformation).

The “*gui-languages*” folder contains *<language\_code>.xml* (one for each language supported by the system) files containing the keys used by the xslt transformation for any static string not available in the database (for example, string used in an html web page).

For example, the fr.xml gui translation file could be:

Exmaple - Gui-languages folder, French file structure

<guilanguage languagecode="fr">

<item key="feedback">Retour d'utilisation</item>

<item key="powered-by">Construit avec</item>

<item key="about">A propos de</item>

<item key="contact">Nous contacter</item>

. . . .

</guilanguage>

Usage example, in the transformations for the html, the values translated has been used

<xsl:value-of select="$languagefile/guilanguage/item[@key='feedback']"/>

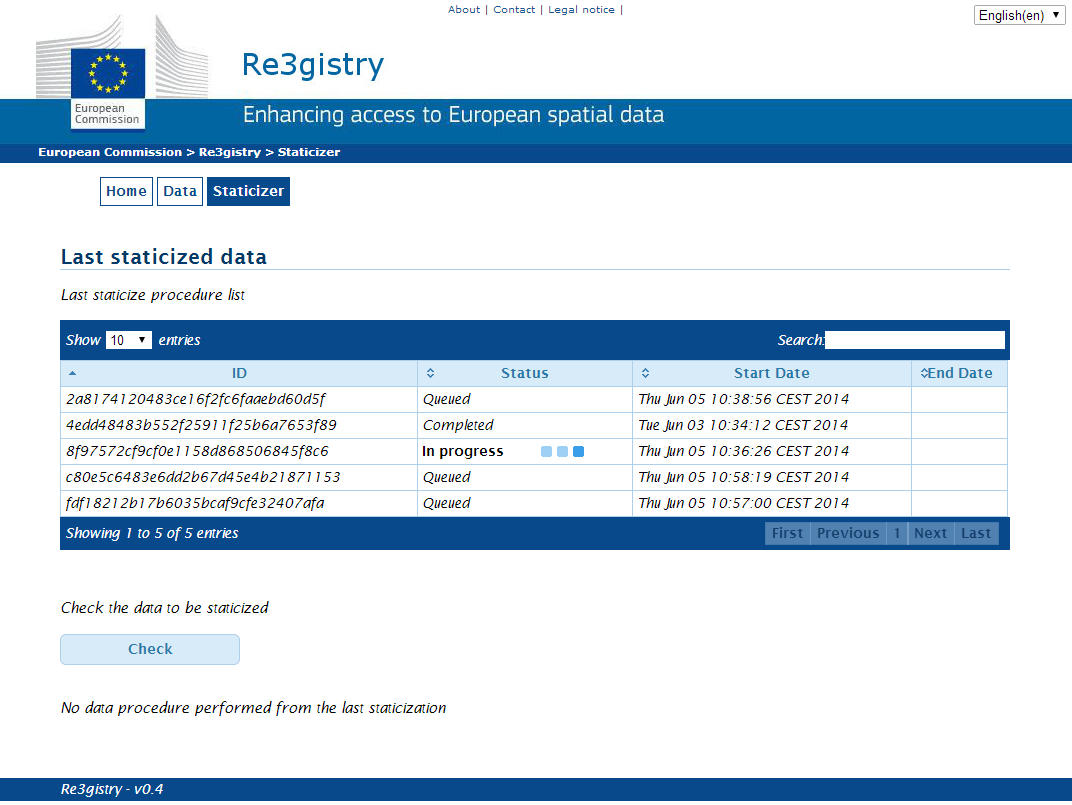
where the variable $languagefile is the path of the document containing the translations.

### Additional information

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 5.2.

The **Solr export** will produce a database representation in XML format (one file for each items, containing all the languages). The Solr XSLT transformation produces files compliant with the Solr indexing system.

Figure - Staticization module, list of staticizer with the status



## Transformation configurations

The transformation system allows the user to define different custom formats for the static file produced by the staticizer. This process uses XSLT files in order to convert the original xml files into other file formats.

## Specific XML transformation

Each transformations use as input the standard xml export (containing all the information stored in the database).

Each transformation defined by the user, shall have an entry in the transformation property. This property is set in the Re3gistryStaticizer.properties.

To configure each transformation defined by the user, the relative entry has to be listed in the xsl.formats.list property. Example:

Exmaple – Staticizer configurations

xsl.formats.list=xml,json,html,atom,rdf

This property tells the system which transformations are available and how to find the related properties.

The folder containing the XSLT for this format shall have the same name of the property defined above (example: /xsl/html).

# Serving the registry contents

The Re3gistry software (staticizer module) produces a set of files organized to set up a web service in order to provide access to the registers items 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 to accessing a specific file, the system supports this approach too.

The file to be used in order to set up a web service can be found in the Staticizer’s output folder. In detail, the files to be used are under the “custom” folder; place it in the web server document root.

Detailed information about the web service and its configuration are provided in the following paragraph.

## Web Service

The files produced by the Re3gistry software (staticizer module) can be used to provide a web service in two ways: a RESTful web service which adopts a content-negotiation approach for serving the resource and a web service, which 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 standard RESTful notation. Below there are some examples:

Table - RESTful URL example

|  |  |
| --- | --- |
| URL | Use |
| http://base\_uri/register\_uriname  http://base\_uri/item\_itemclass/item\_uriname  http://base\_uri/collection\_itemclass/  collection\_uriname/item\_uriname | Used to access a register, an item or an item collection |
| http://inspire.ec.europa.eu/themes | Used to access the INSPIRE theme’s register |
| http://inspire.ec.europa.eu/Applicationschema/ad | Used to access a specific application schema |
| http://inspire.ec.europa.eu/codelist/ ActiveWellTypeValue/decontamination/ | 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 (by default) the xml format in English.

Below there are some examples:

Exmaple - Example of Content Negotiation Parameters

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 addresses the HTTPD server to serve the right file. This file is a .var file (cn .var). The staticizer module has an automatic procedure to produce this file after the execution of the staticization.

