

Re³gistry

Software documentation - Version 1.3

The following documentation provides details related to the [Re3gistry](#) software, including the installation instructions. It provides also several sections including guides to create a custom registry service starting from an example and a guide to extend the software by creating new modules.

This is a live document; it is being improved continuously. To have the last version you can refer to <https://ies-svn.jrc.ec.europa.eu/projects/registry-development>

Please report any feedback on the documentation at: inspire-registry-dev@jrc.ec.europa.eu

Reuse is authorised, provided the source is acknowledged. The reuse policy of the European Commission is implemented by a [Decision of 12 December 2011](#)¹.

¹ <http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32011D0833>

Table of contents

| | |
|---|----|
| Table of contents | 3 |
| Abbreviations | 11 |
| Bibliography..... | 12 |
| 1. Overview | 13 |
| 1.1. What is the Re3gistry?..... | 13 |
| 1.2. Features and capabilities..... | 13 |
| 1.3. Getting the software..... | 13 |
| 1.4. License | 14 |
| 1.5. Background | 14 |
| 1.1. Acknowledgments | 15 |
| 1.2. What is new in the Re3gistry 1.3 | 15 |
| 2. Understanding the Re3gistry | 16 |
| 2.1. The Re3gistry information model | 16 |
| 2.1.1. The information model components | 16 |
| 2.1.1.1. The <i>registry</i> component..... | 16 |
| 2.1.1.2. The <i>register</i> component..... | 16 |
| 2.1.1.3. The <i>itemclass</i> component..... | 17 |
| 2.1.1.4. The <i>item</i> component..... | 17 |
| 2.1.1.5. The <i>status</i> component | 17 |
| 2.1.2. Standard and customised attributes | 18 |
| 2.1.2.1. Registry standard attributes..... | 18 |
| 2.1.2.2. Register standard attributes | 18 |
| 2.1.2.3. Item standard attributes..... | 19 |
| 2.1.2.4. Custom attributes | 19 |
| 2.1.3. Language representations: localisation | 19 |
| 2.2. The Re3gistry system architecture | 20 |
| 2.2.1. Modules overview..... | 20 |
| 2.2.2. Registry core module | 21 |
| 2.2.3. Data import module..... | 21 |
| 2.2.3.1. Import data file | 22 |
| 2.2.3.1.1. Data file structure | 22 |

| | | |
|------------|--|----|
| 2.2.3.1.2. | Internal and external items..... | 22 |
| 2.2.3.1.3. | Data actions and CSV formats..... | 23 |
| 2.2.3.2. | Data analyser | 30 |
| 2.2.3.3. | Data storage | 31 |
| 2.2.4. | Staticiser module..... | 32 |
| 2.2.4.1. | Staticisation process | 32 |
| 2.2.4.2. | XSLT | 32 |
| 2.2.4.3. | Static element localisation..... | 33 |
| 2.2.4.4. | Additional information..... | 34 |
| 2.2.5. | Deployer module..... | 34 |
| 2.3. | Re3gistry administration panel | 35 |
| 3. | Installing the Re3gistry | 36 |
| 3.1. | System requirements | 36 |
| 3.2. | Package details..... | 36 |
| 3.3. | Important notes | 37 |
| 3.4. | Database configuration | 37 |
| 3.4.1. | Creating a new database | 37 |
| 3.4.2. | Running the SQL scripts to create tables and populating them | 37 |
| 3.5. | Configuring the Re3gistry | 39 |
| 3.5.1. | Move the binaries folder to Tomcat's webapp folder | 40 |
| 3.5.2. | Modifying the configuration files | 40 |
| 3.5.2.1. | persistence.xml | 41 |
| 3.5.2.2. | Application.properties | 41 |
| 3.5.2.3. | logcfg.xml | 41 |
| 3.5.2.4. | Re3gistryData.properties | 42 |
| 3.5.2.5. | Re3gistryStaticizer.properties..... | 42 |
| 3.5.2.6. | Re3gistryDeployer.properties [Optional]..... | 43 |
| 3.5.3. | Setting up the authentication method | 44 |
| 3.5.3.1. | Available authentication methods..... | 44 |
| 3.5.3.2. | Choosing and implementing the authentication method..... | 44 |
| 3.5.3.2.1. | Application.properties | 44 |
| 3.5.3.2.2. | web.xml..... | 44 |
| 3.5.4. | Adding users to SHIRO | 45 |

| | | |
|------------|---|----|
| 4. | Using the Re3gistry | 46 |
| 4.1. | Accessing the Re3gistry administration panel | 46 |
| 4.2. | Importing data | 47 |
| 4.3. | Exporting and converting data files | 49 |
| 4.4. | Deploying the contents | 50 |
| 4.4.1. | Moving data from the Re3gistry software to the server | 50 |
| 4.4.2. | Creating a modification summary RSS feed | 50 |
| 5. | Serving the Re3gistry contents | 52 |
| 5.1. | System requirements | 52 |
| 5.2. | Web service | 52 |
| 5.2.1. | RESTful web service | 53 |
| 5.2.2. | Standard web service | 54 |
| 5.3. | Installing the Re3gistry webapp | 54 |
| 5.3.1. | Copy the sample web application folder | 55 |
| 5.3.2. | Setting the web application | 55 |
| 5.3.3. | Configuration | 55 |
| 5.3.3.1. | conf.php | 56 |
| 5.3.3.2. | logger.xml | 56 |
| 5.3.4. | Configuring the HTTP server | 57 |
| 5.3.5. | Set up the service-specific configuration | 58 |
| 5.4. | Managing <i>solr</i> | 59 |
| 5.4.1. | Installing <i>solr</i> | 59 |
| 5.4.2. | Configuring <i>solr</i> | 59 |
| 5.4.2.1. | solr.xml | 59 |
| 5.4.3. | <i>core.properties</i> of <i>solr</i> registry collection | 60 |
| 5.4.3.1.1. | Schema.xml | 60 |
| 5.5. | Connecting <i>solr</i> to the Re3gistry webapp | 61 |
| 5.6. | Indexing your registry contents | 61 |
| 5.7. | Testing the web service | 62 |
| 5.8. | INSPIRE register federation descriptors files - RoR descriptors | 64 |
| 6. | Customising the Re3gistry | 65 |
| 6.1. | Customising the Re3gistry contents | 65 |
| 6.1.1. | Creating a costumised registry | 65 |

| | | |
|------------|---|----|
| 6.1.1.1. | Setting the registry parameters | 65 |
| 6.1.1.2. | Defining the email address for the registry contact point | 66 |
| 6.1.1.3. | Setting the supported languages | 66 |
| 6.1.1.4. | Setting the status values | 67 |
| 6.1.2. | Creating customised registers | 67 |
| 6.1.2.1. | Setting the register parameters | 67 |
| 6.1.2.2. | Defining the itemclass | 68 |
| 6.1.3. | Translating the content | 69 |
| 6.1.4. | Import data file | 71 |
| 6.1.4.1. | Simple register | 73 |
| 6.1.4.2. | Hierarchical register | 74 |
| 6.1.4.2.1. | Hierarchical register – first level | 74 |
| 6.1.4.2.2. | Hierarchical register – second level | 74 |
| 6.1.5. | Transformation files | 75 |
| 6.1.6. | Deployer configuration | 75 |
| 6.2. | Customising the Re3gistry web interface | 76 |
| 6.2.1. | Webapp structure | 79 |
| 6.2.2. | Modes | 79 |
| 6.2.2.1. | GUI localisation file | 82 |
| 6.2.2.2. | MODE 1 descriptor - Registry page | 82 |
| 6.2.2.3. | MODE 2 descriptor - Register page | 84 |
| 6.2.2.4. | MODE 3 descriptor - Item detail page | 87 |
| 6.2.2.5. | MODE 4 Descriptor - Item detail for hierarchical elements | 90 |
| 6.2.3. | Static pages | 93 |
| 6.2.3.1. | Static page example: status.descriptor.json | 93 |
| 6.2.4. | Customised pages | 95 |
| 6.2.4.1. | Example custom page descriptor | 95 |
| 6.2.5. | Website parts | 95 |
| 7. | Developing the Re3gistry | 97 |
| 7.1. | Technology | 97 |
| 7.1.1. | Web Server | 97 |
| 7.1.2. | Database | 97 |
| 7.2. | System structure | 97 |

| | | |
|----------|---|-----|
| 7.2.1. | Module concept | 97 |
| 7.2.2. | <i>Re3gistryCommon</i> module | 98 |
| 7.2.3. | Re3gistry software interface | 98 |
| 7.3. | Source code..... | 99 |
| 7.3.1. | Load projects | 99 |
| 7.3.2. | Configuration files | 99 |
| 7.3.3. | Choose the authentication method | 102 |
| 7.3.4. | Database creation and initialisation..... | 104 |
| 7.3.5. | Build projects..... | 104 |
| 7.3.6. | Creating new modules | 104 |
| 7.3.6.1. | Step 1 | 104 |
| 7.3.6.2. | Step 2 | 105 |
| 7.3.6.3. | Step 3 | 107 |
| 7.3.6.4. | Step 4 | 107 |
| 7.3.6.5. | Step 5 | 107 |
| 7.3.6.6. | Step 6 | 108 |
| 7.3.6.7. | Step 7 | 109 |
| 7.3.6.8. | Step 8 | 112 |
| 7.3.6.9. | Step 9 | 113 |
| | Index of keywords | 114 |

Index of figures and tables

| | |
|---|----|
| Figure 1: Simplified information model | 16 |
| Figure 2: Schematic system description | 20 |
| Figure 3: System description | 21 |
| Figure 4: Example of a zip file structure | 22 |
| Figure 5: Encoding of the addition CSV file when several values are not available | 23 |
| Figure 6: configuration of the mandatory field property..... | 24 |
| Figure 7: Addition CSV example - internal items | 25 |
| Figure 8: Addition CSV example - external items | 26 |
| Figure 9: Clarification CSV file example | 27 |
| Figure 10: Example of supersession CSV file | 28 |
| Figure 11: Supersession CSV example | 29 |
| Figure 12: Retirement CSV file example | 30 |
| Figure 13: XSLT transformation for XML..... | 32 |
| Figure 14: XSLT transformations files for the custom XML format (contained in the xml folder).. | 33 |
| Figure 15: GUI-languages folder, French file structure..... | 33 |
| Figure 16: Usage example. Place this piece of code to retrieve the translated word in the file. ... | 33 |
| Figure 17: Structure of the Re3gistry package | 36 |
| Figure 18: Executing the create-tables.sql script from pgAdminIII | 38 |
| Figure 19: Nineteen tables created and populated | 39 |
| Figure 20: Usual Tomcat installation file structure | 40 |
| Figure 21: Default authentication method | 44 |
| Figure 22: Shiro.ini users section | 45 |
| Figure 23: Authentication page to access the Re3gistry software administration panel | 46 |
| Figure 24: Data management system page | 47 |
| Figure 25: 'Add data file' window | 47 |
| Figure 26: Procedure details in the data management system panel | 48 |
| Figure 27: Produced data in the 'custom' folder for the 'neutral-example' | 50 |
| Figure 28: Windows-deploy.bat contents..... | 50 |
| Figure 29: Popup window allowing creating or updating an RSS file | 51 |
| Figure 30: Example of content-negotiation parameters..... | 53 |
| Figure 31: .var file example | 54 |
| Figure 32: Web service requests - Direct URL example | 54 |
| Figure 33: Paste the webapp folder into the exported data folder | 55 |
| Figure 34: Proposed structure to locate the webapp files and the data files..... | 55 |
| Figure 35: Properties in the conf.php file | 56 |
| Figure 36: conf.php file opened in Notepad++..... | 56 |
| Figure 37: Property in the logger.xml configuration file where the log file is defined | 57 |
| Figure 38: httpd.conf file configuration..... | 57 |
| Figure 39: Settings to configure the HTTP server..... | 58 |
| Figure 40: app_data folder structure after editing | 59 |
| Figure 41: Configuration of the solr.xml file | 60 |
| Figure 42: New 'registry' collection for solr..... | 60 |

| | |
|--|-----|
| Figure 43: solr administration panel | 61 |
| Figure 44: Example of definition of solr endpoint in the conf.php | 61 |
| Figure 45: command to update the indexing of solr | 62 |
| Figure 46: Querying indexed contents within the solr administration panel, | 62 |
| Figure 47: Re3gistry generic user interface serving the 'neutral-example' registers contents | 63 |
| Figure 48: Autocomplete function | 63 |
| Figure 49: Generic URL to get the ROR file in the re3gistry | 64 |
| Figure 50: Fields of registry table of the database | 65 |
| Figure 51: Fields of reference table of the database | 66 |
| Figure 52: Language configuration | 66 |
| Figure 53: Definition of the register status | 67 |
| Figure 54: Register section of the script | 68 |
| Figure 55: itemclass section of the script | 68 |
| Figure 56: Fields in the Localization table of the database | 69 |
| Figure 57: localisation settings | 70 |
| Figure 58: Example of localisation file for the English language | 70 |
| Figure 59: Settings in Open Office Calc to open appropriately he Re3gistry import actions files .. | 72 |
| Figure 60: Re3gistry import action files in a spreadsheet (Open Office Calc) | 72 |
| Figure 61: Re3gistry import action files in a notepad program (Notepad++) | 73 |
| Figure 62: Example of an addition file with an additional customised attribute in a simple register | 73 |
| Figure 63: Example of addition in hierarchical register using the addition.csv of its itemclass | 74 |
| Figure 64: Example of invalidation of an item | 74 |
| Figure 65: Creation of the second level of a hierarchical register | 75 |
| Figure 66: XSLT files needed to support the conversion in HTML formats for the simple and hierarchical registers | 75 |
| Figure 67: Addresses theme within the theme register of the INSPIRE Registry | 80 |
| Figure 68: Restriction code code list within the code list register of the INSPIRE Registry | 81 |
| Figure 69: Example of configuration string for the 'Status' static page | 94 |
| Figure 70: Loading the project with NetBeans IDE | 99 |
| Figure 71: Example POM file | 100 |
| Figure 72: Creating new project from the project browser | 105 |
| Figure 73: Project type selection | 106 |
| Figure 74: Module name | 106 |
| Figure 75: POM edit example | 107 |
| Figure 76: Constants.java example | 107 |
| Figure 77: Creation of the module's folder | 108 |
| Figure 78: Module's properties file | 109 |
| Figure 79: Localization folder for the new module | 110 |
| Figure 80: Localization properties folder | 111 |
| Figure 81: Module's localization properties file | 112 |
| Figure 82: Logger configurations | 113 |

Table 1: Field structure in the addition CSV file24

Table 2: CSV fields for the clarification data file26

Table 3: CSV fields for the supersession data file27

Table 4: CSV fields for the invalidation data file28

Table 5: CSV fields for the retirement data file29

Table 6: Analyser check and report message type by action30

Table 7: Procedure statuses35

Table 8: RESTful URL example53

Abbreviations

| | |
|---------|---|
| ARE3NA | A Reusable INSPIRE Reference Platform |
| CSV | Comma Separated Values file |
| EC | European Commission |
| ECAS | European Commission Authentication Service |
| GUI | Graphical user interface |
| HTML | HyperText Markup Language |
| HTTP | Hypertext Transfer Protocol |
| INSPIRE | INfrastructure for SPatial Information in Europe |
| ISA | <i>Interoperability Solutions for European Public Administrations</i> |
| JRC | Joint Research Centre |
| php | Hypertext Preprocessor |
| POM | Project Object Model |
| ROR | Register Of Registers |
| SQL | Structured Query Language |
| URI | Uniform Resource Identifier |
| URL | Uniform Resource Locator |
| RESTful | Representational State Transfer |
| RSS | Really Simple Syndication |
| XSLT | Extensible Stylesheet Language Transformations |
| webapp | Web application |

Bibliography

| | |
|---------------------|---|
| ISA | Interoperability Solutions for European Public Administrations http://ec.europa.eu/isa |
| INSP-DIR | Infrastructure for Spatial Information for Europe (INSPIRE) Directive 2007/2/EC http://eur-lex.europa.eu/legal-content/EN/ALL/?uri=CELEX:32007L0002 |
| WTF | The EU Water Framework Directive - integrated river basin management for Europe http://ec.europa.eu/environment/water/water-framework |
| SEIS | Shared Environmental Information System http://ec.europa.eu/environment/seis/ |
| ISO-19135 | Geographic information -- Procedures for item registration -- Part 1: Fundamentals http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=54721 |
| INSP-THEME | INSPIRE theme register http://inspire.ec.europa.eu/theme |
| INSP-REG | INSPIRE registry http://inspire.ec.europa.eu/registry |
| INSP-CODELIST | INSPIRE code list register http://inspire.ec.europa.eu/codelist |
| APACHE-CN | Apache HTTPD - Content negotiation http://httpd.apache.org/docs/2.4/content-negotiation.html |
| NETBEANS | NetBeans IDE https://netbeans.org |
| APACHE-HTTPD | Apache HTTPD server https://httpd.apache.org |
| APACHE-TOMCAT | Apache Tomcat server http://tomcat.apache.org |
| APACHE-CONFIG-FILES | Apache Configuration files https://httpd.apache.org/docs/current/configuring.html) |
| EclipseLink | EclipseLink Java persistence solution http://www.eclipse.org/eclipselink |
| SHIRO | Apache SHIRO http://shiro.apache.org |
| pgAdmin | PostgreSQL administration and management tools https://www.pgadmin.org |

1. Overview

1.1. What is the Re3gistry?

The **Re3gistry** is a reusable open source solution for managing and sharing 'reference codes'. It provides a consistent central access point where labels and descriptions for reference codes can be easily browsed by humans and retrieved by machines.

Public administrations, businesses and citizens regularly exchange data across borders and sectors using reference codes. The usefulness of the reference codes depends on their proper management. Shared codes cannot change or simply disappear over time, all versions of a code need to be traceable and properly documented.

1.2. Features and capabilities

- CSV data import with consistency checking
- Highly flexible and customisable
- Supported formats: HTML, XML, JSON, RDF, Atom, CSV
- Formats that can be easily customised or new formats added through transformation files
- An underlying model for register items that can also be easily customised
- Support for multi-lingual content
- Support for versioning
- RESTful API with content negotiation
- Free-text search
- Support for web service deployment
- Highly performant access to register content
- Integration with ECAS authentication
- The solution has been developed following the Standard *ISO 19135* 'Procedures for item registration' [*ISO-19135*].
- Externally governed items referenced through the URI
- Support to the INSPIRE register federation format (RoR)

1.3. Getting the software

The **Re3gistry** software is freely available for download at:
<https://joinup.ec.europa.eu/software/Re3gistry/release/all>

To provide feedback on the software we kindly invite you to contact us at: inspire-registry-dev@jrc.ec.europa.eu

1.4. License

The **Re3gistry** is released under the European Union Public Licence - EUPL v.1.1 ².

1.5. Background

The European Union (EU) Member States are currently implementing the INSPIRE Directive [*INSPIRE-DIR*] and related regulations. Technical guidelines³ 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 [*WTF*], Digital Agenda for Europe, open data, Shared Environmental Information System (SEIS) [*SEIS*] 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*⁴, to aid wide re-use.

² https://joinup.ec.europa.eu/sites/default/files/eupl1.1.-licence-en_0.pdf

³ <http://inspire.ec.europa.eu/inspire-technical-guidance/57753>

⁴ <https://joinup.ec.europa.eu/software/re3gistry/description>

1.1. Acknowledgments

As part of the *Interoperability Solutions for European Public Administrations (ISA) Programme*⁵ [ISA], 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 [INSP-DIR].

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 engineered by the ISA founded resources Daniele Francioli and Emanuela Epure.

We are also grateful for the review of this document to Lorena Hernandez Quirós and Robin S. Smith.



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

ARE³NA

<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/>

1.2. What is new in the Re3gistry 1.3

This is the version 1.3 of the Re3gistry software.

The improvements and changelog of this version are:

- Support to the external items, referenced through the URI with the possibility to store also additional metadata (like the label, status, etc.).
- Support to the INSPIRE register federation format (*RoR*)
- Bug fixes

⁵ http://ec.europa.eu/isa/index_en.htm

2. Understanding the Re3gistry

2.1. The Re3gistry information model

The **Re3gistry** software populates the registry contents from the importing of simple text-based data files. Then it organises and exports the data in different formats. The produced files can then optionally be served online through a customisable web service.

2.1.1. The information model components

A simple representation of the system's information model is shown in Figure 1: Simplified information model.

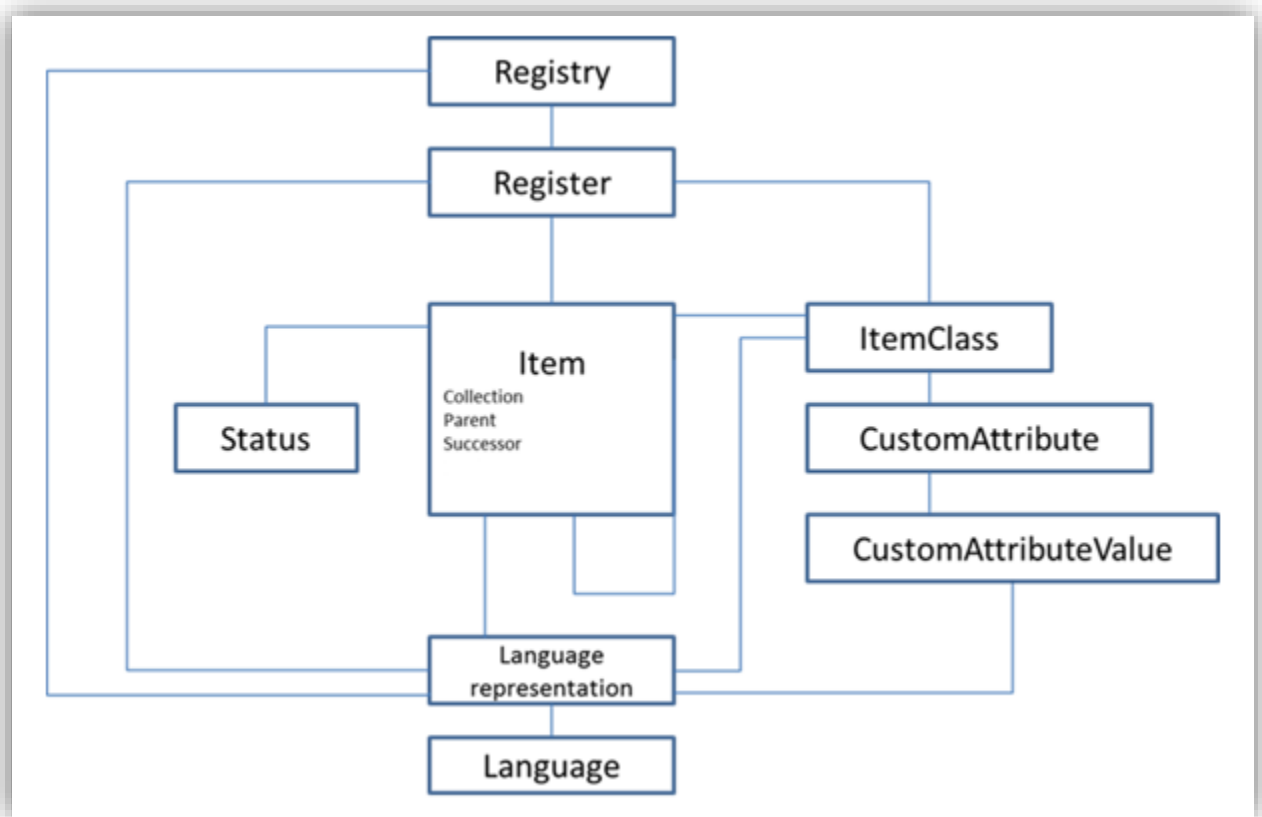


Figure 1: Simplified information model

2.1.1.1. The *registry* component

A *registry* is an information system on which registers are maintained [ISO 19135].