An example of the cn.var file is given below

Exmaple - .var file example

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 can be used.

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

Exmaple - Direct URL example

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

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

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

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

http://inspire.ec.europa.eu/codelist/Article17CountingUnitValue/reference/Article17CountingUnitValue.fr.rdf

## Customizing the web pages.

There are two way of customizing the look and feel of the web pages produced by the Staticizer.

* **Customize the CSS**: use this approach if the pages’ HTML structure and common content (like the footer information) doesn’t need any modification. In fact, only a few graphical aspect of the pages can be changed editing the CSS. The css related to the service are placed in the project package, folder example/web-service-files/res/css.
* **Customize the pages’ structure:** to change the HTML structure or some common contents (like the common text or common page part), edit the XSL transformation files placed in example/xsl

# Technical description

The following information describes the technical specification of the system.

## Technology

The system is implemented using **Java 1.7** technology. The libraries used in the system are described in the Project Object Model (POM) file. This project is built using **Apache Maven** technology.

### Web Server

There are two different web servers: one used to serve the RESTful web service and another is the Servlet container for the Re3gistry Java webapp. The two servers are listed below:

* **Apache HTTPD Version 2**
* **Apache Tomcat Version 7**

### Database

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

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

## System structure

The Re3gistry software is organized in modules: this concept allow a simple customization and extension of the system.

The main components of the system are the *Re3gistryCommon* module and the *Re3gistry* webapp.

The paragraphs below explain the main structure of the system and the modules. By understanding the system’s structure the modules can easily edited and extended.

### Module concept

The module is a Java library plus some web pages that adds functionality to the core system. Each module is composed by a core part (java library) and a presentation part (web pages).

* The module’s core part (java library) contains all the logic related to the functionality provided by the module itself. Each module needs to include the Re3gistryCore module in order to access all the common beans and functions.
* The module’s presentation part is contained in the module’s folder of Re3gistry webapp. This folder contains the user interface to interact with the function provided by the module, and the specific module’s configuration file.

The Re3gistry webapp contains also the main configuration properties files and all the localization properties files.

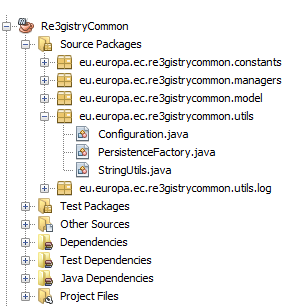
### Re3gistryCommon module

The Re3gistryCommon module is the base of all the software. It contains the object definition that represents each entity in the system and the manager to work with them (read/write). This core module has no a related presentation part in the Re3gistry webapp because its function is to provide the object structure and the main functionalities to all the other modules.

The Re3gistryCommon library is composed by different package described below:

* constants: contains a class with all the constant used in this library;
* managers: contains all the method to access the data for each bean defined in the model package;
* model: contains all the bean related to the information model;
* utils: contains several general utility classes, such as StringUtils or Logger class.

Figure - A view of the Re3gistryCommon module from an IDE



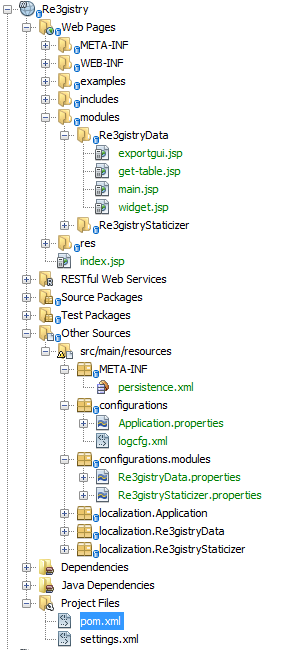
### Re3gistry webapp

The R3gistry webapp contains the web interface to manage and use the functionalities provided by each modules. In this webapp are stored also all the configuration files, both related to the core module and to each of the additional module.

Below there is a description of the components contained in the Re3gistry webapp.

* Web Pages folder: contains all the files related to the web pages (jsp, css, js, resources like images etc.). The “modules” folder, inside the “Web Pages” contains all the module specific web pages and configuration files.
* Source folder: contains the java class related to the Re3gistry webapp (constants, module manager functionalities, servlets, utility, etc.).
* Resources folder: contains all the configurations and localization file. Below there is a description for each file:
  + *META-INF/persistence.xml*: in this file are specified all the configurations related to the database.
  + *configurations/Application.properties*: this file contains all the main Re3gistry configurations (common to all the modules);
  + *configurations/modules*: this folder contains all the properties related to each module. The file name shall be exactly as the name of the module’s folder.
  + *configurations/logcfg.xml*: the configuration related to the logging system;
  + *localization/Application/localizationBundle\_xx.properties*: the folder localization contains all the properties files related to the localization of the system. There is the subfolder “Application” that contains the core system localizations. Then, each module has its own localization file contained in a folder called exactly as the Library part of the module.

Figure - A view of the Re3gistry webapp module from an IDE



More information about the creation of a new module is provided at [chapter 8](#_Creating_new_modules).

# System installation and first run

There are two way to set up and run the system. The project package contains both the source code and a copy of the software precompiled. The example data is also provided in the package.

* The **precompiled webapp** is faster to install and run; this installation can be used to test the system in a quick way. Using this installation, and editing the XSLT example files, there is the possibility to produce all the data needed for a custom registry in a fast way.
* The **source code** needs to be imported in an Integrated Development Environment (IDE - such as NetBeans) and compiled. In this way, there is the possibility to execute the code and, if needed, to debug and edit the code itself (as well as extend and create new modules).

## Common installation steps

**Step 1 - External software installation**

Install the required software:

* Apache HTTPD;
* Java 1.7;
* Apache Tomcat 7;
* PostgreSQL 9.2.