2.1.1.2. The *register* component

A *register* is a single controlled collection or a list of items with unique identifiers. Each register is operated on behalf of some owner organisation that provides the authority for the collection.

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

2.1.1.3. The *itemclass* component

The *itemclass* represents a set of items with common properties. It defines a group of items contained in a specific register.

For example, in a 'theme register', such as the INSPIRE Theme register [INSP-THEME], the itemclass of the items contained can be identified as 'theme items'.

Itemclasses can have parent/child(s) relationship within the same register. In this case the register is a hierarchical register.

For example, in the INSPIRE Registry [INSP-REG], the 'code list register' [INSP-CODELIST] has two itemclasses: CodeList and CodeListValue. More specifically, the CodeList itemclass is the parent of the CodeListValue one. The CodeListValue itemclass contains the items that are part of the 'Code List' collection.

The 'collection' relation and the hierarchy between the itemclasses is used to handle hierarchical registers.

2.1.1.4. The *item* component

The items are elements that can be contained in a register. In a hierarchical register, an item can also be a container (collection) of other items.

For example, in the INSPIRE registry [INS-REG], the itemclass 'CodeList' could contain other items (items with itemclass 'CodeListValue'). The contained item (the 'code list value item'), indicates which collection it belongs to by means of the `collection_id` field in the import file.

Another type of relation between items of the same itemclass is the parent/child relation. This kind of relation is provided by the `parent_id` field in the data import file. This field represents the reference to the `local_id` of the parent item.

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

2.1.1.5. The *status* component

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

- **Valid:** the item has been accepted. It 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.

2.1.2. Standard and customised attributes

Each element represented by the [Re3gistry](#) has a standard list of attributes.

2.1.2.1. Registry standard attributes

| | |
|------------------|---|
| Id | Identifier of the registry |
| Label | Label of the registry |
| Content summary | Description of the purpose for which the registers and their items managed in the registry are made available to potential users. |
| Registry manager | Person or organization responsible for the day-to-day management of the registry |

2.1.2.2. Register standard attributes

| | |
|------------------|---|
| Id | Identifier of the register |
| Label | Label of the register |
| Content summary | Description of the purpose for which items in the register are made available to potential users. It should also specify any limits to the scope of the register and identify the types of applications for which the items are intended. |
| Owner | Organization that establishes a register |
| Register manager | Organization to which management of a register has been delegated by the register owner |
| Control body | Group of technical experts that makes decisions regarding the content of a register. |
| Submitter | Organization authorized by a register owner to propose changes to the content of a register |
| Contact point | Name or position of the person who serves as a point of contact for information about the register owner and the register |
| License | The license under which the register content is being made available |

The information related to the *registry* and the *register* can be configured and edited in the system during the installation process (Refer to section [3.4 Database configuration](#)).

2.1.2.3. Item standard attributes

| | |
|-------------|--|
| Id | Identifier of the item |
| Label | Label of the item |
| Definition | Definition of the item (precise statement of the nature, properties, scope or <i>essential</i> qualities of the item) |
| Description | Description of the element (statement of the nature, properties, scope, or <i>non-essential</i> qualities of the item but are not specified by the definition) |
| Status | Status of the item |

2.1.2.4. Custom attributes

The *custom attributes* are designed to add further information out of the standard attributes that by default are available for any *item*.

The value contained in the custom attributes can be localised like the standard fields.

2.1.3. Language representations: localisation

The **Re3gistry** has two different ways of managing multiple language representations.

- The **Re3gistry software user interface** uses a localisation file to translate the Graphical user interface (GUI) of the management console. This file is not related to the translation of the contents of the registry itself.

*Currently the software package only provides English and Italian translation files for the **Re3gistry** administration panel.*

- The **data processed and produced** by the **Re3gistry** has in turn two ways of handling the multilingualism:
 - The *data localisation*, included in the import data file;
 - The *GUI localisation*, included in the web service's webapp configuration.

By using the examples contained in the package, you should be able to load and export the sample data in all the languages supported by the INSPIRE registry service [INSP-REG].

2.2. The Re3gistry system architecture

2.2.1. Modules overview

The **Re3gistry** handles the entire data flow process for managing register item: from the data import to the export of every item in different formats.

To do so, the system orchestrates a set of modules. There is a `common module`, named Registry core module, implementing the basic features, and additional modules taking care of other functionalities. Figure 2, shows a schematic representation of the system.

More specifically, this package is bundled with:

- the `data import module`, responsible for importing and editing data;
- the `data staticization module`, responsible for saving the data as static files in different formats; and,
- the `deployer module`, responsible for deploying the static files produced to the target production server.

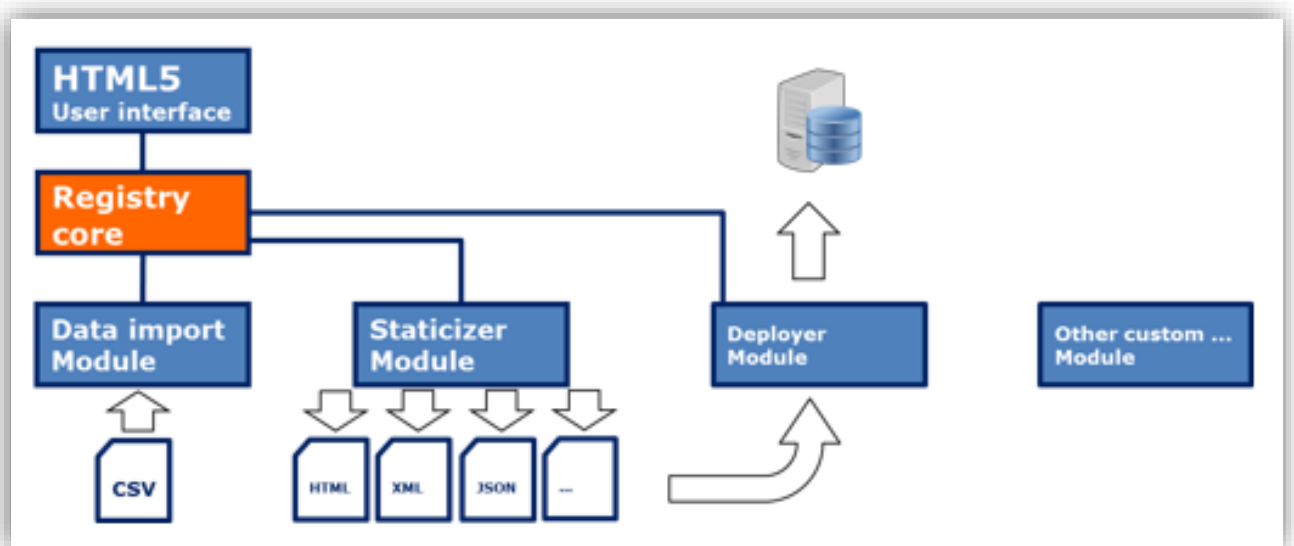


Figure 2: Schematic system description

Figure 3, instead, gives a complete overview of the **Re3gistry** and the rest of components required to provide a *RESTful* web service.

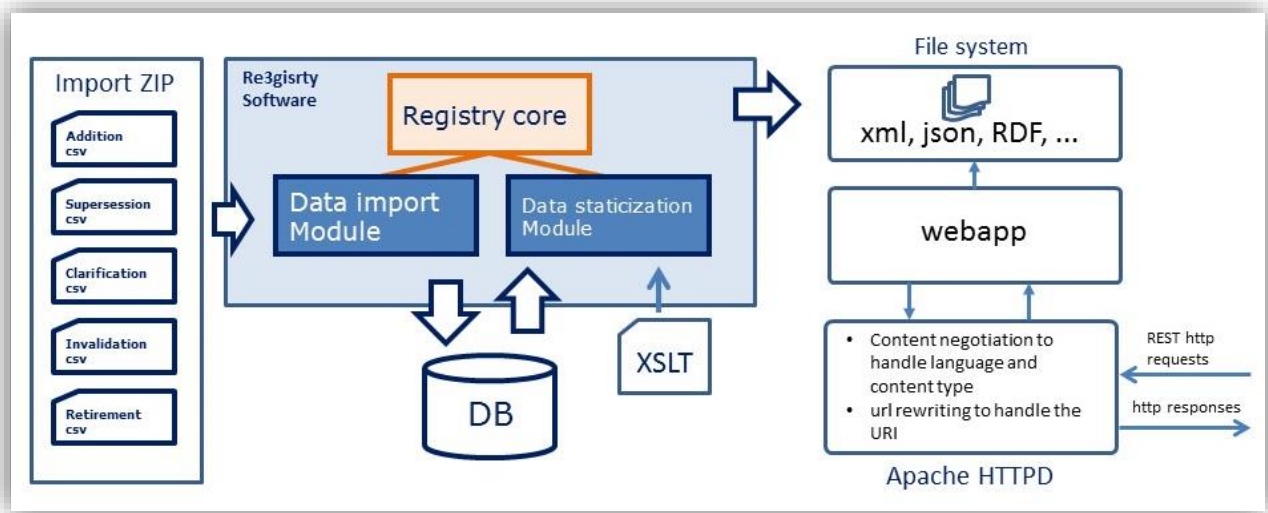


Figure 3: System description

The following paragraphs provide more detailed description of every component involved.

2.2.2. Registry core module

The `registry core` module contains the basic objects of the information model and database operations required by the system.

2.2.3. Data import module

The `data import module` is responsible for the data operations that can be performed on the registers.

The input for this module comes from the import data file (described in detail in section 2.2.3) which is a compressed file in zip format composed of files in CSV format whose name matches with the operations to process per register.

During the import of data, the `data import module` performs different testing to ensure the data consistency within the register.

The `data import module` procedure performs the following actions:

1. **Format check:** checks the correctness of the structure of CSV files, as described in section 2.2.3.1.2
2. **Data analysis:** checks the data semantically by verifying the links between data and its consistency, as described in section 2.2.3.2 Data analyser.
3. **Data storage:** saves in the database the information provided by the CSV files, as described in section 2.2.3.3 Data storage.

An important concept is the ‘operating language’. This property identifies the master language: a language representation that shall be always available.

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

To configure the master language, check how to configure the Re3gistryData.properties file.

2.2.3.1. Import data file

The Re3gistry reads the data to be imported from the data import file (zip file). The zip file structure is described in the following paragraph.

2.2.3.1.1. Data file structure

The root of the zip file contains one folder for each *itemclass* involved (for the definition of *itemclass*, refer to 2.1.1.3 The *itemclass* component).

The name of the folder has to be exactly the same name of the *itemclass*, including the case (the reference to the register is taken through the *itemclass*, which is linked to the register in the information model).

Each of these folders contain those CSV files related to the action to be performed on the specific *itemclass*. The files are named with the name of action in lowercase (See Figure 4).

Figure 4: Example of a zip file structure

```
import.zip
  ApplicationSchema/addition.csv
  ApplicationSchema/clarification.csv
  Theme/supersession.csv
  Theme/retirement.csv
  CodeList/invalidation.csv
```

The zip data files has to contain only those the files and folders required to perform the intended actions. Those files that are not needed should not be part of the zip.

For example, the sample zip data file provided in Figure 4, will perform:

- addition(s) and clarification(s) for the *ApplicationSchema* *itemclass*;
- supersession(s) and retirement(s) for the *Theme* *itemclass*;
- invalidation for the *CodeList* *itemclass*.

2.2.3.1.2. Internal and external items

The Re3gistry is able of storing items as well as referencing externally governed items.

The difference between the two types of items is:

- **Internal items:** these items are defined and governed internally within the registers contained in the registry system. This is the standard type of items. All the information about these items, such as the label, definition, status etc., are stored in the [Re3gistry](#) database.
- **Externally governed items:** these items are defined and governed in an external register. The [Re3gistry](#) can store these items by referencing them. This means that for these items, the only information needed is the URI of the externally governed item. These items can optionally store other information like label, status or other fields. A typical scenario for these items is the register extension. In this case, if a register A is extending a register B, the register A shall contain all of the element of B plus the extended element defined in A. In this case, the items defined in the register B will be referenced using the external item, whereas the extension items defined in A will be internal items.

2.2.3.1.3. Data actions and CSV formats

The system supports five type of actions which matches with the available CSV files for every itemclass.

- Addition
- Clarification
- Supersession
- Invalidation
- Retirement

2.2.3.1.3.1. Addition

This action adds the specified items into the appropriate register.

The CSV file for this action is composed of two different types of fields:

- *Main fields*
- *Additional fields*

The values for the main fields must be provided even if no values are available. If a value is not available, the field will remain blank among the CSV defined separator for the Re3gistry “|”, without any space character (See Figure 5).

Figure 5: Encoding of the addition CSV file when several values are not available

```
value1|||value2|value3|||value4||
```

Moreover, there is a set of mandatory fields required for every type of action.

However, if you wish to change this default behaviour, configure as wished the data module properties file. The property that defines which are the mandatory fields is contained in

```
Path <root_folder>/Project_package_1.X/binaries/Re3gistry-1.X/WEB-INF/classes/configurations/modules/Re3gistryData.properties:  
data.mandatoryfields.<action_name>=<field>.<field>. ...
```

The Figure 6 , shows how the mandatory fields are defined.

Figure 6: configuration of the 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=0.3.3.7
...
List the index of the field as in the property above to set a common field mandatory.
```

The additional fields by us called *custom attributes*, allow adding extra information by the user to fit its needs. The properties of these fields are described in Table 1.

Table 1: Field structure in the addition CSV file

| Type | Code | Description |
|------------------|----------------------|--|
| Mandatory 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'. This field has to be filled in two different way depending on the type of item: <ul style="list-style-type: none"> ▪ The internal items will specify the uriname (e.g. activityCodeValue); ▪ The external items will specify the URL pointing to the external resource. The URI shall be resolvable through an HTTP request (e.g. http://someRegistry/someRegister/activityCodeValue). |
| | ParentLocalId | This is the reference to the parent item |
| | CollectionLocalId | This is the reference to the collection. 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: <code>*custom_attribute_name[required, multivalue,coded,foreignkey]</code> The value in the square parenthesis is a Boolean value, and can be 't' (for true) or 'f' (for false): e.g. <code>*extensibility[t,f,f,f]</code> . The meaning for each of the properties set in the square parenthesis is described in the following rows. |
| | Required | This indicates that the custom attribute 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. The values are separated by comma. E.g. <code>... ab,cd,ef</code> |
| | 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 (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 <i>LocalId</i> of another item in the register. The name of the custom attribute has to be the name of the itemclass referenced. |

Figure 7: Addition CSV example - internal items

```

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 of the item 'sd' in the ApplicationSchema register. There is the mandatory header line and two rows (the 'sd' item in two languages - one row for English language and the other for French).

*In the example in Figure 7, 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.

If the item is externally governed, the only mandatory information to be provided is its URI. The other information are optional

Figure 8: Addition CSV example - external items

```
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
http://someRegistry/someRegister/abc|||||
http://someRegistry/someRegister/def|||en|def||||
```

2.2.3.1.3.2. Clarification

This action allows editing and correcting the register items.

Every time a clarification action is performed, the version of the item is increased.

This action data file, shall be filled only with the fields that need to be updated (apart from the LocalId and the language to identify every item). Only the filled fields will be updated; the field left blank in the action file, will not be updated, and will remain as it 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. Actually, 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 very similar to the addition CSV file but it has not the status field. The clarification CSV structure is described in Table 2 and example of how to fill it instead is provided in Figure 9.

Table 2: 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 itemclass shall be specified in the first row as the addition CSV. |

Figure 9: Clarification CSV file 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 itemclass (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
```

2.2.3.1.3.3. 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(s).

The structure of the supersession CSV is shown in Table 3 and example of its usage is available in Figure 10.

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

Table 3: 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(s). The identifier can be a single identifier or multiple identifier separated by comma (e.g. ac,mf). |

| | |
|----------------------|---|
| | In case of multiple successor elements that are part of a collection, all of the specified successor shall belong to the same CollectionId. If the successor belong from different Collection, one row for each collection shall be specified in the supersede file. |
| 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. |

Figure 10: Example of supersession CSV file

```
Import.zip -> ApplicationSchema/supersession.csv
SupersededLocalId|SupersededCollectionLocalId|NewLocalId|NewCollectionLocalId|Comment
sd||ad||this is a comment
mf||ad,ac||this is a comment
```

This example contains a supersession CSV file for the ApplicationSchema itemclass (linked to the ApplicationSchema register). The file contains two supersessions:

- the 'sd' item, with successor item 'ad'.
- the 'mf' item, with successor items 'ad','ac'.

2.2.3.1.3.4. Invalidation

This action allows invalidating an item. The invalidation CSV file format is almost the same as the supersession CSV file, but there is the 'recursive' flag too (explained in Table 4), because it only requires the reference to the item to be superseded and to the successor item(s). See in Figure 11, an example of its usage.

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

Table 4: 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(s). The identifier can be a single identifier or multiple identifier separated by comma (e.g. ac,mf). In case of multiple successor elements that are part of a collection, all of the specified successor shall belong to the same CollectionId. |

| | |
|----------------------------|---|
| | If the successor belong from different Collection, one row for each collection shall be specified in the supersede file. |
| 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. |

Figure 11: Supersession CSV example

```
Import.zip -> ApplicationSchema/invalidation.csv
LocalId|CollectionLocalId|SuccessorLocalId|SuccessorCollectionLocalId|Recursive|Comment
sd||ad|||this is a comment
so||ac,mf||true|this is a comment
```

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

the 'sd' item, with the successor item 'ad'. This is not recursive and contains also a comment.

the 'so' item, with the successor items 'ac' and 'mf'. This is recursive and contains a comment.

2.2.3.1.3.5. Retirement

This action allows retiring an item. The supersession CSV file format is the simplest one. It only requires the reference to the item to be retired plus a recursive flag (The fields for this action are explained in Table 5 and an example of its usage in Figure 12).

Table 5: 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. |

Figure 12: Retirement CSV file 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 CSV file for the ApplicationSchema
itemclass (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
```

2.2.3.2. Data analyser

The data analyser component performs a number of testing in order to make sure that the data file contains consistent data.

There are two type of messages 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 'ignore warning' flag available in the web interface.

In Table 6, there is a list of the inspections performed by the data analyser.

Table 6: Analyser check and report message type by action

| | Addition | Clarification | Supersession | Invalidation | Retirement |
|--|----------|---------------|--------------|--------------|------------|
| Item already in the database | Error | - | - | - | - |
| Item not available in the database | - | Error | Error | Error | Error |
| Custom attribute (with foreign key flag true) pointing to an item not available in the database nor in the current addition file | Error | Error | - | - | - |
| Custom attribute (with foreign key flag true) pointing to an item not valid | Warning | Warning | - | - | - |
| Item pointing to a parent not available in the database nor in the current addition file | Error | Error | - | - | - |
| Item pointing to a parent not valid | Warning | Warning | - | - | - |

| | | | | | |
|--|---------|---|---------|---------|-------|
| Item pointing to a collection not available in the database nor in the current addition file | Error | - | - | - | - |
| Item pointing to a collection not valid | Warning | - | - | - | - |
| Item pointing to a successor not available in the database nor in the current addition file | - | - | Error | Error | - |
| Item pointing to a successor not valid | - | - | Warning | Warning | - |
| The item's operating language is not available | Error | - | - | - | - |
| The item's language is not in the list of supported language | Error | - | - | - | - |
| The item has collection | - | - | Error | Error | Error |
| The item has child | - | - | Error | Error | Error |
| The item has links (pointed by a custom attribute as foreign key) | - | - | Error | Error | Error |

The system will behave differently according to the situation found:

- If the system finds any errors or warnings, the procedure is stopped
- In case the procedure produces both errors and warnings, the system will lock even if the 'ignore warning' flag is set.

In any of the above mentioned situations and also when the procedure ends successfully, an email notification will be sent to the user. In case of errors or warning, an attachment will be accompanied in the email to help the user fixing the found issues.

2.2.3.3. Data storage

The data storage component saves the data into the database and performs by performing the operations described in the CSV files.

The data storage starts only after the data analyser reports that the data to process is fine.

If something goes wrong during the storage process, the user will receive an email with the detailed list of errors, and the import will be roll-backed.

2.2.4. Staticiser module

2.2.4.1. Staticisation process

The 'Staticisation' component is responsible for exporting into the file system the contents stored in the database through the data management system procedures.

The system uses XSLT transformations files to provide the registry contents in the requested formats.

To do so, the staticiser creates firstly the **master xml files**, containing a structured export of the items stored in the database, that will be used by the different XSLT files, to produce the different files.

The 'custom' folder, will contain all the files produced by the XSLT files. See section on the configuration of the `Re3gistryStaticizer.properties`

2.2.4.2. XSLT

The transformation system based on XSLT is flexible to allow the user customisation by defining the different formats to use and the structure to obtain for them.

The XSLT files translate the information contained in the **master xml files** to the customised file formats. There are three types of master XML files:

- **Registry:** describing the registry
- **Register:** describing a registers
- **Item:** describing the items

The XSLT files shall be contained in a folder specified in the system properties file (`Re3gistryStaticizer.properties` - properties files described at 3.5.2).

Each of the folders used for the XSLT transformation shall contain one XSLT file for each registry, one for each register and one for each itemclass, as illustrated in Figure 13 and Figure 14.

Figure 13: XSLT transformation for XML

```
<itemclass_urname>.<format>.xsl - items  
<register_urname>.<format>.xsl - register  
<registry_urname>.<format>.xsl - registry
```

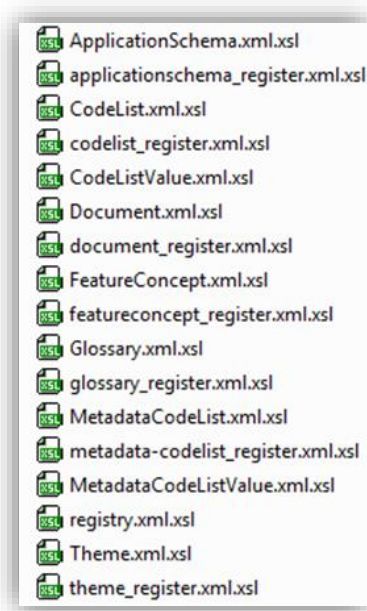



Figure 14: XSLT transformations files for the custom XML format (contained in the xml folder)

2.2.4.3. Static element localisation

To handle the localisation of some static elements, such as some common parts of the formats, there is a *language mapping file*. The 'gui-languages' folder contains `<language_code>.xml` files, one for each language supported by the system, containing the *item keys* (See Figure 15) used by the XSLT files for any static string not available in the database (See Figure 16).