Remember to set two different ports, linked to the Apache HTTPD and the Apache Tomcat. Typically the standard 80 port is used for the Apache HTTPD web server and the 8080 port for the Apache Tomcat. For detailed information related to the external software installation, refer to their relative manuals.

**Step 2 - Database creation and initialization**

Execute the database initialisation using the example SQL scripts available in the project’s package “example\database” folder.

The database script could be executed either from the command line or using a Graphic User Interface (GUI) such as pgAdmin.

Open the *create-tables.sql* into an SQL editor and run the queries. Do the same for the *database-initialization.sql*. It is important to run the *create-tables.sql* before the *database-initialization.sql*. The *database-initialization.sql* file contains some sample data.

Note: for each new register to be added, the register table, the itemcalss table and the localisation table should all be filled in; for more information related to the creation of new register, please refer to [paragraph 7.4](#_Initialize_the_database).

## Using the precompiled webapp

**Step 3 - Copy the project folders**

Before moving the project’s folder, make sure that the Tomcat server is not running. Take the “Re3gistry” folder from the “compiledsource” folder in the project package and move it in the tomcat’s “webapp” folder.

**Step 4 - Setting up the configurations**

Once the folders have been moved, open the configuration files for the core application and for the modules as explained below.

* ***Re3gistry\WEB-INF\classes\META-INF***
  + ***persistence.xml***: this file provides the configuration for the database. The parameters to be configured are: “javax.persistence.jdbc.url", "javax.persistence.jdbc.password", "javax.persistence.jdbc.user;
* ***Re3gistry\WEB-INF\classes\configurations***
  + ***Application.properties****:*this file contains all the main configurations for the system. To run the system the properties to be updated are “application.language.available” which represents the available language on the interface and the related language label, “application.contact”, “mail.sender”, “mail.recipient” which are the e-mail address of the contact, sender and respectively recipient and “mail.smtp.host” which is the server host smpt email;
  + ***logcfg.*xml**: this file contains all the configurations related to the logger system. In this file, the logger file path for each “log appender” has to be configured.
* ***Re3gistry\WEB-INF\classes\configurations\*modules\[modulename.properties]**: each module has its own properties file. To run the system, the configuration to be updated are:
  + ***Re3gistry/WEB-INF/classes/configurations*/modules/RegistryData.properties**: each property is detailed with comment directly in the property file. The important property to be updated is:

- “data.customtempfolder” (folder to save the temporary data files): if this property is left blank, a default folder is taken (inside the webapp folder);

- “data.operatinglanguage”: this property represents the main language of the system;

- data.supportedlanguage: this property represents the list of the language supported by the data management system.

* + ***Re3gistry/WEB-INF/classes/configurations*/modules/RegistryStaticizer.properties:**
    - “xml.formats.list=xml,json,html,atom”: the formats to be used by the *xslt* transformations (separated by comma)
    - “staticizer.SOLR.format=solr”: the SOLR name.
    - “staticizer.cutom.folder.path”: in this folder will be saved all the staticized files.
    - “staticizer.formats.path”: this property specifies the folder containing the xslt files for the transformations. Some example xslt files are available in the project package (example folder).

Start the server: once all the previous steps are completed, start the Tomcat server and test the installation.

Try to access the webapp using its address. An example of url to be called is provided below (this url may be differ depending on the system’s configuration):

*http://localhost:8080/Re3gistry*

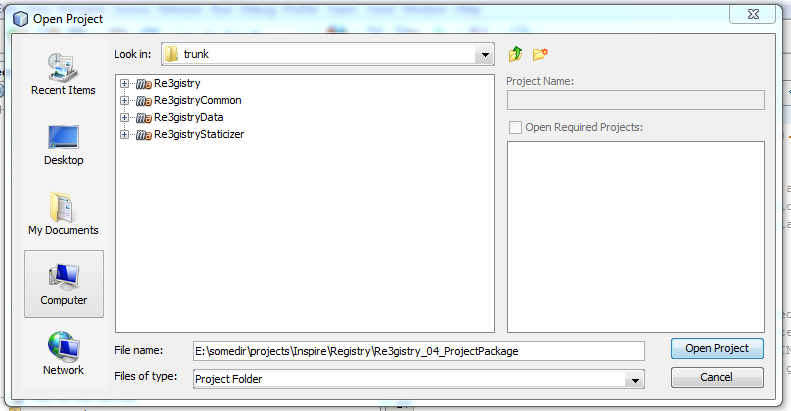
## Using the source code

The following steps detail how to install and run the Re3gistry for the first time using the *NetBeans IDE*. We choose NetBeans for this documentation because it is the official open source IDE provided by Java.

**Step 3 - Load projects**

To deploy and test the application, load the projects contained in the “source” folder of the package using the IDE (make sure it supports Maven).

Figure - Load projects with NetBeans IDE



**Step 4 - Configuration files**

After loading the projects, go to the configuration files under the “resources/META-INF” folder.

For convenience, all the configurations are placed under the “build profile” in the POM files of the Re3gistry web project. These Project Object Model (POM) files can be found under the project’s root folder.

Edit these configuration settings using the local settings:

1. The path for the logs file;
2. The information regarding the database;
3. The path for the temporary import folder/ the custom export data folder (if it is left blank, a standard folder, inside the webapp folder will be taken).

To customize the set up for all the feature of each module, check the following properties file:

* ***Re3gistry/src/main/resources/configurations/modules/*Application.properties:**
  + - this file contains all the main configurations for the system. To run the system the properties to be updated are “application.language.available” which represents the available language on the interface and the related language label, “application.contact”, “mail.sender”, “mail.recipient” which are the e-mail address of the contact, sender and respectively recipient and “mail.smtp.host” which is the server host smpt email;
* ***Re3gistry/src/main/resources/configurations/modules/*RegistryData.properties:**

- “data.operatinglanguage”: this property represents the main language of the system;

- data.supportedlanguage: this property represents the list of the language supported by the data management system.