Figure 15: 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>
```

Figure 16: Usage example. Place this piece of code to retrieve the translated word in the file.

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

The `$languagefile` variable is the path of the document containing the translations and it is defined at the beginning of the xsl file. An example is provided below:

```
<xsl:variable name="language" select="item/language/isocode"/>
```

```
<xsl:variable name="languagefile" select="document(concat('../gui-languages/', $language, '.xml'))" />
```

2.2.4.4. Additional information

The staticiser is responsible as well for the creation of additional files that handle the content-negotiation (`var` files) and the Apache solr indexing.

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 5.2.1 RESTful web service .

*The **solr export** will produce the files to be imported by the Apache solr application (one file for each item, containing all the languages). The solr XSLT transformation produces files compliant with the Apache solr system.*

2.2.5. Deployer module

The `deployer module` is responsible for deploying the static files produced by the staticization system (See 2.2.4 Staticiser module) to the target production server.

This is needed if the production server is in a different machine, or if the files in the same machine need to be moved to another place in the same system.

The module allows to automatically take the set of static files produced and move them to the configured target place.

To see how to configure the deployer module refer to section 3.5.2 Modifying the configuration files.

2.3. Re3gistry administration panel

Since version 1.0, the [Re3gistry](#) provides a simplified interface, the '[Re3gistry administration panel](#)', to allow the user easily managing the registry contents, without the need of understanding the underlying complexity of modules interconnection.

To understand in which step of the procedure the system is , there is a list of procedure status that express the progress reached by the [Re3gistry](#) when handling the imported data. In Table 7, are shown the different possible values for the procedure status.

Table 7: Procedure statuses

| | |
|---------------|---|
| Checking data | The system is checking the data file format and analysing the data file consistency. |
| Storing data | The data has been checked; the system is storing the data to the database |
| Writing files | The data has been stored to the database; the system is writing the static file to the file system |
| Completed | The files has been written to the file system and they are ready for the deployment |
| Imported | This status indicates that the data import has successfully completed but the export to the static files has not been done (or the export went wrong). In this case, the export can be started manually with the specific button in the UI. |
| Deployed | The file has been deployed to the web server |
| Failed | Something went wrong during one of the data procedure step. In this case, a detailed error description is sent to the user's mail |

To understand how to use step-by-step the administration panel, see section 4.1 Accessing the [Re3gistry administration panel](#).

3. Installing the Re3gistry

This section will guide the user installing *Re3gistry* and using it for the first time through ready-to-use examples.

3.1. System requirements

To install the *Re3gistry* components the following programs need to be previously installed in the user's computer.

- Java SE Development Kit (JDK) 7 or higher⁶
- Apache Tomcat 7 or higher⁷
- PostgreSQL 9.2 or higher⁸
- ECAS for Tomcat (provided within the package) [optional - to be used only if the ECAS authentication method is selected].

3.2. Package details

The software package, available for its download in the Re3gistry space of JoinUp⁹, includes: binary files (ready to use application), source files, configuration examples and other required files.

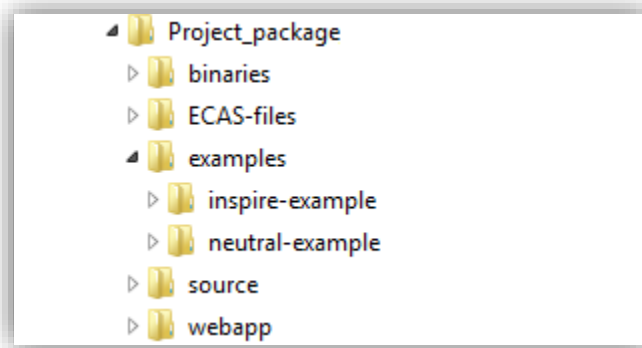


Figure 17: Structure of the Re3gistry package

The structure of the package folder is as follows:

- **binaries**: containing the *Re3gistry software* binary files (ready to use application).

⁶ <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

⁷ <http://tomcat.apache.org/>

⁸ <https://www.postgresql.org/download/>

⁹ <https://joinup.ec.europa.eu/software/re3gistry/release/all>

- `ECAS-files`: containing the ECAS authentication libraries and other required files. These files will be used only if the ECAS authentication method is chosen during the Re3gistry configuration.
- `examples`: containing the example files needed to install, initialise and run the *Re3gistry software* as well as to create an example web service instance. Two full examples are provided within the folder:
 - `inspire-example`: containing a sample of the contents available in the INSPIRE registry service,
 - `neutral-example`: containing more generic registers related to countries and companies.

To know more about the examples offered, refer to the `readme.md` file inside the example's folder.

- `source`: containing the source files of the *Re3gistry software*.
- `webapp`: containing the *PHP* web application that can be used to serve data files produced by the Re3gistry software.

3.3. Important notes

The files contained in the `examples` folder, encloses a couple of ready-to-use examples to help the user quickly setting up and running its *Re3gistry* instance. To proceed with the installation explained next, the user will need to select one of them and make sure he always takes the configuration files relative to the example chosen, otherwise the installation will not work.

3.4. Database configuration

The database initialisation script will create the database structure and it will populate its tables with the registry information and its registers contents. The database example scripts are available within each of the different example files provided in the `examples` folder.

3.4.1. Creating a new database

Before executing the scripts, you need to create a new database in *PostgreSQL* that will store the registry contents.

3.4.2. Running the SQL scripts to create tables and populating them

Select one of the examples provided and follow the path until reaching its `database` subfolder:

Path: `<root_folder>/Project_package_1.X/examples/<inspire-example|neutral-example>/database`

The scripts contained in the 'database' subfolder are four and they need to be executed in the following order:

- 1 `create-table.sql`
- 2 `database-initialization.sql`
- 3 `database-localization.sql`

The `drop-table.sql` will be solely used if there is a need of resetting the database, by removing the whole tables with its contents. Only if you are working on an existing database that needs to be cleaned up, it will need to be executed in first place.

These scripts can be executed either by using the command line or simply by using a graphic user interface (GUI) such as *pgAdminIII* usually included with the *PostgreSQL* installation.

To run the script from *pgAdmin*, make sure you are connected to the target database, open the 'Query tool' (in the menu go to 'Tools' > 'Query tool' or Ctrl-E).

From the 'Query tool' window, Use the 'Open file' button to select the respective SQL file to run, and execute it by clicking on the 'Execute query' button. (In the menu go to 'Query' > 'Execute' or F5). Once the script has been processed, continue by launching the following script (See Figure 18).

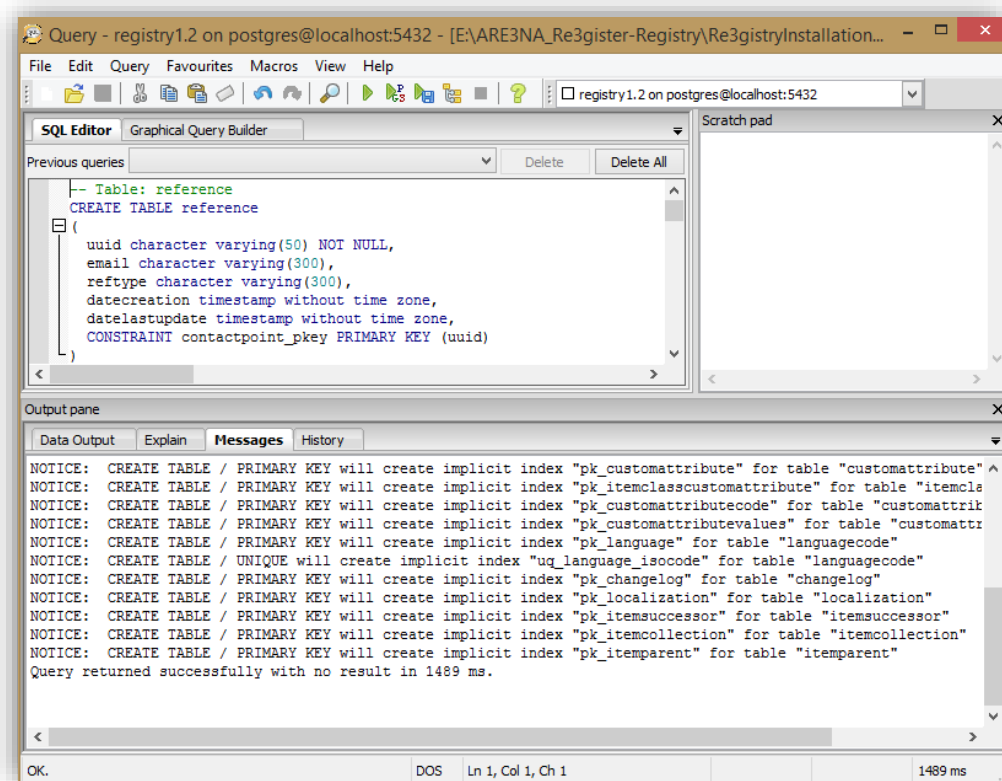


Figure 18: Executing the `create-tables.sql` script from *pgAdminIII*

Once the three scripts have been launched you should have in the recently created database 19 tables already populated (See Figure 19).

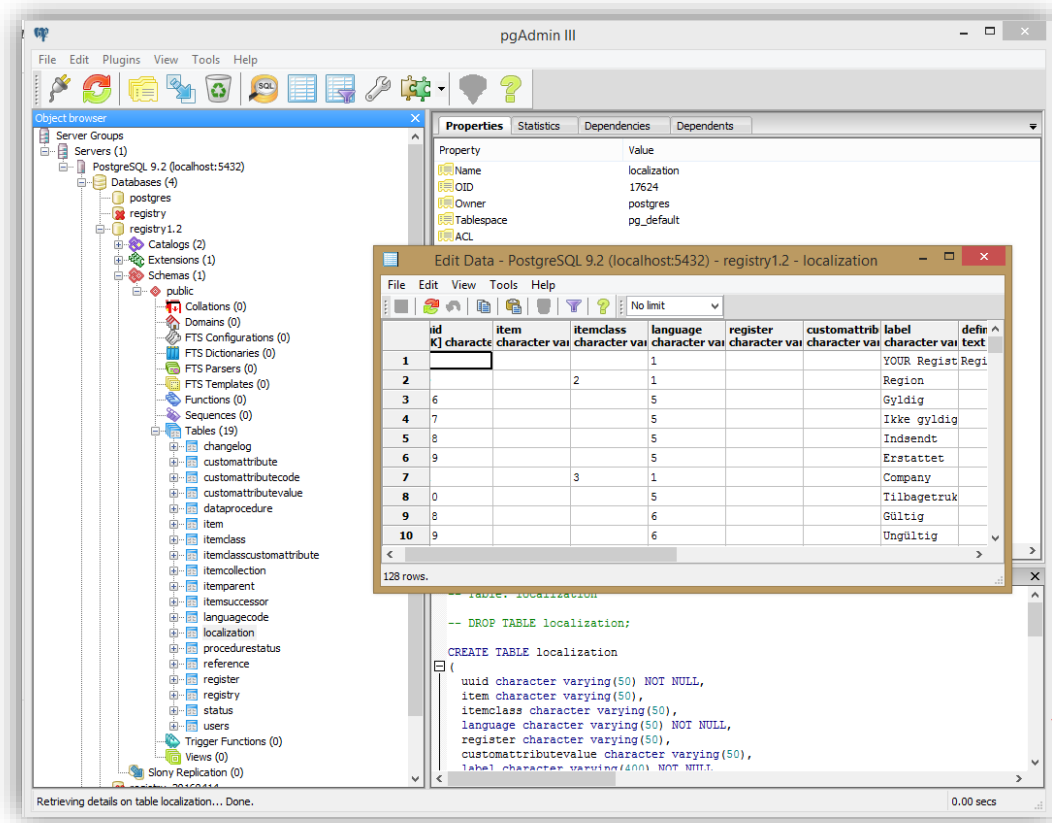


Figure 19: Nineteen tables created and populated

To run them through the shell, follow the following example.

```
psql -f <example-file.sql> <targetdatabase>
```

This command may need additional parameters, depending on the system configuration (e.g. credentials, host ...).

3.5. Configuring the Re3gistry

This section explains how to install and get started with **Re3gistry** software, that is, the part of the package that produces and manages the contents. Those contents can optionally be served through the proposed webapp. To know more on that, please refer to section 0 on

Serving the Re3gistry contents.

3.5.1. Move the binaries folder to Tomcat's webapp folder

Go to the Re3gistry package, inside the 'binaries' folder select and copy the 'Re3gistry-1.X' subfolder and paste it within the Apache Tomcat's 'webapps' folder.

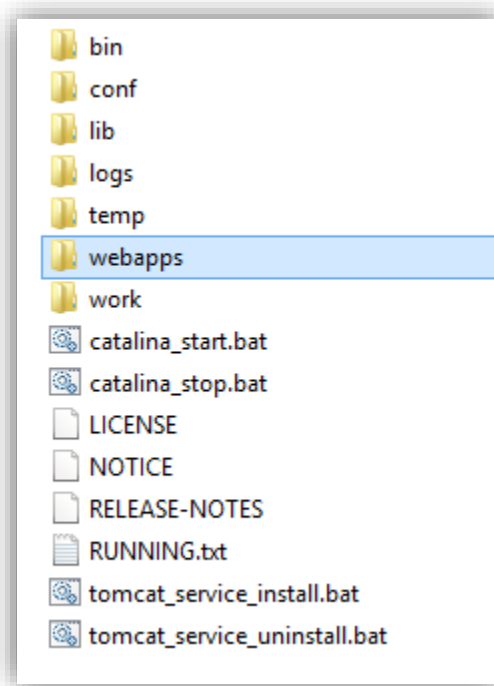


Figure 20: Usual Tomcat installation file structure

Before moving the project's folder, make sure that the Apache Tomcat server is not running.

To stop it, there is a 'shutdown' script in the tomcat's 'bin' folder (depending on your operative system: Windows user, shutdown.bat; Linux users, shutdown.sh).

3.5.2. Modifying the configuration files

Once the binary files have been located in Tomcat, several configurations files need to be modified by the user to set up its **Re3gistry** instance. More specifically, a set of properties must to be updated accordingly to the user's environment and set to fit the user's preferences.

The files to modify are the following:

- Application.properties
- logcfg.xml
- RegistryData.properties
- RegistryStaticizer.properties
- RegistryDeployer.properties

Note: The properties to be changed are marked with the placeholder '`<propertyname>_`', that is they appear among underscore symbols. Usually, tips on how to modify the settings are given directly in the configuration files in the form of comments.

Some examples are illustrating the placeholders are:

`_DBADDRESS_`
`_USERNAME_`.

3.5.2.1. persistence.xml

Path : `<root_Tomcat_folder>/webapps/Re3gistry-1.X/WEB-INF/classes/META-INF`

This file provides the configuration to connect to the database. The parameters to modify are:

| | |
|--|--|
| <code>javax.persistence.jdbc.url</code> | URL of the database including port number and target database name (e.g. " <code>jdbc:postgresql://localhost:5432/registry</code> "). |
| <code>javax.persistence.jdbc.user</code> | User of the database. |
| <code>javax.persistence.jdbc.password</code> | Password of the database. |

3.5.2.2. Application.properties

Path : `<root_Tomcat_folder>/webapps/Re3gistry-1.X/WEB-INF/classes /configurations`

This file manages the general properties of the system as the contact details, the language preferences or the authentication methods.

To know more on how to configure the authentication methods, please go to section 3.5.3 Setting up the authentication method

The parameters to modify are:

| | |
|--|--|
| <code>application.language.available</code> | The available language for the interface, specified the by the 2 letter ISO 639-1 language code. If more than one, separate them with a hyphen symbol (-). |
| <code>application.language.available._xx_</code> | If different languages are available there should be an entry for each one followed by the value of the label that will appear in the language selector |
| <code>application.contact</code> | The e-mail address of the system's contact. |
| <code>mail.sender</code> | The e-mail address of the sender for the system's emails. |
| <code>mail.recipient</code> | The e-mail address for the default recipient. |
| <code>mail.smtp.host</code> | The server host SMTP email. |

3.5.2.3. logcfg.xml

Path : <root_Tomcat_folder>/webapps/Re3gistry-1.X/WEB-INF/classes/configurations

This file contains the settings regarding the logging system.

Log files are useful to monitor the activity of the system and particularly to discover potential errors or warnings and its causes.

The user must update every ‘_LOGPATH_’ placeholder with the path where he wants the log files to be stored. More specifically, there are five types of log files that the system produces. (The procedures in charge of producing them are known as ‘log appender’)

- ‘Re3gistry’ appender produces the Re3gistry.log file
- ‘Re3gistryData’ appender produces Re3gistryData.log file
- ‘Re3gistryStaticizer’ appender produces Re3gistryStaticizer.log file
- ‘Re3gistryDeployer’ appender produces Re3gistryDeployer.log file
- Finally, the ‘Default’ appender produces the Complete.log file. It contains as the name indicates, the full logging information coming from the rest of the log files.

3.5.2.4. Re3gistryData.properties

Path : <root_Tomcat_folder>/webapps/Re3gistry-1.X/WEB-INF/classes/configurations/modules

This file contains the settings regarding the data management. The properties to update are:

| | |
|------------------------|--|
| data.customtempfolder | The folder used to save the temporary data files. If this property is left blank, a default folder will be created directly within Tomcat’s webapp folder. |
| data.operatinglanguage | Code of the ‘master’ language of the data contents according to the 2-letter ISO 639-1 language code. |
| data.supportedlanguage | Code(s) representing the data languages supported and handled by the system according to the 2-letter ISO 639-1 language code |

Note: *If you plan to represent the registry contents in different languages, make sure they all appear in the data.supportedlanguage property, otherwise you might find errors when trying to import data.*

3.5.2.5. Re3gistryStaticizer.properties

Path : <root_Tomcat_folder>/webapps /Re3gistry-1.X/WEB-INF/classes/configurations/modules

This file contains the configurations to allow the system producing the static files from the data available in the database for every requested format. In addition, this file contains the settings to produce solr index data that can optionally be used to power a search engine.

The main properties to be updated here are:

| | |
|--|--|
| <code>staticizer.custom.folder.path</code> | Folder where the <i>staticised</i> and converted files will be saved. |
| <code>xml.formats.list</code> | List of formats that will be handled by the system indicated through the extension names (separated by comma if more than one). |
| <code>staticizer.formats.path</code> | Folder containing XSLT transformation files to process the data conversion in multiple formats. The user must make sure that in the given path there are as many folders as formats to be produced accordingly to the values set in the <code>xml.formats.list</code> property. The folders must be named exactly as the indicated labels in the property <code>xml.formats.list</code> and they must contain the XSLT files with the transformation rules. |
| <code>staticizer.solr.format</code> | Folder name where the solr data file will be saved |

Note: to help the user using and running for the first time the Re3gistry we suggest to copy the XSL folder of any of the examples coming in the software package `/examples/<inspire-example|neutral-example>/xsl` folder and pasting it into the XSL folder created.

3.5.2.6. Re3gistryDeployer.properties [Optional]

Path : `<root_Tomcat_folder>/webapps /Re3gistry-1.X/WEB-INF/classes/configurations/modules`

This configuration file facilitates the automatic creation of an *RSS* file showing the changes occurred in the registry contents and additionally it simplifies file management tasks, moving programmatically files across directories.

The main properties to modify are:

| | |
|--------------------------------------|--|
| <code>deploy.script.folder</code> | Folder containing the deploy script. By default, two template scripts (to be launched either on Windows or On Linux environments) are already included within the folder <code>webapps/Re3gistry-1.X/WEB-INF/classes</code> . This property should be updated if the user decides to locate the script in another path. Moreover, the scripts must be edited accordingly to the user's environment. |
| <code>deploy.rss.file</code> | Path to the RSS template file used by deployment procedure. The <i>news.en.xml</i> file, available in any of the given examples folder within the <i>deployer</i> subfolder, offers the base schema to produce the real RSS . |
| <code>deploy.rss.targetfolder</code> | Folder where the produced RSS will be stored. |

deploy.rss.baseuri

URI to be used by the deployment's procedure when producing the RSS file. Normally the base URI corresponds to registry's homepage URL (e.g. *http://localhost/registry*)

3.5.3. Setting up the authentication method

3.5.3.1. Available authentication methods

The **Re3gistry** allows choosing among two authentication methods: *Apache SHIRO* and the European Commission Authentication Service website known as *ECAS*.

- The *Apache SHIRO* is suitable for any environment. SHIRO allows the connection of different authentication method. The **Re3gistry** has implemented the 'static' user authentication, which is managed by a simple configuration file containing the full list of users.
- The ECAS authentication method instead, can only be used either in domains trusted by the ECAS administrators (that is in European Commission domains) or in 'localhost'.

To learn how to attach other authentication method than the 'static' one, read the own Apache SHIRO documentation.

3.5.3.2. Choosing and implementing the authentication method

The authentication method is handled by two configuration files.

3.5.3.2.1. Application.properties

Path : <root_Tomcat_folder>/webapps /Re3gistry-1.X/WEB-INF/classes
/configurations

The `Application.properties` file includes a parameter that allows switching between `ECAS` and `SHIRO`. By default, the authentication method is set to `Apache SHIRO`. (See Figure 21)

Figure 21: Default authentication method

```
# Login type: SHIRO | ECAS  
application.LoginType=SHIRO
```

To change of authentication method, just modify the value with the proper string as indicated in Figure 23.

Note: If ECAS method is chosen, special Tomcat libraries have to be installed.

3.5.3.2.2. web.xml

Path : <root_Tomcat_folder>/webapps /Re3gistry-1.X/WEB-INF/

The `web.xml` file handles among other things, the authentication method settings. It must be modified accordingly to the authentication method previously chosen in the `Application.properties` file. By default, SHIRO method is set, to change to ECAS, the SHIRO related lines must be commented while the ECAS uncommented.

To help the user in this step, some informative comments in the `web.xml` file have been added to highlight when SHIRO and ECAS related configurations lines begin and end.

For example: To go for ECAS method, the text contained between the following lines `<!--SHIRO authentication configs-->` and `<!--END SHIRO authentication configs-->` should be commented, that is enclosed between `'<!--'` and `'-->'` symbols.

3.5.4. Adding users to SHIRO

Path : `<root_Tomcat_folder>/webapps /Re3gistry-1.X/WEB-INF/shiro.ini`

The **Re3gistry** implements SHIRO through the *Static File* authentication mode. This file, named `shiro.ini` contains the user(s) that are allowed to access the system together with their passwords and roles.

To add or modify a user, it is enough to go to the `[users]` section of the file, and add or modify the line with its username, password and role type information.

The user name can be either a name or an email address (See Figure 22).

Figure 22: Shiro.ini users section

```
[users]
admin@example.eu = admin, ROLE_ADMIN
user1@example.eu = password1, ROLE_ADMIN
```

To use the HTTPS protocol update the `/login` property in the section `[urls]` as follows:

```
/login = ssl[8443],authc
```

If the SSL configuration is chosen, make sure Tomcat is properly configured to support HTTPS.

4. Using the Re3gistry

4.1. Accessing the Re3gistry administration panel

Once the user has completed the installation steps covered in section 3, he just needs to start the *Tomcat Web Application Container* and access the **Re3gistry** through any web browser.

There are different ways of starting Tomcat, depending on the installation, on the O.S. and other configurations. Usually the tomcat can be easily started using the 'startup' (Windows users: startup.bat, Linux users: startup.sh) script available in the Tomcat's 'bin' (<tomcat installation folder>/apache-tomcat.8.x.xx/bin/).

The URL to access the instance usually, if not configured diversely, should follow this pattern:

```
http://localhost:[tomcat_port]/[Name_Of_Webapp]
```

By default, Tomcat works on port 8080, so in common installations, and if the webapp name has been maintained as 'Re3gistry-1.X' (folder name appearing within the TOMCAT webapps folder), typing http://localhost:8080/Re3gistry-1.X should redirect to the authentication page.

Accessing the **Re3gistry**'s URL should take you to a web page asking for authentication details (See Figure 23).

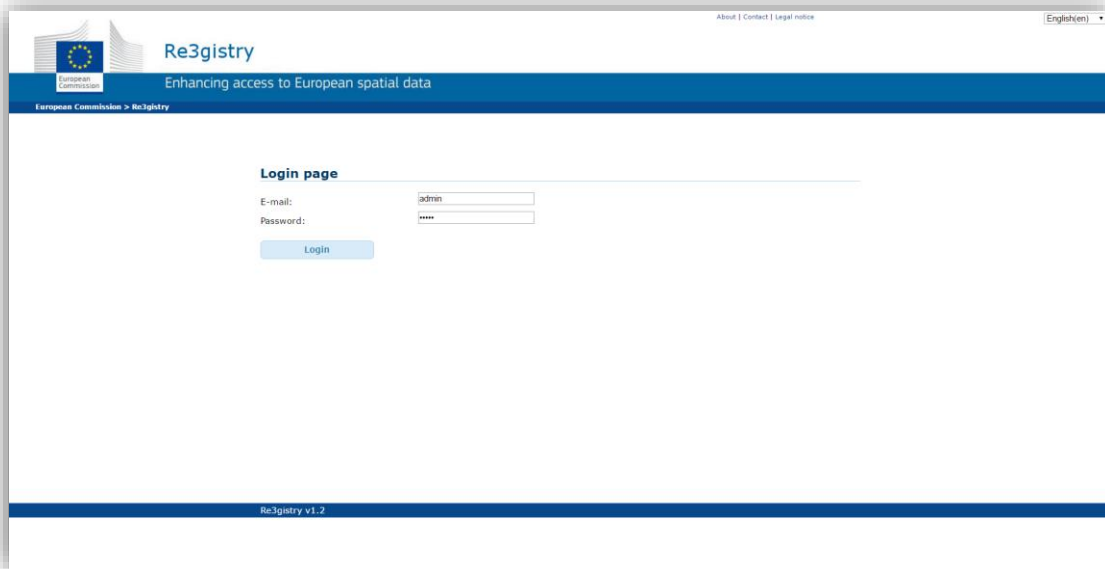


Figure 23: Authentication page to access the Re3gistry software administration panel

Entering the user authentication details previously set in the `shiro.ini` file should give the user access to the **Re3gistry** `Data management system` page (See Figure 24)

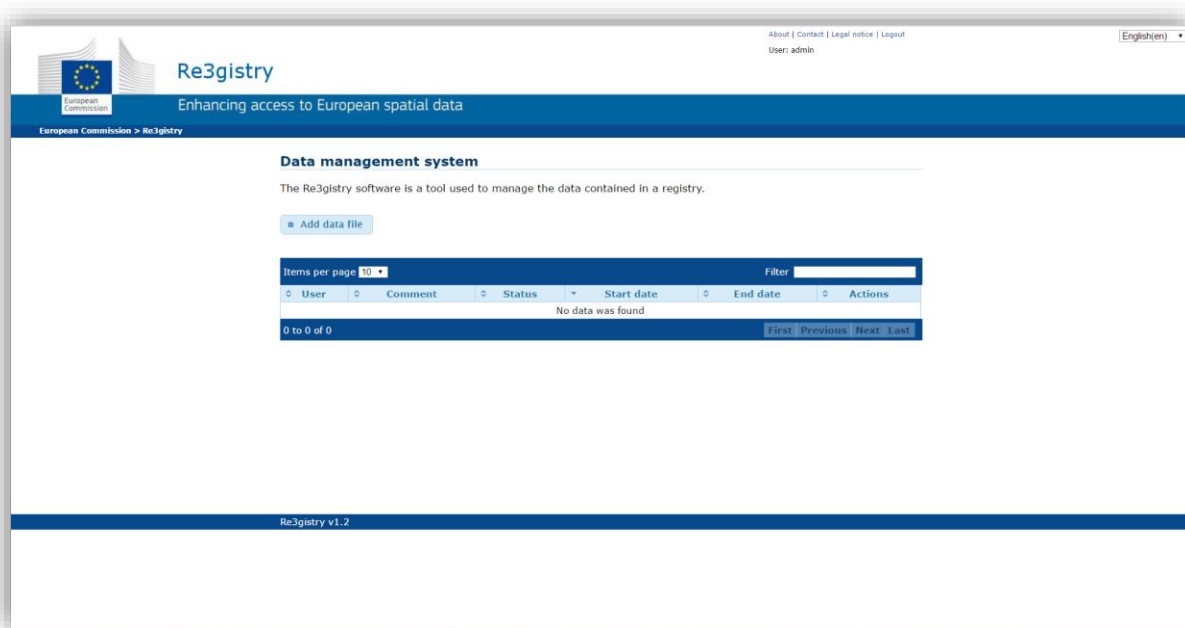


Figure 24: Data management system page

4.2. Importing data

To start the import procedure, click on the 'Add data file' button, a pop window will appear to guide you in the data file 'importing' phase (See Figure 25).

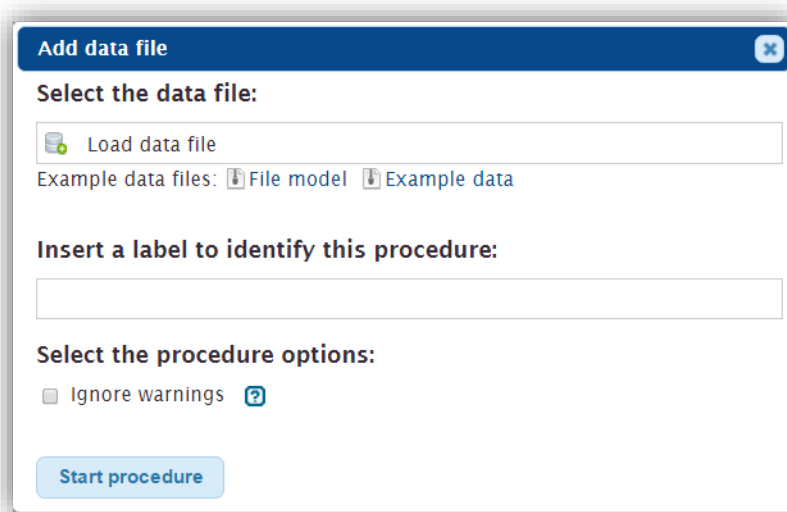


Figure 25: 'Add data file' window

Pressing the 'Load data file' option, will let you browse your computer' directories to pick the target data folder.

Next, it is recommendable to write a brief comment describing the data that is on the point of being imported, that will allow you easily identify the procedure launched in the case you run others, and finally press 'Start procedure'.

Optionally, you can check the 'ignore warning' box in case you do not want the system to check the correctness of the source data structure before inserting it in the database. If instead, you prefer to validate the correctness of your data leave the field unchecked.

Refer to the section 2.2.3.2 on Data analyser to know the differences between *errors* and *warnings*.

The data files to be loaded must be compressed in.zip format.

If you want to load an example file, go to the downloaded software package inside the data folder relative to your chosen example.

Path: <root_folder>/Project_package_1.X/examples/<inspire-example|neutral-example>/data

Remember, if in the Database creation step you chose 'neutral-example', you should be taking the file NeutralExampleV1.X.zip. If you want to change of example, you must clean the database content and load its proper script (See section 3.4.2).

After starting the procedure, a new row will appear in in the data management system's table with the relative details to the launched procedure (See Figure 26).

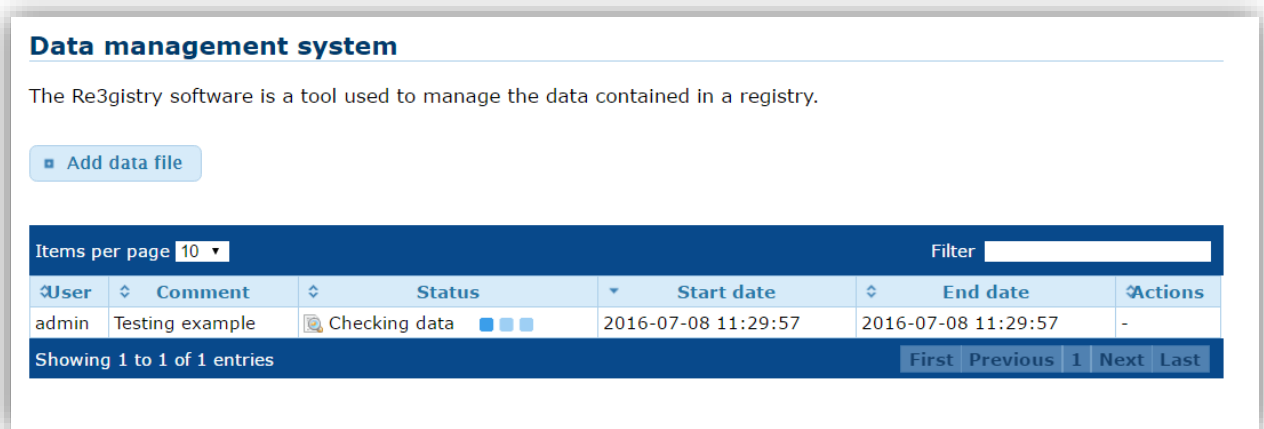


Figure 26: Procedure details in the data management system panel

The `status` column in the table will change accordingly to the task being performed

This table contains the history of the procedures launched, together with the current running procedure and eventual queued procedure. Once the procedure starts, the systems performs all the data operations (import and export). See section 0

Re3gistry administration panel, to know the meaning of the possible status values.

Only one procedure can run per time. If the user adds a new data file while another procedure is running, the newly added data file will be queued. The system will run the queued files once the current running procedure has ended.

Once the procedure has started, 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 problems, the list of problems will be attached to the email.

The procedure may take a considerable amount of time, depending on the number of items to be processed.

When finished, if no error has been encountered, the value of the 'Status' column should now state 'Imported' and the option 'Run Data Export' in the last column ('Action' column) must be available.

4.3. Exporting and converting data files

In case you want to run again the staticization process, you can use the 'Run Data Export' function. It will start exporting the information stored in the database in the requested formats (See Re3gistryData.properties) without the need to re-import it. When the Status column value passes from 'Writing files' to 'Completed,' it means that the static files have been properly produced.

Note: Remember that the conversion across formats relies on XSLT files. If the user would like additional formats, he will need to provide the needed transformations rules.

To check the output files, go to the path defined in the property 'staticizer.custom.folder.path=' of the Re3gistryStaticizer.properties in section 3.5.2). Normally, if the name folder has not been altered, the files should be contained inside the custom subfolder.

If you need the export of the whole data in the database, you can use the option 'Run full export'

If you chose to export the 'neutral-example', you should have obtained a folder with a file structure as shown in Figure 27.

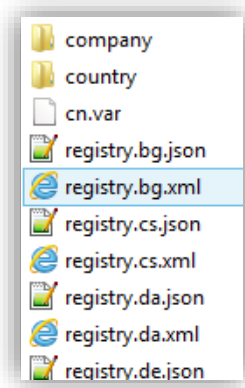


Figure 27: Produced data in the 'custom' folder for the 'neutral-example'

In general terms, the output files are organised in a way that every register has its own folder (matching with the register name) with its belonging items enclosed in it. In the root of the custom folder the registry description appears.

There could be multiple files related to a single item because of the different formats and languages required. This can be identified `[item_name].[language].[format]`

4.4. Deploying the contents

4.4.1. Moving data from the Re3gistry software to the server

To make the user live easier, there is a script in the

```
webapps/Re3gistry-1.X/WEB-INF/classes path
```

that helps the user managing files across system. Namely, it allows moving programmatically the recently 'exported' data to the webserver folders serving the contents coming from the 'custom' folder and using the indexed data too, coming from 'solr' folder.

To make it work properly, the user should open it and edit the file by defining the proper source and target folders to be handled (See Figure 28).

```
1  :: Windows deploy script ::
2  xcopy "C:/Re3gistry-data/staticizer/custom" "C:\\Program Files (x86)\\Apache Software Foundation\\Apache2.2\\htdocs\\custom" /e /i /u /y
3  xcopy "C:/Re3gistry-data/staticizer/solrcustom" "C:\\Program Files (x86)\\Apache Software Foundation\\Apache2.2\\htdocs\\solr" /e /i /u /y
```

Figure 28: Windows-deploy.bat contents

Note: To know more on how to configure the server continue reading the coming sections.

4.4.2. Creating a modification summary RSS feed

RSS feeds are useful to share the modifications occurred in the contents of the registry and keep your audience informed of them.

The **Re3gistry** administration panel, has a 'Deploy' button that allows you to automatically create or update (if already done) an RSS file through a PopUp Window (See Figure 29).

Note: To perform this action you must modify the properties in the `Re3gistryDeployer.properties [Optional]` file.

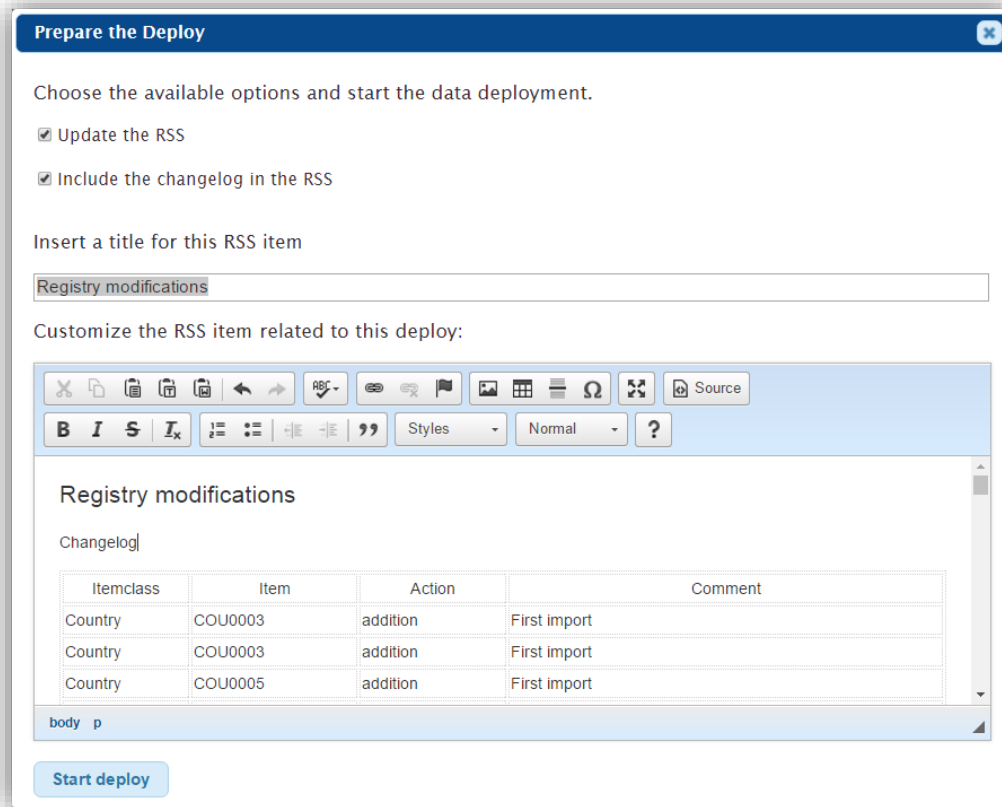


Figure 29: PopUp window allowing creating or updating an RSS file

To check the output file, go to the path defined in the property `\deploy.rss.targetfolder=` of the `Re3gistryDeployer.properties` (See `Re3gistryDeployer.properties`).

5. Serving the Re3gistry contents

5.1. System requirements

To install the webapp included in the [Re3gistry](#), the following programs need to be previously installed in the user's computer.

- Apache HTTPD¹⁰
- PHP 5.4 or higher¹¹
- Apache solr 4.8.0¹²

Once the **step** 4.3 Exporting and converting data files has been performed, the content is ready to be served and shared.

The way the content will be shared, is up to the user. Nevertheless, the [Re3gistry](#) package includes a ready to use web application (webapp) that could help the user providing a friendly user interface to interact with the data.

The [Re3gistry](#), produces a set of files organised to set up a web service providing access to the registers. 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 content-negotiation means that the server can provide automatically the correct file language/format based on the parameters set in the http header which is sent in the http request.

If a user prefers to use the classic way to accessing a specific file (request a specific format/language using a direct URL), the system supports this approach too.

The file to be used in order to set up a web service can be found in the staticiser output folder as configured in the file named Re3gistryStaticizer.properties .

5.2. Web service

The files produced by the [Re3gistry](#) can be used to provide the web service in two ways:

- a RESTful web service which adopts a content-negotiation approach for serving the resource or,

¹⁰ <https://httpd.apache.org/download.cgi>

¹¹ <http://php.net/downloads.php>

¹² <http://www.apache.org/dyn/closer.lua/lucene/solr/6.1.0>

- a standard web service, which uses the resource name in order to access the specific resources.

5.2.1.RESTful web service

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. In Table 8 there are some examples illustrating how to access different resources in [Re3gistry](#) webapp.

Table 8: RESTful URL example

| URL | Use |
|---|--|
| http://base_uri/register_username | Used to access a register, an item or an item collection |
| http://base_uri/register_username/item_username | |
| http://base_uri/register_username/collection_username/item_username | |
| 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 http header set. If no header is set, the web service returns (by default) the xml format in English.

There are different ways of setting the *HTTP header* in the requests. If the request to a resource is done through the browser, the *HTTP header* can be set using a specific plugin for that browser. Look at Figure 30 to see how the parameters can be set.

Search for 'HTTP header' in the browser's component store. If the request is done programmatically, refer to the guide of the programming language used.

Figure 30: Example of content-negotiation parameters

| | |
|-----------------|------------------|
| Accept | application/xml |
| Accept-Language | en |
| Accept | application/atom |
| Accept-Language | fr |

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 file with a `.var` extension (for example, `cn.var`), see Figure 31 to understand how it looks like.

The staticiser module produces this file automatically.

Figure 31: `.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 the *Apache HTTPD guide [APACHE-CN]*.

5.2.2. Standard web service

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

Figure 32: Web service requests - 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
```

5.3. Installing the Re3gistry webapp

5.3.1. Copy the sample web application folder

Browse the [Re3gistry](#) software package and copy the **webapp** folder available at:

```
Path: <root_folder>/Project_package_1.X/webapp
```

Paste the copied contents into the folder storing the *exported* data, as shown in Figure 33.

Remember this path given to the 'staticizer.custom.folder.path=' of the Re3gistryStaticizer.properties configuration file

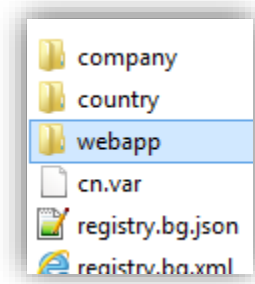


Figure 33: Paste the webapp folder into the exported data folder

5.3.2. Setting the web application

The webapp is built in php programming language, for this reason, your server should have installed php v5.4 or higher.

The webapp folder shall be put in the same folder containing the files exported from the [Re3gistry](#) during the exporting phase. See how in Figure 34.

To make the HTML webapp work, at least the JSON export of the data managed by the [Re3gistry](#) is needed. An example file system structure is shown below.

Figure 34: Proposed structure to locate the webapp files and the data files

```
/var/www/your_app/data/ -> This folder contains all of the files produced by  
the Re3gistry software.  
/var/www/your_app/data/webapp -> This folder contains the webapp.
```

5.3.3. Configuration

The webapp needs as well a couple of files to be modified and included in the webapp folder. Those are:

- conf.php
- logger.xml

5.3.3.1. conf.php

Path : /webapp/app_data/

This file manages the basic configuration of the web application: from the web application URL itself, to the languages available, to the number of items to be showed per page.

With the help of a notepad++ program, open the file to update the properties according to your needs and save them as shown in Figure 36.

The important properties to be configured are listed and explained in Figure 35 :

Figure 35: Properties in the conf.php file


```
define('APPLICATION_ROOT','/var/www/INSPIRERGD/data/webapp/'); // This is
the path of the root folder of the webapp.

define('APPLICATION_ROOT_URL','http://inspire.ec.europa.eu/registry/'); //
This is the root URL of the registry webapp.

define('CDN_URL','http://inspire.ec.europa.eu/cdn/latest/'); // This is the
URL of the CDN containing all of the style and script for the website.
Currently we provide the INSPIRE CND as an example included in the package.
You can start from that to customize your User Interface.

The other important properties are located at the end of the config.php file
after the line "/* Service configs */". Here you have to basically
change the URL of each properties according to your needs. For example, if
your URL starts with http://www.example.com, you have to replace all of
the http://inspire.ec.europa.eu strings with http://www.example.com. The
reason to have one properties for each element is that in some advanced
installation, there can be different URL for each of the think in the
system.
```

Detailed information in form of comments will help the user configuring the webapp to his environment.



```
1  <?
2  /* Application configuration file */
3
4  /* Application main configs */
5
6  /* Webapp paths */
7  define('APPLICATION_ROOT','C:/xampp/tomcat/webapps/Re3gistry-1.2/DataRepository/StaticisedData/custom/webapp');
//The root path of the webapp. (Example: /var/www/registry/webapp/ or C:/wwwroot/registry/webapp/)
8
9  /* Webapp URLs */
10 define('REGISTRY_BASE_URL','http://localhost/registry/'); // The root URL of the registry service. (Example:
http://localhost/registry/)
11 define('APPLICATION_ROOT_URL','http://localhost/webapp/'); // The root URL of the application. (Example:
http://localhost/webapp/)
```

Figure 36: conf.php file opened in Notepad++

5.3.3.2. logger.xml

Path : /webapp/app_data/

This file contains the settings of the logging system. The property to define is called `'_LOGPATH_'`. This text will be replaced with the path to the folder where the logs of the web application will be saved (See Figure 37).

You can use the `log` folder used formerly in step 3.5.2.3 `logcfg.xml`, or just create a new folder to better distinguish the logs files.



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <configuration xmlns="http://logging.apache.org/log4jphp/">
3   <appender name="default" class="LoggerAppenderRollingFile">
4     <layout class="LoggerLayoutPattern">
5       <param name="conversionPattern" value="%date [%pid] %server{HTTP_HOST}%server{REDIRECT_URL} - %5level - %msg%n" />
6     </layout>
7     <param name="file" value="C:/xampp/tomcat/logs/WebappLogs/application.log" />
8     <param name="maxFileSize" value="10MB" />
9     <param name="maxBackupIndex" value="10" />
10  </appender>
11 <root>
12   <appender_ref ref="default" />
13 </root>
14 </configuration>
```

Figure 37: Property in the `logger.xml` configuration file where the log file is defined

5.3.4. Configuring the HTTP server

Path: `<root_folder>/Project_package_1.X/webapp/examples/<inspire-example|neutral-example>/apache-configurations/service-configuration.conf`

The HTTP configuration files included in the examples have been tested with Apache HTTPD 2.4 server.

Consider that Apache may be installed and configured in multiple ways. Here we cover only an example on how to do it. For more information on how to add additional configuration files to Apache HTTPD server, refer to [APACHE-CONFIG-FILES].

Copy the file and paste it into the apache configuration folder (it depends on the O.S.).

If you are using Windows, it is likely that your Apache installation file system includes a subfolder named `extra` within the `conf` folder. If this is your case, paste the `service-configuration.conf` in the `extra` folder and mention its presence inside the `httpd.conf` file (available in the `conf` folder). To do so, open the file `httpd.conf` and include at the end of the document, the following lines as shown in Figure 38.

Figure 38: `httpd.conf` file configuration

```
# Registry configuration
# Include 'conf/extra/service-configuration.conf'
```

Arrived at this point, open the `service-configuration.conf` file to edit and save the required modifications. The settings to edit are located in the beginning of the file, see Figure 39.

Figure 39: Settings to configure the HTTP server

```
Alias /data /      [root-path]/output/custom
DocumentRoot      [root-path]/output/custom

#<Directory "[root-path]/output/custom">
#   AllowOverride none
#   Require all granted
#</Directory>
```

The user should replace twice the value `'/[root-path]/output/custom'` by the path where the *exported* data is stored.

(Remember this path is the given to the 'staticizer.custom.folder.path=' of the `Re3gistryStaticizer.properties` configuration file).

Regarding the commented lines (beginning with #), check if you need to uncomment it to make it work.

Note: Remember to restart the web server every time you perform a modification, in order Apache to load the new configuration file(s).

5.3.5. Set up the service-specific configuration

The **Re3gistry** package includes a generic user interface through which the contents are displayed. The files handling those are included in the `webapp-configurations` folder.

Path: `<root_folder>/Project_package_1.X/webapp/examples/<inspire-example|neutral-example>/webapp-configurations`

This generic interface is suitable and ready to use for both the available examples: `inspire-example` and `neutral-example`.

To make use of the generic user interface, copy the `webapp-configurations` 5 subfolders available in the **Re3gistry** package and paste them into the `'app_data'` folder present into the output data folder where the exported data is stored. The `app_data` folder should appear as in Figure 40.

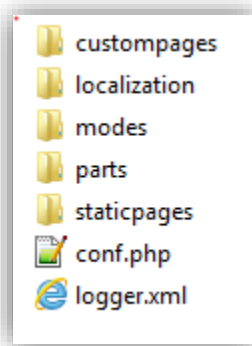


Figure 40: *app_data* folder structure after editing

The layout and the contents belonging to the web application are fully customisable. For more information on the customisation of the web service see section: Customising the Re3gistry.

5.4. Managing *solr*

5.4.1. Installing *solr*

After downloading the Apache *solr* 4.8.0 version¹³ (available in the archived versions), go to the `dist` folder of the package and copy the `solr-4.8.0.war` file to paste inside Tomcat's `webapp` folder.

You also need to copy the libraries contained in `solr-4.8.0/dist/solrj-lib` and paste them inside Tomcat's `lib` folder.

Again in the *solr* package, copy the folder '`solr`' available under the `example` folder and copy it as well in tomcat's `webapp` folder renaming it '`solr-home`'.

5.4.2. Configuring *solr*

5.4.2.1. `solr.xml`

Browse tomcat's configuration files following this path: `tomcat/conf/Catalina/localhost` and once there, create a file called `solr.xml`.

*The name of the file must match with the name of the *solr* instance places in the tomcat's *webapp*, folder if you didn't change *solr.xml* should work properly*

Open the file and add the lines indicated in Figure 41, by changing appropriately the paths to both your `solr.war` file and its data repository '`solr-home`' (normally already available in tomcat's `webapp` folder).

¹³ <https://archive.apache.org/dist/lucene/solr/4.8.0/>

Figure 41: Configuration of the solr.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<Context docBase="C:/xampp/tomcat/webapps/solr.war" debug="0"
crossContext="true">
  <Environment name="solr/home" type="java.lang.String"
value="C:/xampp/tomcat/webapps/solr-home" override="true"/>
</Context>
```

5.4.3.core.properties of solr registry collection

Within the `solr-home` recently renamed folder, you must find a subfolder named 'collection1', copy it and rename it, for example 'registry' (See Figure 42) .

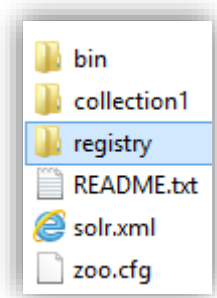


Figure 42: New 'registry' collection for solr

Inside registry folder, there must be a file named 'core.properties', open the file and edit the value for the name so that it matches with the name given to the folder, in our example: `name=registry`.

5.4.3.1.1. Schema.xml

Finally, you need to place the `schema.xml` file under the `conf` folder of the solr registry collection. The xml file comes within the Re3gistry package under its the `example` folder. Go to your respective chosen example and take the `schema.xml` file available within the folder called 'solr'.

Restart Tomcat to apply the changed and check that solr is working. The solr administration panel (See Figure 41) should be accessible by follow this URL pattern:

```
http://localhost:[tomcat_port]/[Name_Of_Webapp]
```

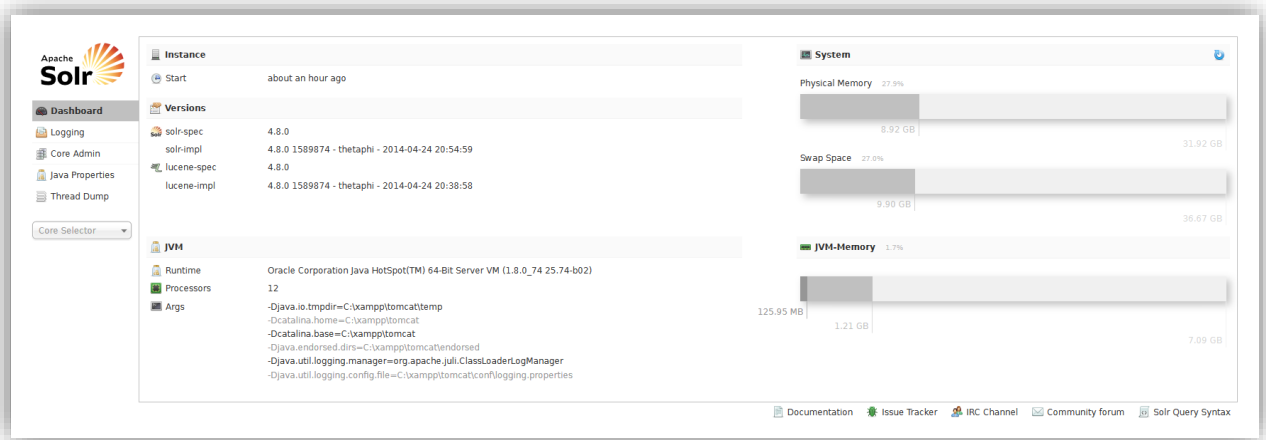


Figure 43: solr administration panel

5.5. Connecting solr to the Re3gistry webapp

Once your solr instance is properly installed and running, you will be able to define the solr endpoint in the `conf.php` file (already manipulated in former steps when setting up the web application), to enable both the search and the autocomplete functionalities (Find an example in Figure 44). The solr endpoint normally follows this pattern:

```
http://[IP_of_your  
machine]:[tomcat_port]/[solr_instance_Name]/[name_solr_collection]/select
```

```
/* Searching system configs */  
define('SEARCH_AUTOCOMPLETE_URL', 'http://localhost:8081/solr/registry/select'); //  
define('SEARCH_CORE_URL', 'http://localhost:8081/solr/registry/select'); // The Ap  
// The searching system needs an instance of Apache Solr
```

Figure 44: Example of definition of solr endpoint in the `conf.php`

5.6. Indexing your registry contents

To provide a good user experience when searching in the web application, you need to index the data of the registry as you modify it.

To execute the index procedure, you need to populate or update your solr service through its 'update' operation by using the `post.jar` library that will fetch the data in the indicated folder to index it and make it available. See an example of how to execute it in the shell in Figure 48.

Figure 45: command to update the indexing of solr

```
java -Durl=http://localhost:8081/solr/registry/update -Dauto -Drecursive -
jar post.jar C:/xampp/tomcat/webapps/Re3gistry-
1.X/DataRepository/StaticisedData/solrcustom/
```

To check that the indexing has worked properly make a test query as for example:

```
http://localhost:8081/solr/registry/select?q=*
```

Alternatively, simply, go to the solr administration panel and check its results (making sure you requesting the collection of data related to the registry contents) See Figure 46.

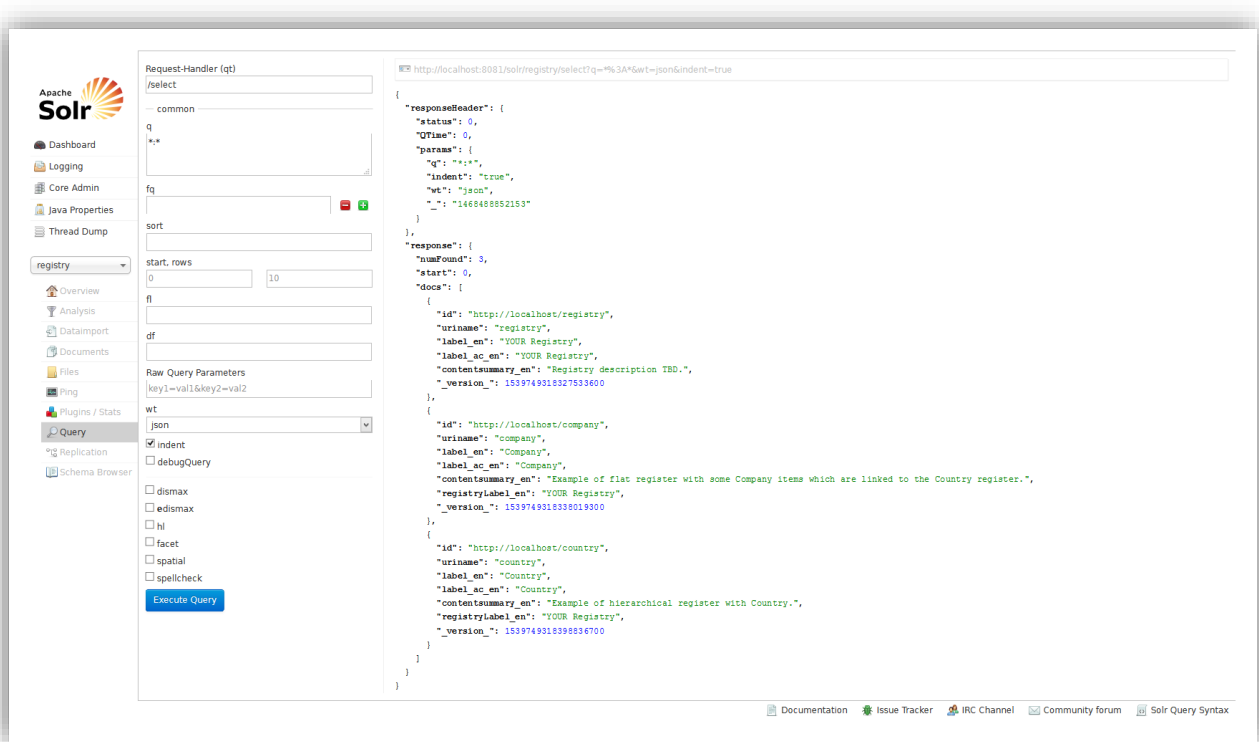


Figure 46: Querying indexed contents within the solr administration panel,

To call the `post.jar` function, remember to locate the shell in the proper solr package folder, where the library is available (normally at the path `solr-4.8.0/example/exampledocs/post.jar`) or to specify explicitly the location of the library.

5.7. Testing the web service

To check that everything is working properly, try the URL you defined in the configuration file `conf.php` when setting up the web application (Check previous steps).

```
/* Webapp URLs */
define('REGISTRY_BASE_URL', 'http://localhost/registry/'); // The root
URL of the registry service. (Example: http://localhost/registry/)
```

By default, if you have not changed the value, it should be answering to the URL <http://localhost/registry/>. If that is the case, you should get a website similar to the one shown in Figure 47.

In the image of the following Figure, only XML and JSON formats are available, as set by the user according to its preferences in the former steps (See configuration of `Re3gistryStaticizer.properties`). Also note that the example is incomplete information on the registry is still to be completed. These questions will be covered in the customisation section.

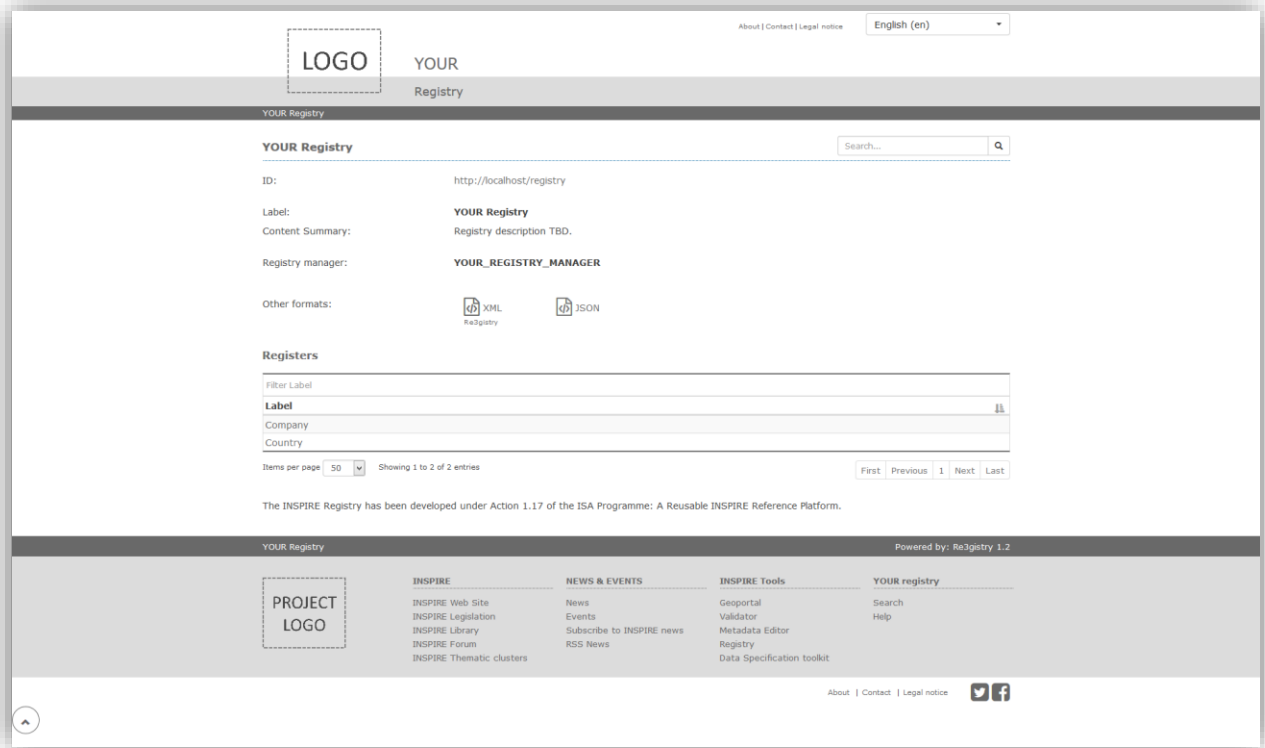


Figure 47: Re3gistry generic user interface serving the 'neutral-example' registers contents

To finish, check that the indexing function works properly, to do that try to search something you know it should have been indexed, as for example the names of the available registers provided with the examples. Normally only by typing it should get some suggestions (See Figure 48).

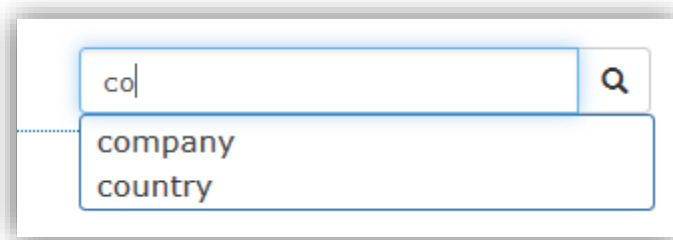


Figure 48: Autocomplete function

5.8. INSPIRE register federation descriptors files- RoR descriptors

The **Re3gistry** provides also descriptor files for the registry and its registers called RoR descriptor files, because of the 'Registry of registers'.

This files allows the user adding those registers that he considers important, normally because they are extending INSPIRE into the *INSPIRE register federation service*¹⁴.

*The descriptor file is considered by the **Re3gistry** as an additional type of format with the difference that it is only provided in English, hence, the file does not need the language identifier.*

The descriptor file contains the list of all the registers available in the registry system. To see the contents of the file, you should be able to obtain it by typing the URL following the pattern shown in Figure 50.

Figure 49: Generic URL to get the ROR file in the re3gistry

```
http://registry.example/registry/registry.ror
```

¹⁴ <http://inspire-regadmin.jrc.ec.europa.eu/ror/>

6. Customising the Re3gistry

The following section will help you creating your own registry and registers.

To do so, we will reuse one of the example `neutral-example` provided with the software package to adapt them as needed. It consists of a registry that contains two registers 'Country' and 'Companies', being 'Country' a hierarchical register and 'Companies' a plain one.

The `neutral-example` example is available in the package software and located at:

```
<root_folder>/Project_package_1.X/examples/neutral-example
```

To create a new customised project, we suggest you to begin by copying the folder of the example named `neutral-example` and to rename as you like. In this guide, in order to refer to the paths of the files, we will rename the folder to 'customised-example'.

6.1. Customising the Re3gistry contents

6.1.1. Creating a costumised registry

Most of the settings for the registry and its contents are contained in the `database-initialization.sql` SQL file. The content of this file relies very closely on the `create-tables.sql` file that produces the database structure (tables, columns etc.) and hence, it should have already been launched.

With a text editor, open the `database-initialization.sql` file and update the contents of the file as explained in the following steps. The `database-initialization.sql` file is located at:

```
<root_folder>/Project_package_1.X/examples/customised-example/database/database-initialization.sql
```

6.1.1.1. Setting the registry parameters

The information related to the registry system is located under the 'registry' section in the script and it relates to the named 'registry' database table (See Figure 50).

| uuid [PK] character varying(300) | uriname character varying(300) | baseuri character varying(200) | registrymanager character varying(300) | datecreation timestamp without time zone | datelastupdate timestamp without time zone |
|-------------------------------------|-----------------------------------|-----------------------------------|---|---|---|
|-------------------------------------|-----------------------------------|-----------------------------------|---|---|---|

Figure 50: Fields of registry table of the database

Look for the following line which is below the 'registry' table section:

```
--registry
INSERT INTO registry VALUES ('1', 'registry', 'http://localhost', '4', '2016-05-28 15:30:00', NULL);
```

Some of the highlighted elements may need to be updated depending on your needs. In order of appearance these elements are:

- The **unique identifier** (`uuid`) of the registry. This parameter needs to be changed if you intend to work with two or more registries. If you are using a single registry, keep the default value '1'.
- The **code** (`uriname`) related to the registry. By default, it is set to 'registry'. Consider to rename it as this code will be appended to the chosen 'base URI' to compose the URIs of the registry.
- The **base URI** of the registry

In this case, the resulting URL of the registry will be `http://localhost/registry`.

6.1.1.2. Defining the email address for the registry contact point

The elements appearing in this part of the SQL script contain the metadata for both the registry and the registers. They relate to the 'reference' database table (See Figure 51).

| uuid | email | reftype | datecreation | datelastupdate |
|------------------------|--------------|------------------------|-----------------------------|-----------------------------|
| [PK] character varying | character va | character varying(300) | timestamp without time zone | timestamp without time zone |

Figure 51: Fields of reference table of the database

Look for the following line below the 'reference' table section:

```
INSERT INTO reference VALUES ('1', 'YOUR_EMAIL@email-example.eu',
'contactpoint', '2014-10-14 00:00:00', NULL);
```

Update the highlighted parts of the line with the email address that you want your registry to have as contact point.

6.1.1.3. Setting the supported languages

This part of the script handles the multilingualism of both the registry and the registers.

Every line of the 'language code' section defines a new language. By default, the script contains all the official European languages. See Figure 52 to see where the language is configured in the script.

You can remove those that you do not need, or instead, add further languages depending on your requirements.

Figure 52: Language configuration

```
--languagecode
INSERT INTO languagecode VALUES ('1', 'english', 'en', 'eng', TRUE);
INSERT INTO languagecode VALUES ('2', 'italiano', 'it', 'ita');
INSERT INTO languagecode VALUES ('3', 'czech', 'cs', 'cze');
```

.....

The highlighted parameters might need to be updates. In order of appearance these are:

- The **unique identifier** (`uuid`) of the language.
- The **label of the language** (the human readable label).
- The **two-digit code of the language**.
- The **ISO 639 – 2 three-digit code of the language**.
- The **default system language** (*masterlanguage*) for the registry system. Note that only a single language can be set as the master language (TRUE).

Depending on your organisation needs, it could be more convenient for you to set the master language to your official language, since registered items must be available at least for the master language.

6.1.1.4. Setting the status values

The information available under the `status` section of the script, contain the values together with the public URIs that will be used by the versioning system of the [Re3gistry](#).

The codes (`uriname`) for the different status come from the *[ISO 19135] 'Procedures for item registration'* standard.

We recommend to update solely the base URL, to align it with your registry URL as defined in the step 6.1.1.1. (See Figure 56)

Figure 53: Definition of the register status

```
--status
INSERT INTO status VALUES ('1', 'http://localhost/registry/status',
'valid');
INSERT INTO status VALUES ('2', 'http://localhost/registry/status',
'invalid');
INSERT INTO status VALUES ('3', 'http://localhost/registry/status',
'submitted');
INSERT INTO status VALUES ('4', 'http://localhost/registry/status',
'superseded');
INSERT INTO status VALUES ('5', 'http://localhost/registry/status',
'retired');
```

6.1.2. Creating costumised registers

6.1.2.1. Setting the register parameters

The following section of the script handles the creation of the registers that will be contained in your registry.

Look for the lines beginning by `INSERT INTO register VALUES,` present under the ‘register’ section (See Figure 56).

Figure 54: Register section of the script

```
--register
INSERT INTO register VALUES ('1', 'http://localhost', 'country', '6', '5',
'3', '7', '1', '2', '1', '2016-03-25 14:14:05.33835', NULL);
INSERT INTO register VALUES ('2', 'http://localhost', 'company', '6', '5',
'3', '7', '1', '2', '1', '2016-03-25 14:16:22.677696', NULL);
```

For each register that you plan to create, there must be as many `INSERT INTO` statements, as registers will be hosted.

The table of the database where this information is stored is named ‘register’.

In the example file that we are using as template, you can see that there are two registers: ‘country’ and ‘company’.

The highlighted parameters in Figure 54, need to be updated according to your needs. In order of appearance these elements are:

- The **unique identifier** (`uuid`) of the register.
- The **base URL** of the register. If there is more than a register in a common registry, the value of the base URL needs to be shared.
- The **code** (`uriname`) related to the register. It can be considered as the machine readable name of the register, consider to rename it with a proper and recognizable name as it will be appended to the former ‘base URI’ to compose the URIs of the registry.
 - The **reference to the id of the registry** assigned in the 6.1.1.1 step. If you want to handle more than a registry, make sure you are referring to the correct `registry ID`.

6.1.2.2. Defining the itemclass

This part of the SQL Script handles the itemclasses of your register. An *itemclass* is a key element of the register where parent/child relationships can be defined. To know more on the ‘itemclass’, please refer to the section 2.1.1.3, The *itemclass* component.

Look for the lines under the ‘itemclass’ section as shown in Figure 55.

Figure 55: itemclass section of the script

```
--itemclass
INSERT INTO itemclass VALUES ('1', '1', 'Country', 0, 1, NULL, '2016-03-26
11:24:45.727957', NULL);
INSERT INTO itemclass VALUES ('2', '1', 'Region', 1, 2, '1', '2016-03-26
11:24:56.096222', NULL);
```

```
INSERT INTO itemclass VALUES ('3', '2', 'Company', 0, 3, NULL, '2016-03-25
15:40:30.457694', NULL);
```

The highlighted elements need to be updated according to your register contents. In order of appearance these elements are:

- The **unique identifier** (`uuid`) of the itemclass.
- The **reference to the register unique identifier** it belongs to (`register`), assigned in previous step.
- The **itemclass** code (`uriname`), consider to rename it with a proper and recognisable name as it will be part of the public URI .
- The **hierarchical order** of the *itemclass* (`order number`). By default, it is set to '0'. For plain registers keep the default value, in case of a hierarchical register make sure the children itemclass appear after the parent one and that their hierarchical order number are higher than the parent one.

In the example, the hierarchical Country register contains information on Countries (Country itemclass) and regions belonging to them (Region itemclass).

- The **data procedure order number**, that number will define when the data is going to be processed (`dataprocudure`) for its load in the database. Those itemclass that have lower numbers will be processed first.
- The **reference to the identifier** of the parent itemclass (`parent`)

Note that in the example, the country register has a parent-child relationship, where the country itemclass contains the region itemclass. For that reason, country itemclass is considered the parent of region itemclass and regarding the data procedure order, the country itemclass must be processed before the region one.

6.1.3. Translating the content

The elements of the registers are subject to 'localisation', that is, they can be translated or 'localised'. These contents, are handled by the 'Localization' table of database (See Figure 56). This table contains the multilingual fields as label, definition, description and any custom attribute.

| uuid | item | itemclass | language | register | customattributevalue | label | definition | description | uri | datecreation | datelastupdate | registry | reference | status |
|--------------|---------------|---------------|---------------|---------------|-----------------------|------------------------|------------|-------------|------|--------------|----------------|-----------------------|-----------------------|---------------|
| [PK] charact | character vai | character vai | character vai | character vai | character varying(50) | character varying(400) | text | text | text | timestamp w | timestamp with | character varying(50) | character varying(50) | character vai |

Figure 56: Fields in the Localization table of the database

To run the example and populate the database with localised data we will make use of the script named `database-localization.sql` available at:

```
<root_folder>/Project_package_1.X/examples/customised-example/database/
database-localization.sql
```

The 'master language' needs to be always available, whereas the localisation for any other language is optional.