* ***Re3gistry/src/main/resources/configurations/modules*/RegistryStaticizer/RegistryStaticizer.properties:**
  + - “xml.formats.list=xml,json,html,atom”: the formats to be used by the *xslt* transformations (separated by comma)
    - “staticizer.SOLR.format=solr”: the SOLR name.

An example of the POM file related to the profile is provided below:

Exmaple - Example POM file

<pofiles>

<profile>

<id>env-dev1</id>

<activation>

<property>

<name>env</name>

<value>dev1</value>

</property>

</activation>

<properties>

<!-- Loggers configurations -->

<Default.log.file.dir>C:/re3gistry-data/logs/Complete.log</Default.log.file.dir>

<Re3gistry.log.file.dir>C:/re3gistry-data/logs/Re3gistry.log</Re3gistry.log.file.dir>

<Re3gistryData.log.file.dir>C:/re3gistry-data/logs/Re3gistryData.log</Re3gistryData.log.file.dir>

<Re3gistryStaticizer.log.file.dir>C:/re3gistry-data/logs/Re3gistryStaticizer.log</Re3gistryStaticizer.log.file.dir>

<!-- Database configurations -->

<persistence.jdbc.url>jdbc:postgresql://db\_ip:5432/inspire\_registry </persistence.jdbc.url>

<persistence.jdbc.driver>org.postgresql.Driver</persistence.jdbc.driver>

<persistence.jdbc.username>db\_username</persistence.jdbc.username>

<persistence.jdbc.password>db\_password</persistence.jdbc.password>

<!-- Specific module configurations -->

<!-- RegistryData -->

<RegistryData.import.dir>C:/re3gistry-data/temp</RegistryData.import.dir>

<!-- / -->

<!-- RegistryStaticizer -->

<!-- The root folder where to *save* the files produced by the Staticizer. ->

<RegistryStaticizer.export.dir>C:/re3gistry-data/staticizer</RegistryStaticizer.export.dir>

<!-- The path from where to read the *xslt* transformation and the translations for the GUI interface -->

<RegistryStaticizer.xsl.dir>C:/re3gistry-data/xsl</RegistryStaticizer.xsl.dir>

</properties>

</profile>

</profiles>

Prior to starting the build process, ensure that the build profile (specified in the POM) for each the Re3gistry web project is selected. To do check this, in NetBeans go to “Set Configuration” under the project and choose the right profile (right click on the project -> “Set Configuration”).

**Step 5 - Build Projects**

To get the system running, build each project and start the web application (Re3gistry).

## Initialize the database with your data

After testing the system with the sample database initialization scripts, the system will be cleaned up and configured for the specific user needs.

* To clean up the example database run the dtop-tabels.sql script.
* Then run the create-table.sql script.
* Customize and run the database-initialization.sql to fit your needs. In particular, edit the first line regarding the “Register” and “ItemClass” insert statements.
* Customize and run the database-localization.sql in accordance to the new customized database-initialization.sql. This initialization file contains the language translation for the static items (Register name, Register description, ItemClass name, …).

## Running the Re3gistry software

Once the system is up and running, it can be tested using the example import data file (provided in the project package).

* Firstly, the database has to be filled using the Data module. To start the data procedure, go to the “Data” button in main the web interface menu.
* Select the example data file, from the project package, insert the procedure description and run the procedure.
* Once the import procedure is finished, go to Staticizer module (“Staticizer” button in the main menu - [*http://localhost:8080/Re3gistryExport*](http://localhost:8080/Re3gistryExport)) and by click on the “Check” button will be shown all the items imported from the last statizer procedure, now by click on the “Start” button will start the procedure of producing the standard files, the custom files using the xslt transformations, the VAR generator will also produce the files, and the last si the producing of the SOLR files.

NOTE: some of the xsl for the custom export (for performance reasons) uses the “base\_uri” property and reads the xml via a http request to this URL. Please check that the http server is running and that the “base\_uri” points to the local http server (edit the Operative System’s (OS) host file. This file can be place in different folder depending on the OS).

## Setting up the web service

Move the exported files (saved in the “*custom*” folder) under the Apache HTTPD document root (or under a specific, previously configured, folder).

To use the content-negotiation the Apache HTTPD has to be configured, following the details provided on the Apache HTTPD official web site, see: <http://httpd.apache.org/docs/2.2/content-negotiation.html>

# Creating new modules

To start creating a new module, it is important to understand the module’s structure by reading the chapter 6 and taking a deep look at one of the modules included in the package (Data module of Staticizer Module).

After understanding the structure, the development of a new module can be started from an IDE like NetBeans. The following guide is provided using the NetBeans IDE as example.

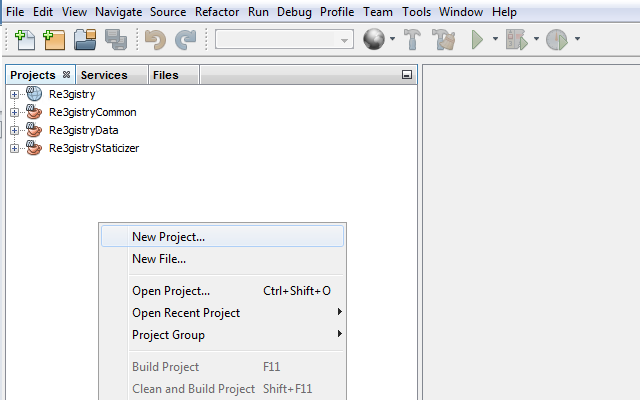
**Step 1**

Open the main project contained in the project package, following the instruction provided at [chapter 7](#_System_installation_and) ([paragraph 7.3](#_Using_the_source) “Using the source code”).

**Step 2**

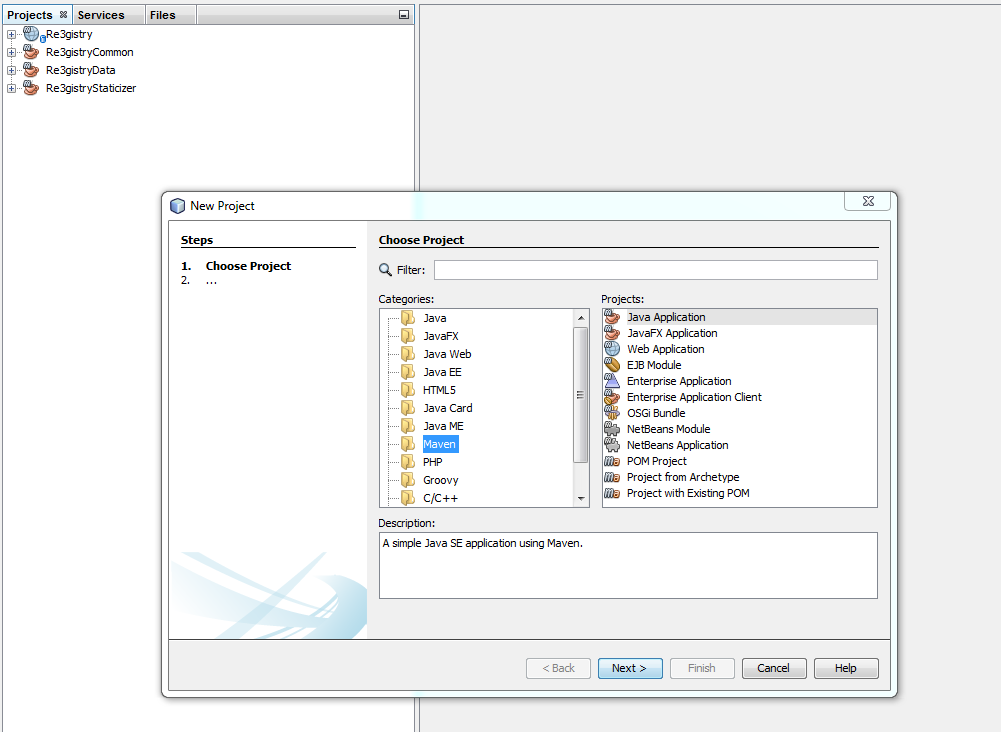
Create a new project: right click on the project browser and click on “new project”.

Figure - Creating new project from the project browser



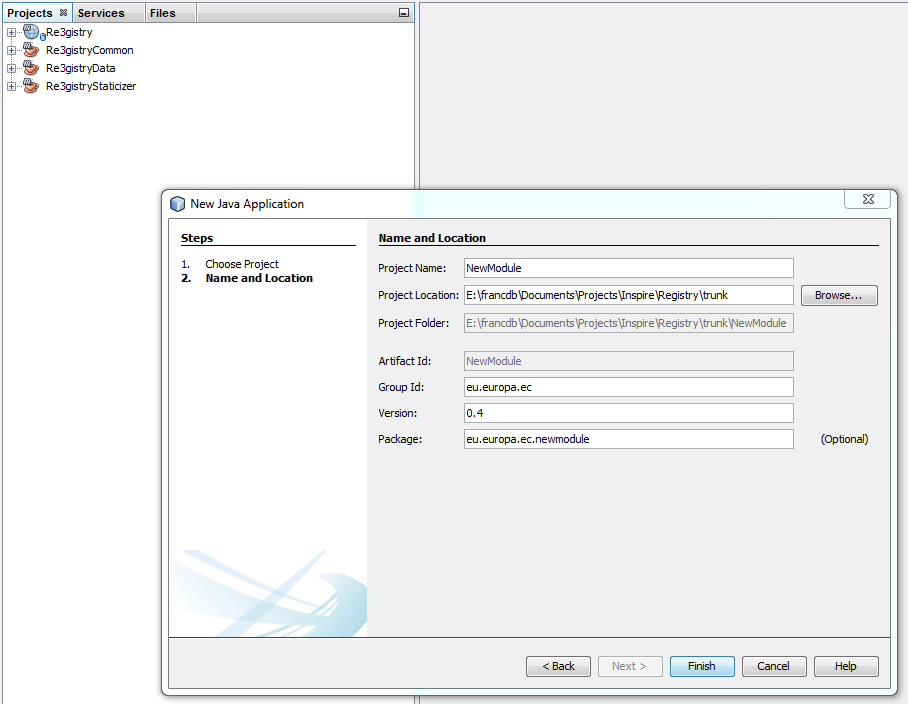
Choose “Maven” -> “Java Application” in the opened window, then click the “next” button.

Figure - Project type selection



In the following window, give a name to the new module; in this example the “NewModule” name is used.

Figure - Module name



**Step 3**

Go to the newly created project and edit the POM file, to insert the dependency to the RegistryCommon module as shown at example 16 .

Exmaple - POM edit example

<dependency>

<groupId>eu.europa.ec.re3gistrycommon</groupId>

<artifactId>Re3gistryCommon</artifactId>

<version>0.4</version>

</dependency>

**Step 4**

The library component of the module is now ready. To start the development, create the required package and classes in the project’s source folder.

It is important to create the Class “Constants.java” under package “eu.europa.ec.newmodule.constant”. This is a common used Class containing all the constants related to this project.

This class shall contain at least the class name. An example is provided below.