If the system does not find the language, it will reuse the available value in the 'master language' to represent the item.

The script for the localisation will refer to every item, inserted by the initialisation script, using their unique id.

The structure of the script is shown in Figure 57. Depending on the element to be localised, the contents in square brackets shall be completed, See an example of its usage in Figure 58.

Figure 57: localisation settings

```
INSERT INTO localization VALUES('_localization unique id_', '['_item unique id_']', '['_itemclass unique id_']', '_language code_', '['_register unique id_']', '['_custom attribute value unique id_']', '_label_', '_definition_', '_description_', '_uri_', '2015-06-30 00:00:00', '_date last update_', '['_registry unique id_']', '['_reference unique id_']', '['_status unique id_']');
```

Figure 58: Example of localisation file for the English language

```
--reference information localization for language en

INSERT INTO localization VALUES('1',NULL,NULL,'1',NULL,NULL,' Registry manager name',NULL,NULL,NULL,'2015-06-30 00:00:00',NULL,NULL,'4',NULL);

INSERT INTO localization VALUES('2',NULL,NULL,'1',NULL,NULL,'Your contact point',NULL,NULL,NULL,'2015-06-30 00:00:00',NULL,NULL,'1',NULL);

INSERT INTO localization VALUES('3',NULL,NULL,'1',NULL,NULL,'Legal notice label',NULL,NULL,'http://ec.europa.eu/geninfo/legal notices en.htm','2015-06-30 00:00:00',NULL,NULL,'2',NULL);

INSERT INTO localization VALUES('4',NULL,NULL,'1',NULL,NULL,'INSPIRE Maintenance and Implementation Group (MIG)',NULL,NULL,NULL,'2015-06-30 00:00:00',NULL,NULL,'3',NULL);

INSERT INTO localization VALUES('5',NULL,NULL,'1',NULL,NULL,'Members of INSPIRE Maintenance and Implementation Group (MIG)',NULL,NULL,NULL,'2015-06-30 00:00:00',NULL,NULL,'7',NULL);

INSERT INTO localization VALUES('6',NULL,NULL,'1',NULL,NULL,'European Commission, Joint Research Centre',NULL,NULL,NULL,'2015-06-30 00:00:00',NULL,NULL,'5',NULL);

INSERT INTO localization VALUES('7',NULL,NULL,'1',NULL,NULL,'European Union',NULL,NULL,NULL,'2015-06-30 00:00:00',NULL,NULL,'6',NULL);

-- localization for the registry information(language en)

INSERT INTO localization VALUES('8',NULL,NULL,'1',NULL,NULL,'TEST registry','This is the description of the registry',NULL,NULL,'2015-06-30 00:00:00',NULL,'1',NULL,NULL);

--register localization for language en
```

```
INSERT INTO localization VALUES('9',NULL,NULL,'1','1',NULL,'INSPIRE simple register', 'This is a test.',NULL,NULL,'2015-06-30 00:00:00',NULL,NULL,NULL,NULL);

INSERT INTO localization VALUES('10',NULL,NULL,'1','2',NULL,'INSPIRE hierarchical register', 'This is a test.',NULL,NULL,'2015-06-30 00:00:00',NULL,NULL,NULL,NULL);

--itemclass localization for language en

INSERT INTO localization
VALUES('11',NULL,'2','1',NULL,NULL,'Simple',NULL,NULL,NULL,'2015-06-30 00:00:00',NULL,NULL,NULL,NULL);

INSERT INTO localization
VALUES('12',NULL,'3','1',NULL,NULL,'Hierarchical',NULL,NULL,NULL,'2015-06-30 00:00:00',NULL,NULL,NULL,NULL);

INSERT INTO localization VALUES('13',NULL,'4','1',NULL,NULL,'Hierarchical level 1',NULL,NULL,NULL,'2015-06-30 00:00:00',NULL,NULL,NULL,NULL);

--status localization for language en

INSERT INTO localization
VALUES('14',NULL,NULL,'1',NULL,NULL,'Valid',NULL,NULL,NULL,'2015-06-30 00:00:00',NULL,NULL,NULL,'1');

INSERT INTO localization
VALUES('15',NULL,NULL,'1',NULL,NULL,'Invalid',NULL,NULL,NULL,'2015-06-30 00:00:00',NULL,NULL,NULL,'2');

INSERT INTO localization
VALUES('16',NULL,NULL,'1',NULL,NULL,'Submitted',NULL,NULL,NULL,'2015-06-30 00:00:00',NULL,NULL,NULL,'3');

INSERT INTO localization
VALUES('17',NULL,NULL,'1',NULL,NULL,'Superseded',NULL,NULL,NULL,'2015-06-30 00:00:00',NULL,NULL,NULL,'4');

INSERT INTO localization
VALUES('18',NULL,NULL,'1',NULL,NULL,'Retired',NULL,NULL,NULL,'2015-06-30 00:00:00',NULL,NULL,NULL,'5');
```

6.1.4.Import data file

This part of the guide helps you creating you own `import data file`. We recommend you to take some of the existing examples coming in the software package and adapt it as needed.

If you prefer to use a software that organises your data into rows and columns, that is spreadsheets, instead of dealing with basic notepad programs, we recommend you the use of the open source Open Office Cal¹⁵ with the appropriate settings to work with the Re3gistry as shown in Figure 59

¹⁵ <https://www.openoffice.org/>

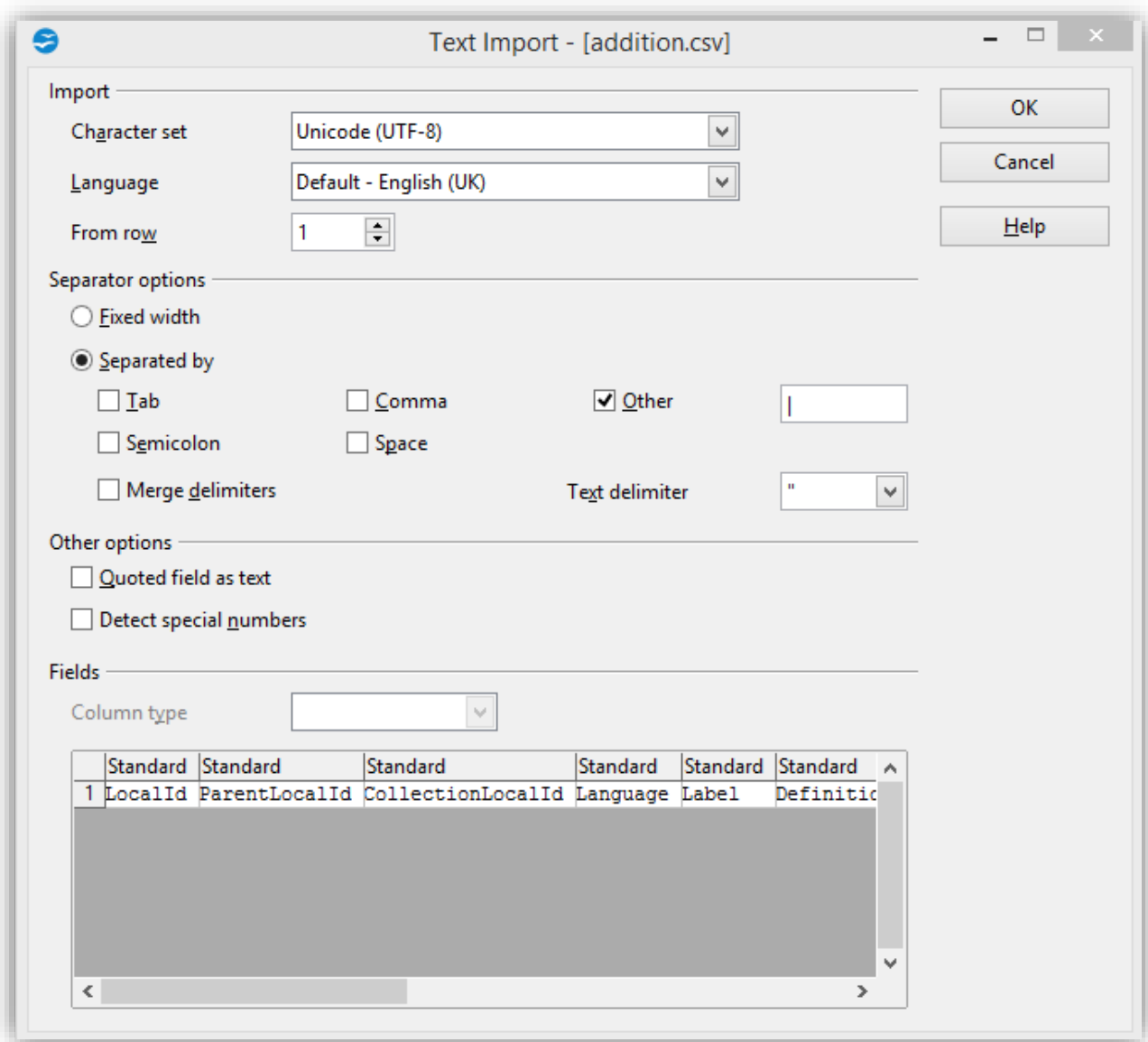


Figure 59: Settings in Open Office Calc to open appropriately the Re3gistry import action files.

By using a spreadsheet manager you should get something as in Figure 60, otherwise, if you prefer to use a notepad program it will look like more to Figure 61.

| | A | B | C | D | E | F | G | H | I |
|---|---------|---------------|-------------------|----------|---------------|------------------|------------------|--------|--------------|
| 1 | LocalId | ParentLocalId | CollectionLocalId | Language | Label | Definition | Description | Status | Comment |
| 2 | it1 | | | en | item test | test definition | test description | valid | First import |
| 3 | it1 | | | it | elemento test | definizione test | descrizione test | valid | primo import |

Figure 60: Re3gistry import action files in a spreadsheet (Open Office Calc)


```

1 LocalId|ParentLocalId|CollectionLocalId|Language|Label|Definition|Description|Status|Comment
2 it1||en|item test|test definition|test description|valid|First import
3 it1||it|elemento test|definizione test|descrizione test|valid|primo import
    
```

Figure 61: Re3gistry import action files in a notepad program (Notepad++)

- Start by opening the example data file contained in the software package '`<root_folder>/Project_package_1.X/examples/neutral-example/data`' .
- Rename the file for example to '`CustomisedExample.zip`' and unzip the file. The folder and files contained were created for the '`neutral-example`', sample files used during the installation guide in section 3 .
- Take a look at the structure of the folders and the files. The data file structure, shall contain one folder per *itemclass* available in the system, and per each itemclass there must be several CSV files matching the available actions.

However, if the only action to perform is an **addition**, the only file that needs to be present in the folder is the `addition.csv` file. For more information on the different action supported by the Re3gistry software, refer to section 2.2.3.1.3.

- To create the data file for the '`customisedExample`' create the three subfolders needed for this example and name them as you wish, for the guide purpose, we will rename as following:
 - '`simple`', it will contain a plain register based on the '`Company`' example register.
 - '`hierarchical`', it will contain the first level of a hierarchical register based on the '`Country`' example register.
 - '`hierarchicallevel1`', it will contain the second level of the linked to the former hierarchical register, based on the '`Region`' example register.

The name of the folders must match exactly with the *itemclass* names defined in the database though the database creation scripts, see section 3.4.

6.1.4.1. Simple register

The file `addition.csv`, will add a custom attribute called `ExampleCustomAttribute`. This customised attribute will not be multivalued, coded or foreign key. For more information on the customised attributes, please refer to section 2.1.2.

```
<root_folder> /customised-example/data/Simple/addition.csv
```

The example shown in Figure 62, will insert, through an addition action an item with the identifier (id) '`it1`' in English and Italian in a simple type register.

Figure 62: Example of an addition file with an additional customised attribute in a simple register

```

LocalId|ParentLocalId|CollectionLocalId|Language|Label|Definition|Description|Status|Comment|*ExampleCustomAttribute[t,f,f,f]
    
```

```
it1|||en|item test|test definition|test description|valid|First
import|example custom attribute content

it1|||it|elemento test|definizione test|descrizione test|valid|primo
import|esempio di contenuto per custom attribute
```

6.1.4.2. Hierarchical register

6.1.4.2.1. Hierarchical register – first level

The example in Figure 63, will insert three items with the id 'h1', 'h2' and 'h3' in a hierarchical register in English with no customised attributes.

```
<root_folder> /customised-example/data/Hierarchical/addition.csv
```

Figure 63: Example of addition in hierarchical register using the addition.csv of its itemclass

| LocalId | ParentLocalId | CollectionLocalId | Language | Label | Definition | Description | Status | Comment |
|---------|---------------|-------------------|----------|-------------|-----------------|------------------|--------|--------------|
| h1 | | | en | item test 1 | test definition | test description | valid | First import |
| h2 | | | en | item test 2 | test definition | test description | valid | First import |
| h3 | | | en | item test 3 | test definition | test description | valid | First import |

The example shown in Figure 64, will invalidate the element with the id h3 and will set the element h2 as successor of the invalidated item.

```
<root_folder> /customised-example/data/Hierarchical/invalidation.csv
```

Figure 64: Example of invalidation of an item

| LocalId | CollectionLocalId | SuccessorLocalId | SuccessorCollectionLocalId | Comment |
|---------|-------------------|------------------|----------------------------|---------|
| h3 | | h2 | | |

6.1.4.2.2. Hierarchical register – second level

The example in Figure 65, will insert an item with the id 'h11' and another with the id 'h12' in the hierarchical register created in the former section 6.1.4.2, in English with no custom attribute.

This file is related to an itemclass that has a parent (the parent is the 'Hierarchical' itemclass used in the former section); that is why the field `CollectionLocalId` is filled with an element from the 'Hierarchical' itemclass.

The h12 item has also a parent/child relation. In fact, it has the item h11 as parent. This parent/child relation is different from the collection hierarchy relation. Indeed, the collection hierarchy relation establishes a relation between elements belonging from different itemclasses (like the 'Hierarchical' and 'HierarchicalLevel1' itemclass). The parent/child relation establishes a relation between elements belonging from the same itemclass.

```
<root_folder> /customised-example/data/HierarchicalLevel1/addition.csv
```

Figure 65: Creation of the second level of a hierarchical register

| LocalId | ParentLocalId | CollectionLocalId | Language | Label | Definition | Description | Status | Comment |
|---------|---------------|-------------------|-------------|-----------------|------------------|------------------|--------------|--------------|
| hl1 | h1 | en | item test 1 | test definition | test description | valid | First import | |
| hl2 | hl1 | h2 | en | item test 2 | test definition | test description | valid | First import |

6.1.5. Transformation files

For each new register added to the system, a related XSLT transformation files shall be created in order to get the right export files.

An example of XSLT files can be found in the 'customised example' folder (duplicated from the 'neutral-example') folder: /example/xsl

There is an XSLT file for each register and for each itemclass available in the system. Below in Figure 72, there is a list with the location of the files needed to produce the HTML format supporting the simple and hierarchical register explained in the sections above.

Figure 66: XSLT files needed to support the conversion in HTML formats for the simple and hierarchical registers

```
<root_folder> /customised-example/xsl/html/simple_register.html.xsl
<root_folder> /customised-example/xsl/html/Simple.html.xsl
<root_folder> /customised-example/xsl/html/hierarchical_register.html.xsl
<root_folder> /customised-example/xsl/html/Hierarchical.html.xsl
<root_folder> /customised-example/xsl/html/HierarchicalLevel1.html.xsl
```

6.1.6. Deployer configuration

The `deployer module` is responsible for deploying all the static files produced by the staticisation system to the target production server. This is needed if the production server is in a different machine (or the files in the same machine need to be moved to another place in the same system).

This step is optional, in fact, the files produced by the **Re3gistry** can also be moved by hand to the target path.

The module automatically takes all the static files produced and move them to the configured target place.

The system uses a configurable script to launch the deployment process. This script can be found in the main application properties folder under

```
<root_folder>/Project_package_1.X/binaries/Re3gistry-1.X/WEB-INF/classes/scripts/Re3gistryDeployer.
```

The system automatically recognises the operative system (Linux or Windows) and launch the relative file (`linux-deploy.sh` or `windows-deploy.bat`). This shell script can be configured and customized to perform the needed operation in order to move the produced files to another directory as well as to another server.

To configure the Deployer module, the properties to be customized are contained in the file

```
<root_folder>/Project_package_1.X/binaries/Re3gistry-1.X/WEB-INF/classes/configurations/module/RegistryDeployer.properties
```

The properties are described below.

- `deploy.script.folder`: this property represents the folder where the `.sh` or `.bat` file are stored.
- `deploy.rss.file`: this property is the location of the base RSS file. It is the master file that keeps all the RSS news. If the 'RSS update' option has been selected, the RSS update system takes this file and updates it with the new information. Then the file is copied to the target path.
- `deploy.rss.targetfolder`: the folder in which the RSS file is stored after the update (usually it is the same folder that contains all the files produced by the staticizer).
- `deploy.rss.baseuri`: this is the base URI of the RSS file, stored in the FSS file as the channel's 'link' element (example: for the inspire registry, it is <http://inspire.ec.europa.eu/registry/rss/>)

6.2. Customising the Re3gistry web interface

The **Re3gistry** web application (webapp) is a component that allows you to publish the data exported by the **Re3gistry** software through a web interface, exposing also all of the formats produced (e.g. XML, JSON, RDF, ...).

The webapp is available at

```
<root_folder>/Project_package_1.X/webapp'
```

To set up the basic configuration of the web application, follow the guide available at section 0.

The following paragraphs will guide you understanding how the system works and how to set up your own service. All the files to be customised are available at folder

```
<root_folder>/Project_package_1.X/examples/<selected-example>/webapp-configurations
```

In case this section is read after the

Serving the Re3gistry contents *at* section 5 , the user should have created the 'custom-example' folder by duplicating the 'neutral-example' folder. In this case all the files are already in the 'webapp-configurations' folder, and they have only to be updated.

6.2.1. Webapp structure

The web application reads the data from the JSON files exported by the **Re3gistry** software and shows them in the web interface.

The web pages are provided in different languages; the user interface is localised using the json-based localisations files contained in the folder

```
<root_folder>/Project_package_1.X/examples/<selected-exampe>/webapp-  
configurations/app_data/localization
```

To change the default labels in the desired language, edit the <XX>.json file (<XX> represents the 2-digit language code).

The following paragraphs will explain in details how to customise the web application.

6.2.2. Modes

A registry is made of different components: register and register items that are differently described and hence differently represented as far as the layout is concerned.

For this reason, the **Re3gistry** webapp provides by default four 'layout modes':

- Mode 1: Registry page
- Mode 2: Register page
- Mode 3: Detail page
- Mode 4: Detail page for the hierarchical register (like the INSPIRE code list register).

For each of the different elements available in the registry (e.g. Registry page, Registers page, Items page), there are different representations, that is different fields to be visible in the web pages.

To better explain this concept, we recommend you to take a look at the INSPIRE registry, an instance of the **Re3gistry**, more specifically at the different fields available in one of the INSPIRE theme web page [INSP-THEME-AD] see **Error! Reference source not found.** and in the INSPIRE code list register web page [INSP-CODELIST-AccessRestrictionValue], see Figure 68.

The screenshot displays the 'Addresses' theme page within the INSPIRE Registry. At the top, there is a navigation bar with the INSPIRE logo and the text 'INSPIRE Registry'. Below this, a breadcrumb trail reads 'European Commission > INSPIRE > INSPIRE registry > INSPIRE theme register > Addresses'. A search bar is located on the right side of the page. A blue banner encourages users to help improve the Re3gistry software by filling out a survey at <http://europa.eu/1Bn84Ct>.

The main content area provides the following metadata for the 'Addresses' theme:

- ID:** <http://inspire.ec.europa.eu/theme/ad>
- This version:** <http://inspire.ec.europa.eu/theme/ad:1>
- Latest version:** <http://inspire.ec.europa.eu/theme/ad>
- Label:** **Addresses**
- Definition:** Location of properties based on address identifiers, usually by road name, house number, postal code.
- Description:** An address is an identification of the fixed location of a property. The full address is a hierarchy consisting of components such as geographic names, with an increasing level of detail, e.g.: town, then street name, then house number or name. It may also include a post code or other postal descriptors. The address may include a path of access but this depends on the function of the address.

Addresses serve several purposes, these include the four uses described in the Dutch Address Registration catalogue:

 - (i) location (e.g. for visits or the delivery of mail),
 - (ii) identification (e.g. in context of a building registration),
 - (iii) jurisdiction (e.g. authority responsible for the property identified by the address),
 - (iv) sorting and ordering (VROM 2006). There may be other uses identified in the INSPIRE user requirements survey, for example, to aid emergency response.

A number of different object types can be related to property. The most commonly recognised types that have addresses are land parcels and buildings (including flats or apartments). However, other object types, such as water pumping stations, and agricultural buildings, are also types of property. Although they do not receive post they may need to have an address for other functions. This is true in both rural and urban areas. Some other property types that might have addresses include a sports ground, a foothold or a mooring place. Collectively, objects which can have addresses are referred to as addressable objects.

The location of an address is most often defined so that it characterises the location of the related addressable object.

Although all national or local address systems share similar concepts and general properties, differences exist in formal and informal standards, rules, schemas and data models within Europe. Differences also exist in the extent of the address system, for example, it may be simplified in rural areas. (From revised D2.3)
- Governance level:** eu-legal
- Status:** Valid
- Annex:** I

At the bottom, under 'Other formats:', there are five download icons: XML (Re3gistry), XML (ISO 19135), RDF/XML, JSON, and Atom.

Figure 67: Addresses theme within the theme register of the INSPIRE Registry

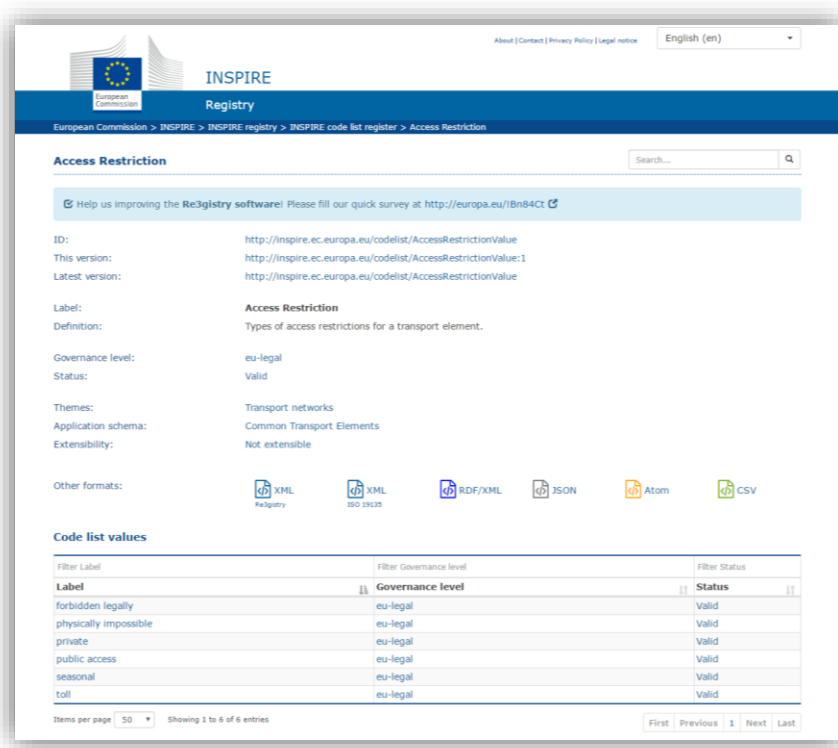


Figure 68: Restriction code code list within the code list register of the INSPIRE Registry

The detail pages always provide the standard fields as ‘label’, ‘definition’, ‘description’ and ‘status’ ; but also they may differ with additional customised fields closely related on the type (itemclass) of each of the items (e.g. ‘Annex’ for the Theme, ‘Themes’, ‘Application schema’ and ‘Extendibility’ for the Code List).