Exmaple – Constants.java example

package eu.europa.ec.newmodule.constants;

public final class Constants {

/\* ###################################### \*/

/\* ########## Common constants ########## \*/

/\*\*

\* The name of this module

\*/

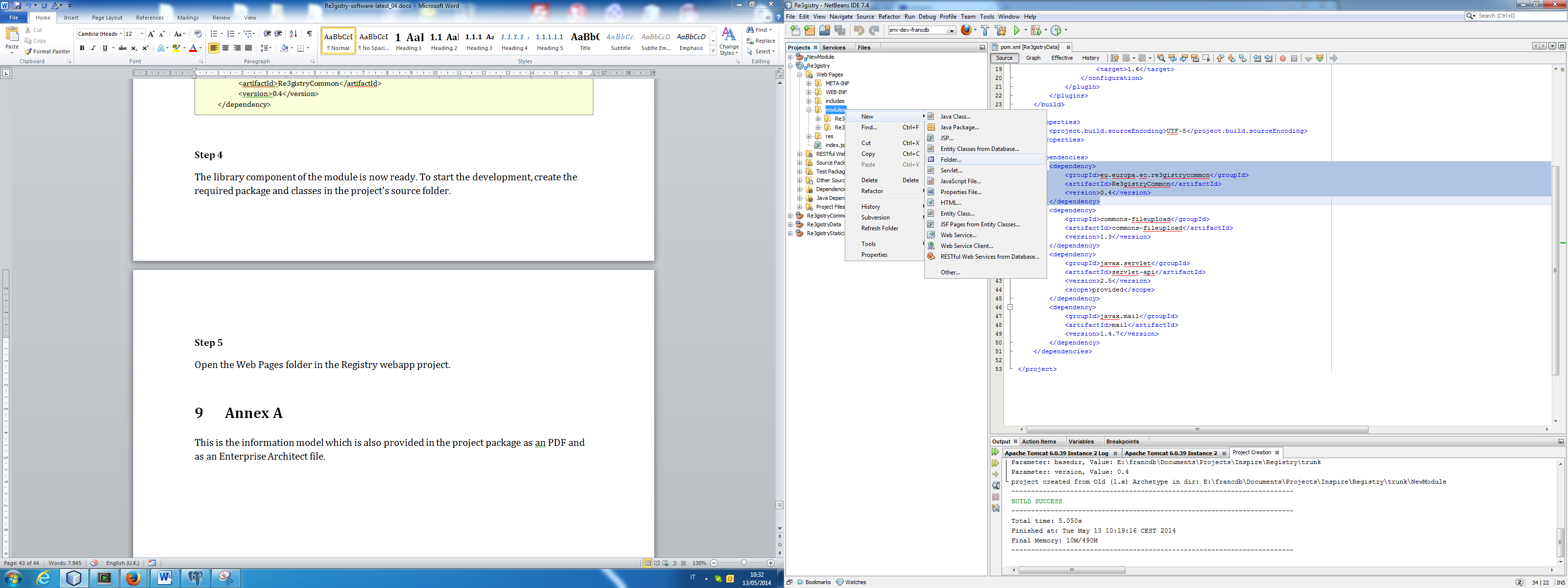
public static final String MODULE\_NAME = "NewModule";

}

**Step 5**

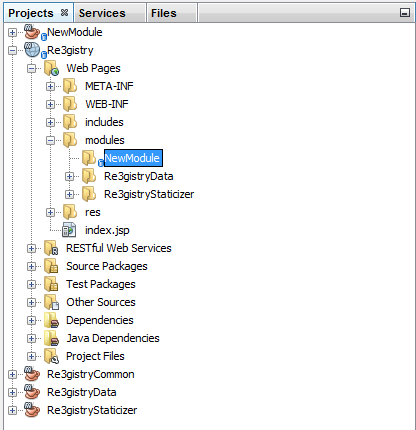
Open the Web Pages folder in the Registry webapp project. Search the modules folder, right click on it and select “New” - > “Folder”.

Figure - Creation of the module's folder



Name the new folder exactly as the previously created Java Application (at step 2).

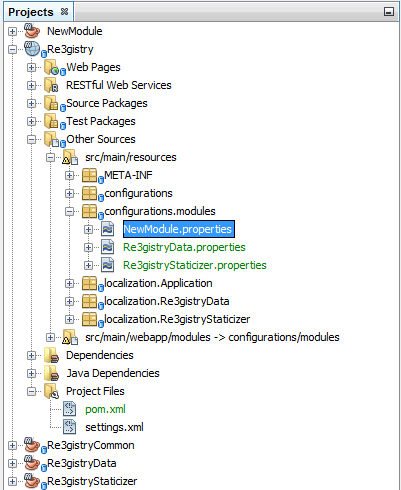
Figure - The new folder in the Re3gistry webapp project



**Step 6**

Create the module’s property file, inside the “Other sources – configurations/modules” project folder (in the file system this folder is located under ***Re3gistry/ WEB-INF/classes/configurations/modules***) with the same module name (“NewModule.properties”).

Figure - Module's properties file



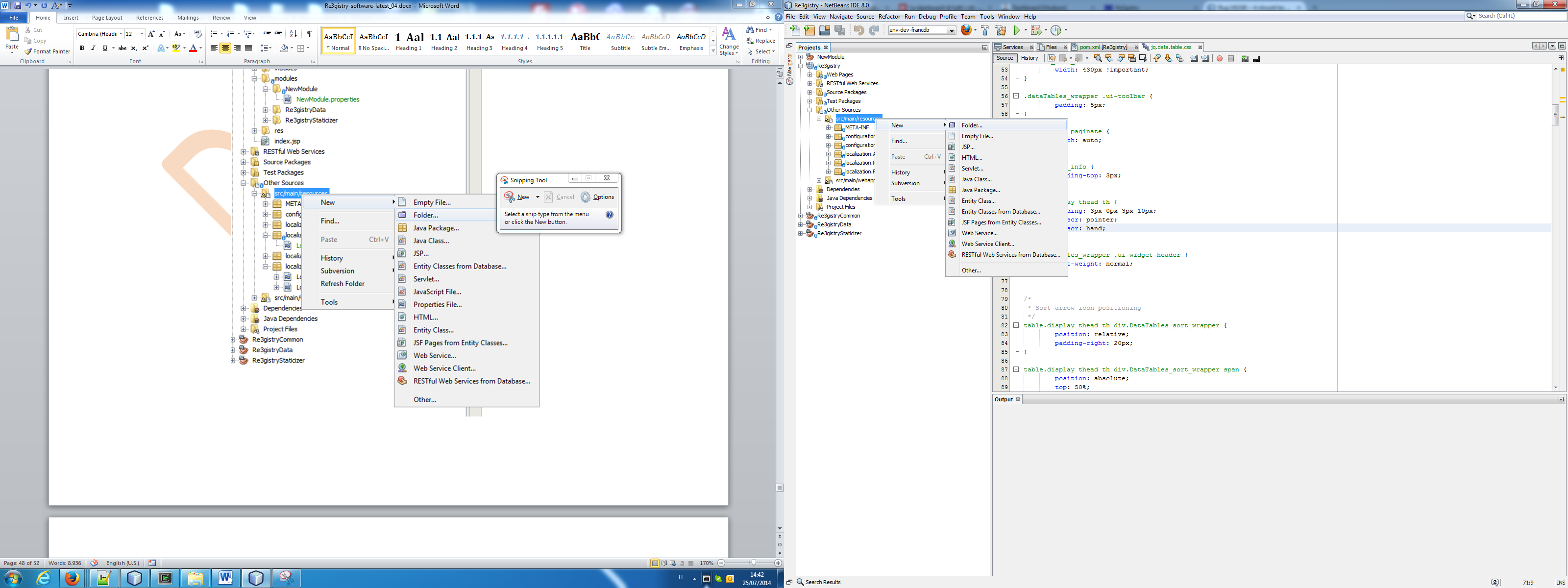
The required property keys for all the new modules are shown below:

* **module.active=true** – if set to true, the module is active and is loaded in the system.
* **module.menuitem.visible=true** – if this property is true, the menu entry related to this module is shown in the main system webpage.
* **module.menuitem.label=Data** – This property rapresent the label of the menu item in the system main menu.
* **module.homewidget.enabled=true** – This property enable/disable the system to show the module widget in the home page.
* **module.dateformat=dd-MM-yyyy HH:mm:SS** – The module dat/time format.
* **module.menuitem.order=2 –** The order in which this menu item is shown in the main menu. The higher values mean to be displayed later.

**Step 7**

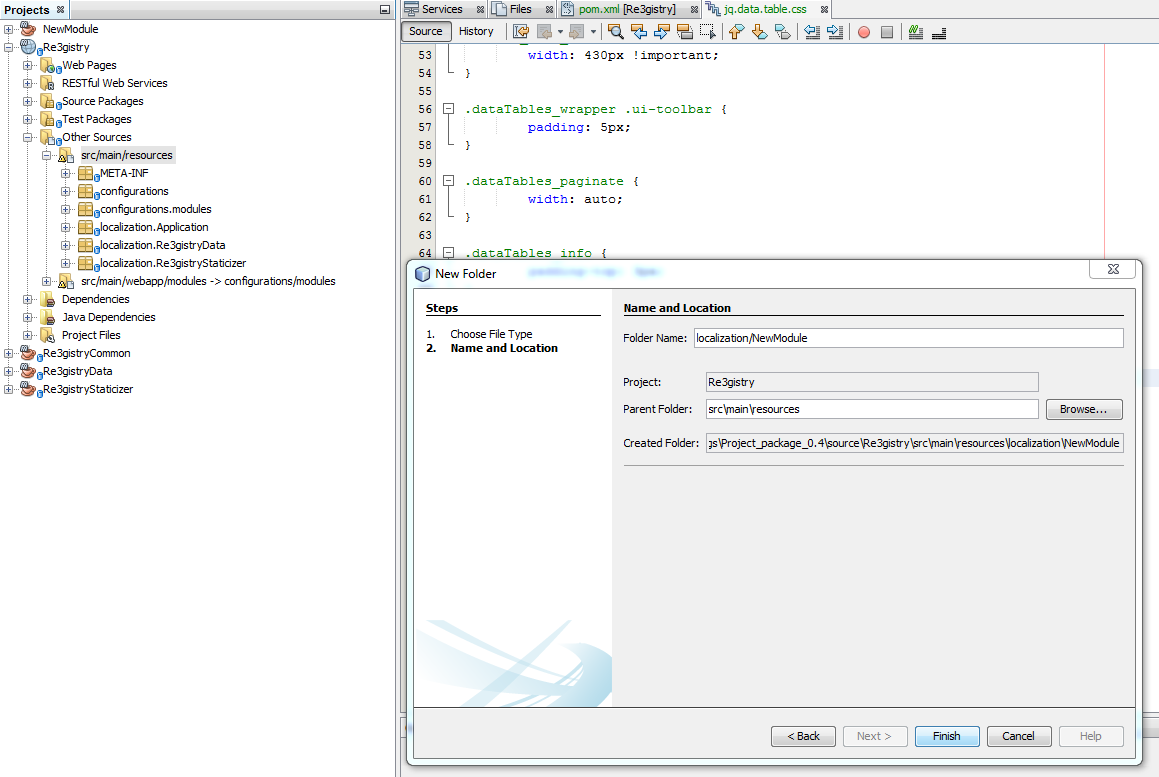
Add the localization files in the “Other sources” -> “src/main/resources/localization” folder. To do this, right click on the “Other sources” -> “src/main/resources” folder and select “New” -> “Folder”.

Figure - Localization folder for the new module



Name the new folder “localization/NewModule”.

Figure - Localization properties folder

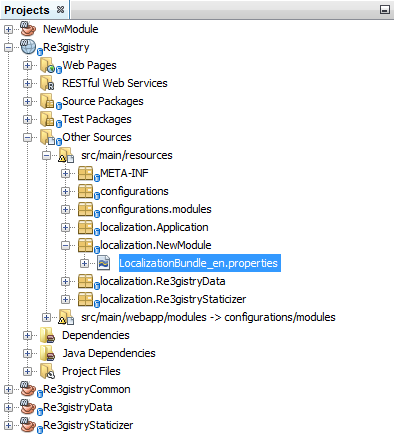


Then create a file for each language using this naming pattern: “LocalizationBundle\_[language\_code].properties”.