The web application provided by the **Re3gistry** software allows you to customise the page for each of the itemclass and the element type described above (Registry, Register and Items). If the data file for a specific itemclass has more costumised fields, they can be specified using a JSON file that describes the fields to be visible for that specific itemclass.

This is done through a configuration file in the JSON format. This file, also called ‘mode descriptor’ is contained in the following folder:

```
<root_folder>/Project_package_1.X/examples/<selected-exampe>/webapp-
configurations/app_data/modes/mode.descriptor.json
```

This ‘mode descriptor’ refers to the elements contained in the related JSON data files (the data files produced by the **Re3gistry**) and in the GUI localisation files (described at the beginning of this section).

The next examples contain snippets from each of the files referenced by the descriptor: JSON data file and the GUI localisation file.

To help the understanding of each properties, a color code has also been used: the **color** used in the following example of the GUI localisation file, will be used to match the fields in the mode descriptor files (subsequent sections).

6.2.2.1. GUI localisation file

This is an example of GUI localisation file; it is useful to understand how it is used in the 'mode descriptor' (described in the following sections).

```
{
...
"label":"Label",
"themes":"Theme list",
"annex":"Annex",
"status":"Status",
...
"layer-label":"Layer label",
"contact":"Contact",
"layers":"Layers",
"registers":"Registers",
"extensible-none-label":"Not extensible",
"results":"Results",
"latestversion":"Latest version",
"ec":"European Commission",
...
}
```

6.2.2.2. MODE 1 descriptor - Registry page

The color used in the example below are used to match elements between files.

Mode descriptor (mode.descriptor.json)

```
"model":{
  "http://localhost/registry":{
    "id":"register",
    "listtitlekey":"registers",
    "tablecolumns":[
      {
        "labelkey":"label",
        "itemkey":"label->text",
        "href":"id"
      }
    ]
  }
}
```

Related JSON data file (registry.en.json)

```
{
  "registry":{
    "id":"http://localhost/registry",
    "language":"en",
    "label":{
      "lang":"en",
      "text":"INSPIRE registry"
    },
    "registrymanager":"European Commission, Joint Research Centre",
    "contentsummary":{
      "lang":"en",
      "text":"The INSPIRE infrastructure involves ..."
    },
    "registers":[
      {
        "register":{
          "id":"http://localhost/theme",
          "label":{
            "lang":"en",
            "text":"INSPIRE theme register"
          }
        }
      },
      ...
    ]
  }
}
```

As you can see, each of the modes uses the **page URL (registry URI)** as identifier, this way, the web application knows, for each of the paged viewed, which mode apply to the web page.

The URL and all of the other properties shall have exactly the same name of the fields contained in the JSON format or in the GUI localisation files. In the following table, the elements are explained.

| Descriptor element | JSON element | file UI translation key | Description |
|----------------------------|---------------------------------|-------------------------|---|
| "id":"registrer" | registry.registers .register | | It identifies the main element name contained in the JSON list of items |
| "listtitlekey":"registers" | | registers | This is the key of the string of the table's title |

| | | | |
|--------------------------------------|--|-------|---|
| "tablecolumns" | | | Defines the column to be displayed in the table containing the list of elements |
| "tablecolumns.labelkey":"label" | | label | This defines the id of the string translation available in the UI localization file |
| "tablecolumns.itemkey":"label->text" | registry.registers.register.label.text | | This property is the reference to the JSON data element to be displayed |
| "tablecolumns.href":"id" | registry.registers.register.id | | This defines the link associated to the label described in the previous row. |

6.2.2.3. MODE 2 descriptor - Register page

The color used in the example below are used to match elements between files.

```
"mode2":{
  "http://localhost/theme":{
    "id":"theme",
    "listtitlekey":"themes",
    "tablecolumns":[
      {
        "labelkey":"label",
        "itemkey":"label->text",
        "href":"id"
      },
      {
        "labelkey":"annex",
        "itemkey":"annex"
      },
      {
        "labelkey":"status",
        "itemkey":"status->label->text",
        "href":"status->id"
      }
    ]
  },
  ...
}
```

Related JSON data file ([theme.en.json](#))

```
{
  "register":{
    "id":"http://localhost/theme",
    "language":"en",
    "label":{
      "lang":"en",
      "text":"INSPIRE theme register"
    },
    "contentsummary":{
      "lang":"en",
      "text":"The INSPIRE theme register contains all spatial data themes, as defined in the Annexes of the INSPIRE Directive ..."
    },
  },
}
```

```

"registerowner":"European Union",
"registermanager":"European Commission, Joint Research Centre",
"registercontrolbody":"INSPIRE Maintenance and Implementation Group
(MIG)",
"submitter":"Members of INSPIRE Maintenance and Implementation Group
(MIG)",
"contactpoint":{
  "label":"JRC INSPIRE Registry Team",
  "email":"inspire-registry-dev@jrc.ec.europa.eu"
},
"license":{
  "label":"Europa Legal Notice",
  "uri":"http://ec.europa.eu/geninfo/legal_notices_en.htm"
},
"registry":{
  "id":"http://localhost/registry",
  "label":{
    "lang":"en",
    "text":"INSPIRE registry"
  }
},
"containeditems":[
  {
    "theme":{
      "id":"http://localhost/theme/ad",
      "version":"0",
      "annex":"I",
      "label":{
        "lang":"en",
        "text":"Addresses"
      },
      "definition":{
        "lang":"en",
        "text":"Location of properties based on address
identifiers, usually by road name, house number, postal code."
      },
      "description":{
        "lang":"en",
        "text":"An address is an identification of the
fixed location of a property. The full address is ..."
      },
      "itemclass":{
        "id":" Theme",
        "label":{
          "lang":"en",
          "text":"Theme"
        }
      },
      "status":{
        "id":"http://localhost/registry/status/valid",
        "label":{
          "lang":"en",
          "text":"Valid"
        }
      },
      "register":{

```

```

        "id": "http://localhost/theme",
        "label": {
            "lang": "en",
            "text": "INSPIRE theme register"
        },
        "registry": {
            "id": "http://localhost/registry",
            "label": {
                "lang": "en",
                "text": "INSPIRE registry"
            }
        }
    }
}
},
...

```

Each of the modes use the [page URL \(register URI\)](#) as identifier. In the `mode 2` descriptor, there is one descriptor for each of the registers.

In the example above there is the theme register as example.

The URL and all of the other properties shall have exactly the same name of the fields contained in the JSON format or in the GUI localization files.

In the following table, each of the element is explained.

| Descriptor element | JSON file element | UI translation key | Description |
|---------------------------------------|--|--------------------|--|
| "id": "theme" | register.containeditems. theme | | It identifies the main element name contained in the JSON list of items (in this case the theme) |
| "listtitlekey": "themes" | | themes | This is the key of the string of the table's title |
| "tablecolumns" | | | Defines the column to be displayed in the table containing the list of elements |
| "tablecolumns.labelkey": "label" | | label | This defines the id of the string translation available in the UI localization file |
| "tablecolumns.itemkey": "label->text" | register.containeditem.t heme.label.t ext | | This property is the reference to the JSON data element to be displayed |
| "tablecolumns.href": "id" | register.containeditem.t heme.id | | This defines the link associated to the label described in the previous row. Since the last JSON data item was inside the label->text element, we use here the parent:id to refer to the theme.id element. |

| | | | |
|--|--|--------|---|
| "tablecolumns.labelkey":"annex" | | annex | This defines the id of the string translation available in the UI localization file |
| "tablecolumns.itemkey":"annex" | register.containeditem.theme.annex | | This property is the reference to the JSON data element to be displayed |
| "tablecolumns.labelkey":"status" | | status | This defines the id of the string translation available in the UI localization file |
| "tablecolumns.itemkey":"status->label->text" | register.containeditem.theme.status.label.text | | This property is the reference to the JSON data element to be displayed |
| "tablecolumns.href":"status->id" | | | This defines the link associated to the label described in the previous row. |

In some cases, a field could be a multi-value field. In this case, the JSON descriptor is represented in the following way. You can find a complete example in the mode descriptor file -> mode 2 -> code list.

```

...
"itemkey":"parents=parent->label->text",
...
The related JSON data file is
...
"parents": [
{
  "parent": {
    "id": "http://localhost/codelist/DesignationValue",
    "label": {
      "lang": "en",
      "text": "Designation"
    }
  }
}
],
...

```

6.2.2.4. MODE 3 descriptor - Item detail page

```

"mode3": {
  "theme": {
    "id": "theme",
    "detailelements": [
      {
        "labelkey": "annex",
        "itemkey": "annex",
        "topseparator": "true",
        "bold": "false"
      }
    ]
  }
}

```

```
    ],
  },
  ...

```

Related JSON data file (theme.en.json)

```
{
  "theme":{
    "id":"http://localhost/theme/ad",
    "thisversion":"http:// localhost /theme/ad:1",
    "latestversion":"http:// localhost /theme/ad",
    "language":"en",
    "annex":"I",
    "label":{
      "lang":"en",
      "text":"Addresses"
    },
    "definition":{
      "lang":"en",
      "text":"Location of properties based on address identifiers, usually
by road name, house number, postal code."
    },
    "description":{
      "lang":"en",
      "text":"An address is an identification of the fixed location of a
property. The full address is a ..."
    },
    "governance-level":{
      "id":"http://localhost/registry/governance-level/eu-legal",
      "label":{
        "lang":"en",
        "text":"eu-legal"
      }
    },
    "itemclass":{
      "id":"Theme",
      "label":{
        "lang":"en",
        "text":"Theme"
      }
    },
    "status":{
      "id":" http://localhost/registry/status/valid",
      "label":{
        "lang":"en",
        "text":"Valid"
      }
    },
    "register":{
      "id":"http://localhost/theme",
      "label":{
        "lang":"en",
        "text":"INSPIRE theme register"
      },
      "registry":{
        "id":"http://localhost/registry",

```



```

        "label":{
            "lang":"en",
            "text":"INSPIRE registry"
        }
    }
}
}
}
}

```

Each of the modes use the **itemclass** as identifier. In the mode 3 descriptor, there is one descriptor for each of item details page.

In the example above the theme register has been used.

The itemclass and all of the other properties shall have exactly the same name of the fields contained in the JSON format or in the GUI localization files. In the following table, each of the element is explained.

| Descriptor element | JSON element | file | UI translation key | Description |
|-------------------------------------|--------------|------|--------------------|--|
| "detailedelements" | | | | This is the container of all the field to be displayed in the HTML view. The standard fields are always visible in the HTML. If you have additional fields, you can add it using this element. |
| "detailedelements.labelkey":"annex" | | | annex | This defines the id of the string translation available in the UI localization file |
| "detailedelements.itemkey":"annex" | theme.annex | | | This property is the reference to the JSON data element to be displayed |
| "topseparator":"true" | | | | This property add a separator between one element and eanother in the HTML detail page. |
| "bold":"false" | | | | This property render the text related to this field in bold. |

In case the element has a hierarchy, like the code list detail page, some additional elements are added into the descriptor of this mode (a full example can be found on the mode 3 descriptor file -> code list).

The additional elements are:

- `tablecolumns` (already described in the tables above): Defines the column to be displayed in the table containing the list of elements in the hierarchy (for example, in case the detail page is a code list, the table will contain the code-values)
- `relationtablecolumns`: Defines the column to be displayed in the table containing the list of elements with some relation (for example the list of parent of that specific item).

Example of additional fields in mode 3 for the hierarchical elements:

```

...
"tablecolumns":[
  {
    "labelkey":"label",
    "itemkey":"label->text",
    "href":"id"
  },
  {
    "labelkey":"status",
    "itemkey":"status->label->text",
    "href":"status->id"
  },
  {
    "labelkey":"governance-level-label",
    "itemkey":"governance-level->label->text",
    "href":"governance-level->id"
  }
],
"relationlisttitlekey":"codelists",
"relationlistkey":"codelist",
"relationtablecolumns":[
  {
    "labelkey":"label",
    "itemkey":"label->text",
    "href":"id"
  },
  {
    "labelkey":"themes",
    "itemkey":"themes=theme->label->text",
    "href":"theme->id"
  },
  {
    "labelkey":"application-schema",
    "itemkey":"applicationschema->label->text",
    "href":"applicationschema->id"
  },
  {
    "labelkey":"status",
    "itemkey":"status->label->text",
    "href":"status->id"
  }
]
...

```

6.2.2.5. MODE 4 Descriptor - Item detail for hierarchical elements

```

"mode4":{
  "CodeListValue":{
    "id":"value",
    "collectionelementid":"codelist",
    "detailedelements":[
      {
        "labelkey":"themes",
        "itemkey":"themes=theme->label->text",
        "topseparator":"true",
        "href":"id",
        "bold":"false"
      },
      {
        "labelkey":"application-schema",
        "itemkey":"applicationschema->label->text",
        "topseparator":"false",
        "href":"id",
        "bold":"false"
      },
      {
        "labelkey":"code-list",
        "itemkey":"codelist->label->text",
        "topseparator":"false",
        "href":"id",
        "bold":"false"
      }
    ],
    "listtitlekey":"narrower",
    "tablecolumns":[
      {
        "labelkey":"label",
        "itemkey":"label->text",
        "href":"parent:id"
      },
      {
        "labelkey":"status",
        "itemkey":"status->label->text",
        "href":"id"
      },
      {
        "labelkey":"governance-level-label",
        "itemkey":"governance-level->label->text",
        "href":"id"
      }
    ]
  },
  ...
}

```

Related JSON data file

<http://inspire.ec.europa.eu/codelist/AccessRestrictionValue/forbiddenLegally/forbiddenLegally.en.json>

```

{
  "value":{
    "id":"http://localhost/codelist/AccessRestrictionValue/forbiddenLegally",
    "thisversion":"http://localhost/codelist/AccessRestrictionValue/forbiddenLegally"
  }
}

```

```
lly:1",
"latestversion":"http://localhost/codelist/AccessRestrictionValue/forbiddenLe
gally",
"language":"en",
  "label":{
    "lang":"en",
    "text":"forbidden legally"
  },
"definition":{
  "lang":"en",
  "text":"Access to the transport element is forbidden by law."
},
"itemclass":{
  "id":"CodeListValue",
  "label":{
    "lang":"en",
    "text":"Code list value"
  }
},
"status":{
  "id":" http://localhost/registry/status/valid",
  "label":{
    "lang":"en",
    "text":"Valid"
  }
},
"register":{
  "id":"http://localhost/codelist",
  "label":{
    "lang":"en",
    "text":"INSPIRE code list register"
  },
  "registry":{
    "id":"http://localhost/registry",
    "label":{
      "lang":"en",
      "text":"INSPIRE registry"
    }
  }
},
"governance-level":{
  "id":"http://localhost/registry/governance-level/eu-legal",
  "label":{
    "lang":"en",
    "text":"eu-legal"
  }
},
"themes":[
  {"theme":{
    "id":"http://localhost/theme/tn",
    "label":{
      "lang":"en",
      "text":"Transport networks"
    }
  }
}
]
```

```

],
"applicationschema":{
  "id":"http://localhost/applicationschema/tn",
  "label":{
    "lang":"en",
    "text":"Common Transport Elements"
  }
},
"odelist":{
  "id":"http://localhost/odelist/AccessRestrictionValue",
  "label":{
    "lang":"en",
    "text":"Access Restriction"
  }
}
}
}
}

```

The mode 4 has basically the elements already described in the previous pages. In this case the table columns element in the mode descriptor describes the column for the narrower table (the narrower table is visible only if the item has narrower items).

6.2.3.Static pages

The static pages are those pages that have a fixed layout and present a list of elements that are not available in the data files. It could be seen as a kind of ‘service list’.

An example of static page could be the ‘Status’ page of the INSPIRE registry (<http://localhost/registry/status>).

The files describing the static pages are stored in the ‘<root_folder>/Project_package_1.X/examples/<selected-exampe>/webapp-configurations/app_data/staticpages’ folder.

Each of the static pages available in the system has one JSON descriptor file.

To add a static pages just add the related descriptor file. The file name pattern is <page_name>.descriptor.json. In the following example, there is the "Status" static page, available in the INSPIRE registry.

6.2.3.1. Static page example: status.descriptor.json

```

{
  "id":"http://localhost/registry/status",
  "labelkey":"status",
  "descriptionkey":"status-definition",
  "listtitlekey":"values",
  "types":{
    "invalid":{
      "id":"http://localhost/registry/status/invalid",
      "labelkey":"invalid",
      "descriptionkey":"invalid-desc"
    }
  }
}

```

```

    },
    "valid":{
        "id":"http://localhost/registry/status/valid",
        "labelkey":"valid",
        "descriptionkey":"valid-desc"
    },
    "superseded":{
        "id":"http://localhost/registry/status/superseded",
        "labelkey":"superseded",
        "descriptionkey":"superseded-desc"
    },
    "retired":{
        "id":"http://localhost/registry/status/retired",
        "labelkey":"retired",
        "descriptionkey":"retired-desc"
    },
    "submitted":{
        "id":"http://localhost/registry/status/submitted",
        "labelkey":"submitted",
        "descriptionkey":"submitted-desc"
    }
}
}
}

```

| Field | Description |
|----------------|---|
| id | The id of the element shall be the URL of the page |
| labelkey | The key of the translated label contained in the localization files. |
| descriptionkey | The key of the translated description contained in the localization files. |
| listtitlekey | The key of the title for the table contained in the localization files. |
| types | This field contains a list of items, one for each of the values available in the page. For each type, the key of the list item is the one used to compose the URL. |

Once this file has been created, in order to correctly access the static page, the Apache configuration file has to be updated by adding the URL rewrite rule for the newly created page. An example of URL rewrite configuration for the 'Status' static page is available in Figure 69.

Figure 69: Example of configuration string for the 'Status' static page

```

# Service pages
# Status
RewriteRule ^/registry/status$ /webapp/index.php?static=status [L]
RewriteRule ^/registry/status/$ /webapp/index.php?static=status [L]
RewriteRule

```

```
^/registry/status/ (valid|invalid|retired|submitted|superseded) $ /webapp/index.php?static=status&type=$1 [L]
RewriteRule
^/registry/status/ (valid|invalid|retired|submitted|superseded) /$ /webapp/index.php?static=status&type=$1 [L]
```

6.2.4. Customised pages

The customised pages can contain any type of content; an example of those pages could be the common 'help' and 'about' page.

The files describing the custom pages are stored in the folder

```
<root_folder>/Project_package_1.X/examples/<selected-exampe>/webapp-configurations/app_data/custompage
```

. Inside that folder, there is one folder for each of the custom pages available in the system.

Inside the specific folder, there are two files:

- `descriptor.json`: it is the descriptor of the page
- `en.php`: this file contains the contents (HTML code is allowed) that will be included in the page.

The name of the folder shall be exactly the name used in the URL. The following example (about page), shows a correct folder structure:

```
webapp/app_data/custompages/about/descriptor.json
webapp/app_data/custompages/about/en.php
```

6.2.4.1. Example custom page descriptor

```
{
  "id": "http://localhost/registry/about",
  "labelkey": "about-title"
}
```

The `descriptor.json` file contains the id of the element that shall be the URL of the target page. The `labelkey` contains the key to the page title available in the localization files (describe above at section).

The `en.php` file is filled with the HTML to create the custom page.

6.2.5. Website parts

The folder `<root_folder>/Project_package_1.X/examples/<selected-exampe>/webapp-configurations/app_data/parts'` contains the mapping files between some parts of the webapp and the localization file.

Currently the webapp has only the file `footer.json` that represents the label-key (pointing to the localization file) and links of each footer menu entry. If you want to customize the footer, you can just edit the entry in this file (and in the localization files).

7. Developing the Re3gistry

This guide is especially devoted to those developers that may want to reuse the **Re3gistry** software source code.

7.1. Technology

The **Re3gistry** software requires **Java 1.7** or **Java 1.8** technology.

The libraries used in the system are described in the Project Object Model (POM) file.

This project is built using **Apache Maven** technology.

The php web application requires **php >= 5.4**.

7.1.1. Web Server

The **Re3gistry** needs two different web servers: one to serve the RESTful web service and the other is a servlet container for the **Re3gistry** Java webapp.

The two servers used are:

- Apache HTTPD Version 2.4 [APACHE-HTTPD]
- Apache Tomcat Version 7 or Apache Tomcat 8 [APACHE-TOMCAT]

7.1.2. 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**

7.2. System structure

The **Re3gistry** software is organised in modules. This concept allows a simpler customisation and extension of the system.

The main components of the system are the

- `Re3gistryCommon` module and
- The `Re3gistry` webapp.

The coming paragraphs describe the main structure of the software package and its modules.

7.2.1. Module concept

A module is a Java library that is usually composed of a **core part** (Java library) but there may also be accompanied of a **presentation part** (web pages).

- The module's core part contains all the logic related to its functionality. Each module needs to include the `Re3gistryCore` module in order to access all the common libraries and functionalities.
- The module's presentation part is optional (you can access the module's core functionalities by customising the standard interface provided by the `Re3gistry`) and it is contained in the module's folder of the `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.

7.2.2. `Re3gistryCommon` module

The `Re3gistryCommon` module is the basis of whole 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 not a related presentation part in the `Re3gistry webapp` since its function is to provide the object structure and the main functionalities to the other modules.

The `Re3gistryCommon` library is composed of different packages:

- `Constants`: containing a class with all the constants used in this library;
- `Managers`: containing all the methods to access the data for each bean defined in the model package;
- `Model`: containing all the beans related to the information model;
- `Utils`: containing several general utility classes, such as `StringUtils` or `Logger class`.

7.2.3. `Re3gistry` software interface

The `Re3gistry` software contains the web interface to manage and use the functionalities provided by each module. In this component, all the configuration files are stored, both related to the core module and to each of the additional module.

Below there is a description of the components (organised by folders) contained in the `Re3gistry webapp`.

- `Web Pages`: contains the files related to the web pages (jsp, css, js, images etc.). The 'modules' folder, inside the 'Web Pages' one, contains specific web pages and configuration files.
- `Source`: contains the Java class related to the `Re3gistry` webapp (constants, module manager functionalities, servlets, utility, etc.).
- `Resources`: contains the configurations and localization files. Below there is a description for each file within it:
 - `META-INF/persistence.xml`: file with the configurations related to the database

- `configurations/Application.properties`: file containing the main **Re3gistry** configurations (common to all the modules);
- `configurations/modules`: containing the properties related to each module. The file name shall be exactly as the name of the module's folder.
- `configurations/logcfg.xml`: 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 also a subfolder named '*Application*' containing 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.

7.3. Source code

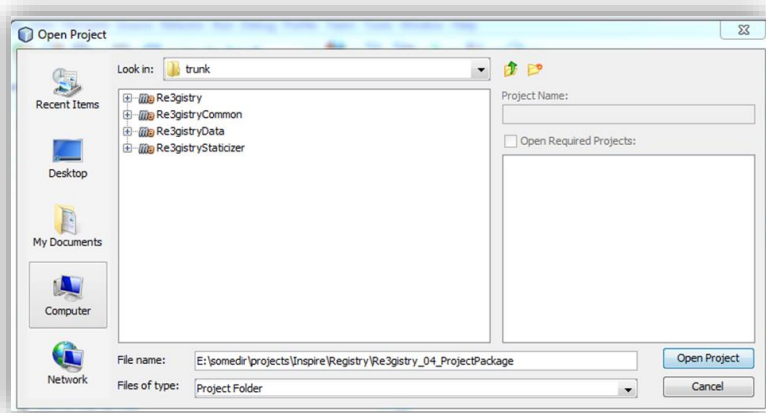
The following section will show how to install and run the **Re3gistry** using the *NetBeans IDE* step by step [NETBEANS].