The minimum set of property to be available in the localization files are:

* The module’s title: common.pagetitle=New Module
* The module’s description: main.description=Descriptive text

Figure - Module's localization properties file



**Step 8**

Add into the logger configurations the entry relative to this new module. Open the logcfg.xml file under “Other sources” –> configuration.

Add the entry related to the new module as defined below. Note that this example is using maven profile properties. In this case the property key related to this logger file has also to be added to the pom file.

Exmaple – Logger configurations

<!-- NewModule -->

<appender name="appender.NewModule" class="org.apache.log4j.DailyRollingFileAppender">

<param name="File" value="${NewModule.log.file.dir}"/>

<param name="DatePattern" value="'.'yyyy-MM-dd"/>

<layout class="org.apache.log4j.PatternLayout">

<param name="ConversionPattern" value="%d - %-5p - %-10c [%C{1}.%M:%L] %m%n"/>

</layout>

</appender>

<logger name="NewModule">

<level value="info"/>

<appender-ref ref="appender.NewModule"/>

</logger>

**Step 9**

The module structure is now complete.

Now the next step is to create the required web pages and to start the development of the functionalities for the module (both in the library and in the web part).

The web pages shall be placed in the module’s folder root, created at step 5.

There are 2 file required:

* main.jsp: contain the main web page of the module, with all the functionalities.
* widget.jsp: this page represents a sort of summary that can be placed in the home page of the system.

**Notes**

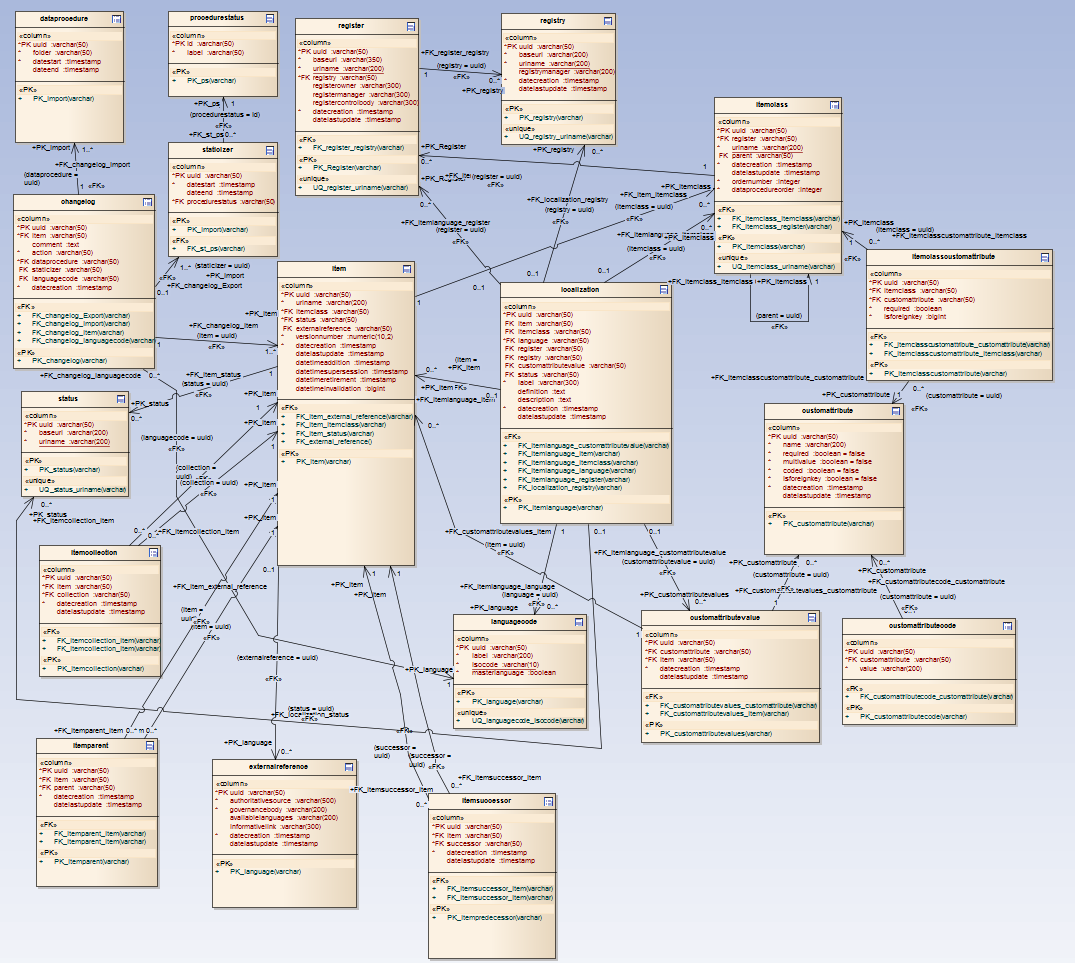
The main structure of the modules can be easily understood by taking a look at the files from the modules available in the project package.

There are some particular module related code snippets, used in the web pages that are explained below.

* ${module.localization['property.name']} – It is used to retrieve a specified text in the language selected by the user. The keys passed as argument to this method has to be defined in the module specific localization file (created at step 7).
* ${module.properties['property.name']} – It is used to retrieve a specific properties related to this module. The keys passed as argument are defined in the module’s related properties file (defined at step 6).

# Annex A

This is the information model which is also provided in the project package as PDF and as 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. The complete information model is available in the Annex A of this document and as attachment in the software package. [↑](#footnote-ref-4)
5. ISO 19135-1 – Procedures for item registration – Part 1: Fundamentals [↑](#footnote-ref-5)