We have chosen NetBeans¹⁶ for this documentation because it is the official open source IDE provided by Java.

7.3.1. Load projects

The first step is to load the projects contained in the '`source`' folder of the package using NetBeans IDE, as shown in Figure 68. The project is available at:

```
<root_folder>/Project_package_1.X/source
```



7.3.2. Configuration files

Figure 70: Loading the project with NetBeans IDE

¹⁶ <https://netbeans.org/>

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, See an example of a POM file in Figure 71.

Edit these configuration settings using the local settings:

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

To customise 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 update 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 smtp 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 languages 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.

Figure 71: Example POM file

```
<profiles>
  <profile>
    <id>env-dev1</id>
    <activation>
      <property>
        <name>env</name>
        <value>dev1</value>
      </property>
    </activation>
  </profile>
</profiles>
```

```

    <properties>
      <!-- Loggers configurations -->
<Default.log.file.dir><ROOT_FOLDER>/Re3gistry-
data/logs/Complete.log</Default.log.file.dir>
<Re3gistry.log.file.dir><ROOT_FOLDER>/Re3gistry-
data/logs/Re3gistry.log</Re3gistry.log.file.dir>
<Re3gistryData.log.file.dir><ROOT_FOLDER>/Re3gistry-
data/logs/Re3gistryData.log</Re3gistryData.log.file.dir>
<Re3gistryStaticizer.log.file.dir><ROOT_FOLDER>/Re3gistry-
data/logs/Re3gistryStaticizer.log</Re3gistryStaticizer.log.file.dir>

      <!-- Database configurations -->
      <persistence.jdbc.url>jdbc:postgresql://db_ip:5432/inspire_regi
stry </persistence.jdbc.url>
      <persistence.jdbc.driver>org.postgresql.Driver</persistence.
jdbc.driver>
      <persistence.jdbc.username>db_username</persistence.jdbc.userna
me>
      <persistence.jdbc.password>db_password</persistence.jdbc.passwo
rd>

      <!-- Specific module configurations -->

      <!-- RegistryData -->
<RegistryData.import.dir><root_folder>/Re3gistry-
data/temp</RegistryData.import.dir>
      <!-- / -->

      <!-- RegistryStaticizer -->
<!-- The root folder where to save the files produced by the Staticizer. ->
<RegistryStaticizer.export.dir><ROOT_FOLDER>/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><ROOT_FOLDER>/Re3gistry-
data/xsl</RegistryStaticizer.xsl.dir>

    </properties>
  </profile>
</profiles>

```

Before 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').

7.3.3. Choose the authentication method

In order to choose the desired authentication method, two configuration files have to be updated. Below you can find a reference for both files in each case.

- `source/Re3gistry/src/main/resources/Application.properties`
- `source/Re3gistry/src/main/webapp/WEB-INF/web.xml`

The `Application.properties` file contains the property that allows the switch between ECAS or SHIRO authentication method.

The default configuration is Apache SHIRO as authentication method.

```
# Login type: SHIRO | ECAS
application.LoginType=SHIRO
In addition, the web.xml file has to be updated based on the authentication
method selected.
For example, the default configuration is Apache SHIRO: the SHIRO related
lines are uncommented and the ECAS lines are commented:
<!--SHIRO authentication configs-->
    <listener>
        <listener-
class>org.apache.shiro.web.env.EnvironmentLoaderListener</listener-class>
    </listener>

    <filter>
        <filter-name>ShiroFilter</filter-name>
        <filter-class>org.apache.shiro.web.servlet.ShiroFilter</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>ShiroFilter</filter-name>
        <url-pattern>/*</url-pattern>
        <dispatcher>REQUEST</dispatcher>
        <dispatcher>FORWARD</dispatcher>
        <dispatcher>INCLUDE</dispatcher>
        <dispatcher>ERROR</dispatcher>
    </filter-mapping>
```

```
<!--END SHIRO authentication configs-->

<!--ECAS authentication configs-->
  <!--
  <login-config>
    <auth-method>ECAS</auth-method>
    <realm-name>Re3gistry Realm</realm-name>
  </login-config>

  <security-constraint>
    <web-resource-collection>
      <web-resource-name>not protected content</web-resource-name>
      <url-pattern>/res/*</url-pattern>
      <url-pattern>/ChangeLocale</url-pattern>
      <url-pattern>/login</url-pattern>
      <url-pattern>/login.jsp</url-pattern>
    </web-resource-collection>
  </security-constraint>
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>inspire-regadmin</web-resource-name>
      <description>Requires users to be authenticated but does not
require them to be authorized</description>
      <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>*</role-name>
    </auth-constraint>
    <user-data-constraint>
      <description>Encryption is not required for the application in
general.</description>
      <transport-guarantee>NONE</transport-guarantee>
    </user-data-constraint>
  </security-constraint>
  -->

<!--END ECAS authentication configs-->
```

If the chosen authentication method is ECAS, comment the SHIRO configuration lines and uncomment the ECAS ones.

To use the ECAS authentication method, follow the ECAS installation guide at https://ies-vn.jrc.ec.europa.eu/projects/registry-development/wiki/Re3gistry_ECAS_install.

7.3.4. Database creation and initialisation

The database initialisation procedure creates the structure of the database and populate the registry contents. Execute the database initialisation using the example SQL scripts available in the project's package `'example/<chosen_example>/database'` folder.

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

The first step is to create a database. After the database has been created, the initialisation script can be executed.

Open the `create-tables.sql` into an SQL editor and run the queries.

Do the same for the `database-initialization.sql` and `database-localization.sql`. It is important to run the `create-tables.sql` before the any other scripts. The `database-initialization.sql` and `database-localization.sql` files contain some sample data.

If the clean-up of the database is needed, in the example packages, there is also a `drop-tables.sql` script.

For each new register to be added, the register table, the itemclass table and the localisation table should all be filled in; for more information related to the creation of new register, please refer to the customization guide at .

7.3.5. Build projects

To get the system running, build each project and start the web application. To build the project, right click on the project name and select `'build'` or `'clean and build'`.

7.3.6. Creating new modules

To start creating a new module, it is important to understand the module's structure by reading the chapter and taking a deep look at one of the modules included in the package (for example the `Data module` or the `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.

7.3.6.1. Step 1

Open the main project contained in the project package (`project_package_folder/source`).

7.3.6.2. Step 2

Create a new project by right clicking on the project browser and click on 'new project' as shown in Figure 72.

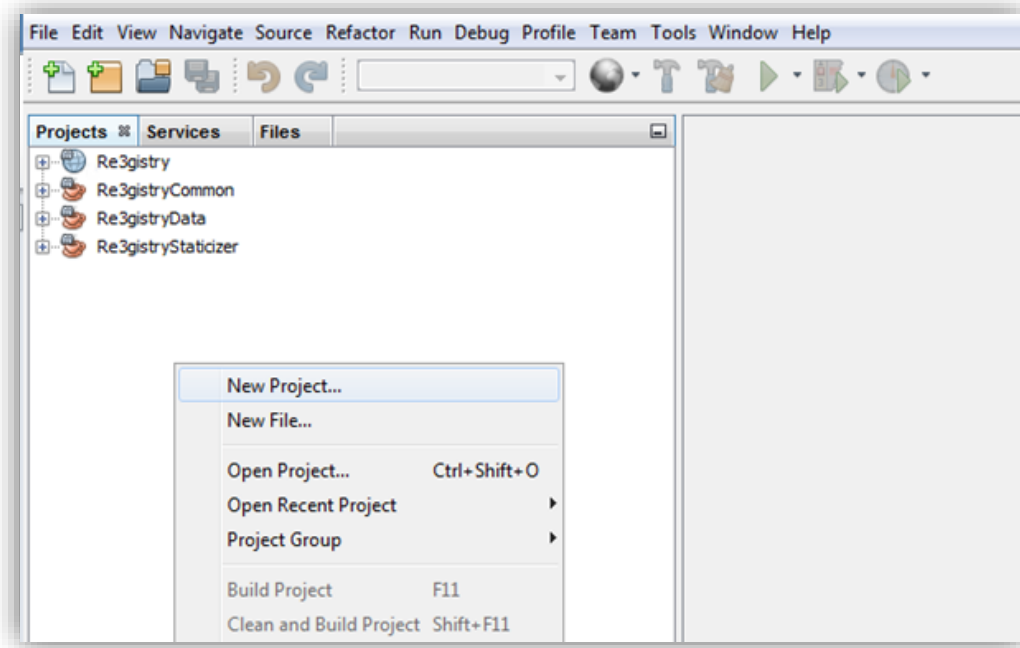


Figure 72: Creating new project from the project browser

Choose 'Maven' -> 'Java Application' in the opened window, and then click the 'next' button (See Figure 73).

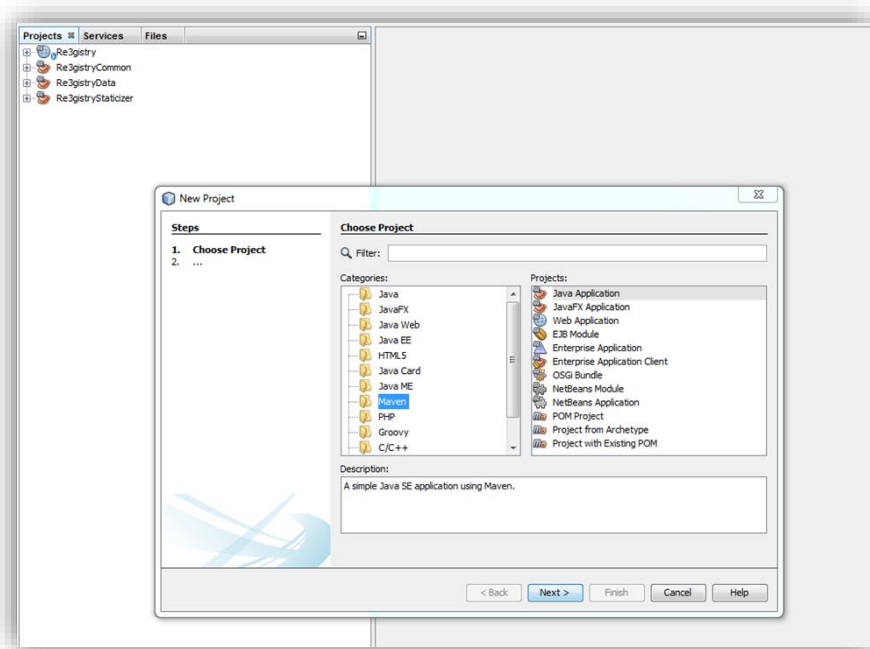


Figure 73: Project type selection

In the following window (See Figure 74), give a name to the new module; in this example the 'NewModule' name is used.

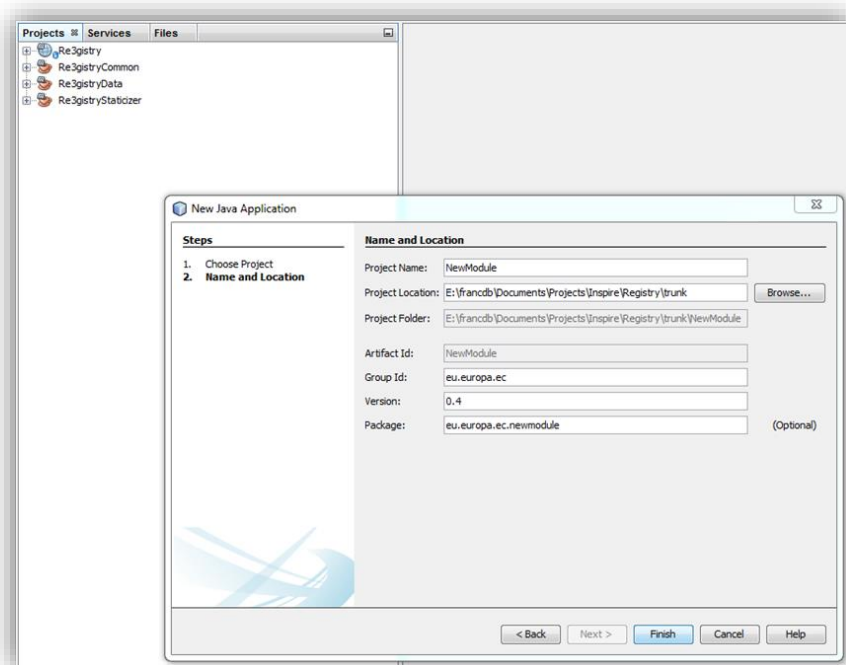


Figure 74: Module name

7.3.6.3. Step 3

Go to the newly created project and edit the POM file, to insert the dependency to the `RegistryCommon` module (See Figure 75).

Figure 75: POM edit example

```
<dependency>
    <groupId>eu.europa.ec.Re3gistrycommon</groupId>
    <artifactId>Re3gistryCommon</artifactId>
    <version>x.x</version>
</dependency>
<!-- Replace x.x with the right version of the module -->
```

7.3.6.4. 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 module.

This class shall contain at least the class name (See Figure 76).

Figure 76: 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";

}
```

7.3.6.5. Step 5

If the new module requires a dedicated interface, the user shall create it using the following instructions. Otherwise, you can jump to Step 6.

Open the Web Pages folder in the Registry webapp project. Search the modules folder, right click on it and select 'New' -> 'Folder' (See Figure 77). Name the new folder exactly as the previously created Java Application.

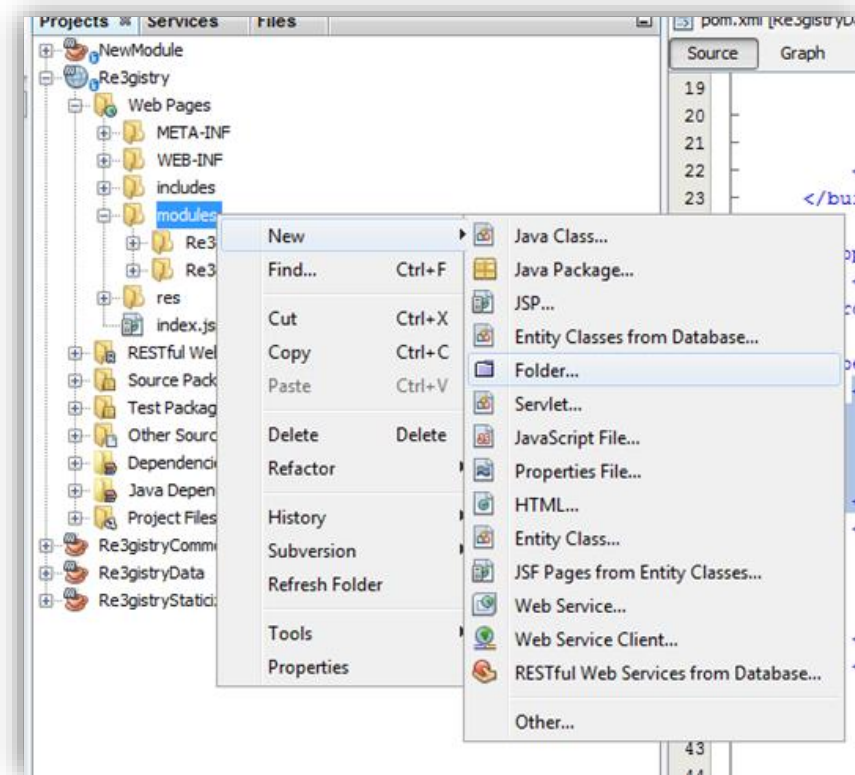


Figure 77: Creation of the module's folder

7.3.6.6. Step 6

Create the module's property file, inside the 'other sources - configurations/modules' project folder.

This folder is located under Re3gistry/WEB-INF/classes/configurations/modules) with the same module name 'NewModule.properties', see Figure 78.

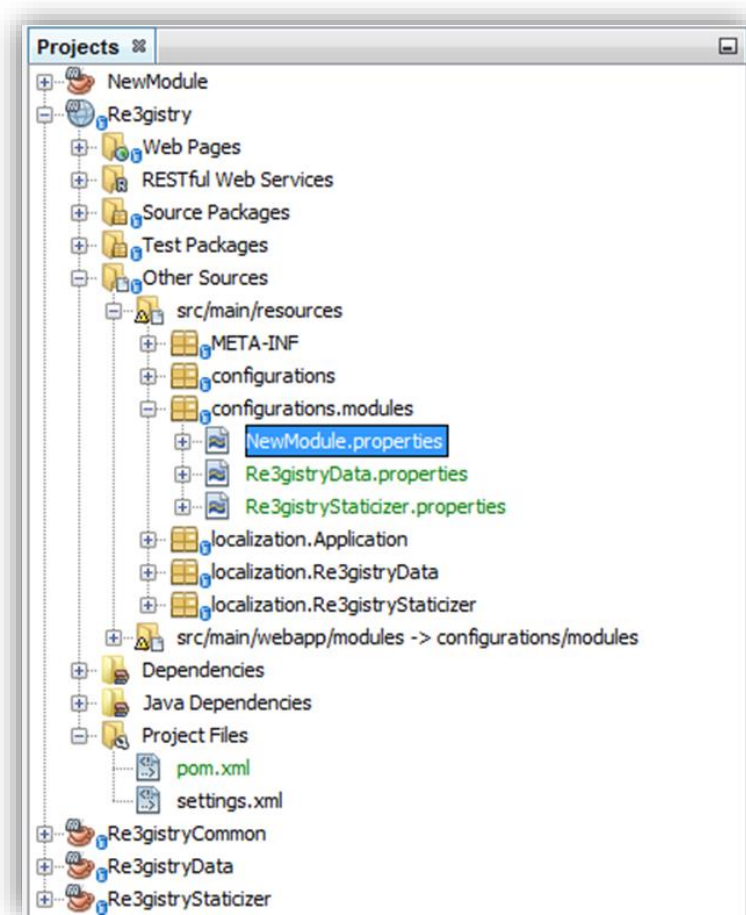


Figure 78: Module's properties file

The required property keys for all the new modules are the following:

- **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. If no module has this property as visible, the menu bar is not visible.
- **module.menuitem.label=Data** – This property represents the label of the menu item in the system main menu.
- **module.homewidget.enabled=true** – This property enables/disables the system to show the module widget in the home page.
- **module.dateformat=dd-MM-yyyy HH:mm:ss** – The module date/time format.
- **module.menuitem.order=2** – The order in which this menu item is shown in the main menu. The higher values will be displayed later.

7.3.6.7. 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' (See Figure 79) .

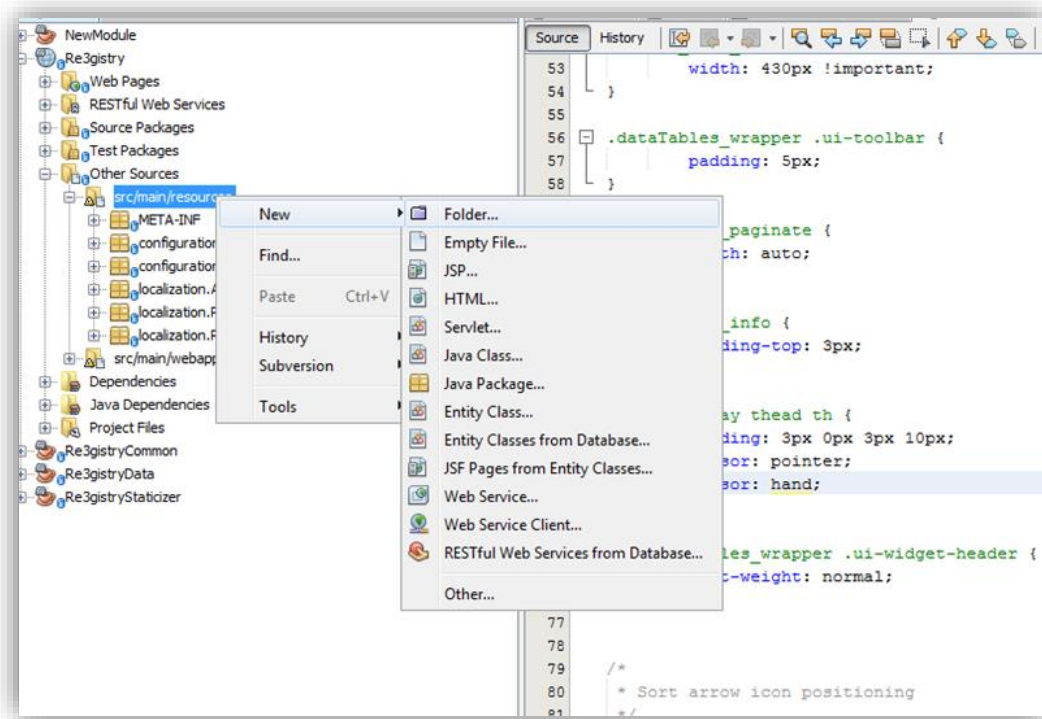


Figure 79: Localization folder for the new module

Name the new folder 'localization/NewModule' (See Figure 80).

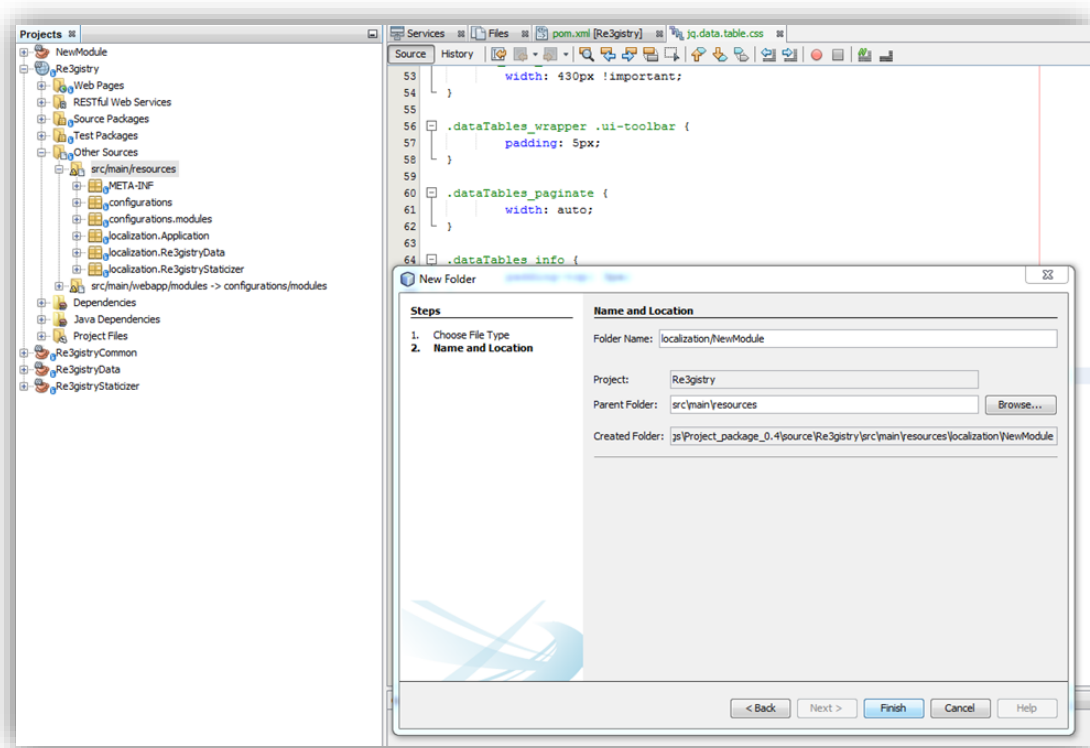


Figure 80: Localization properties folder

Then create a file for each language using the following naming pattern (See Figure 79):

```
'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

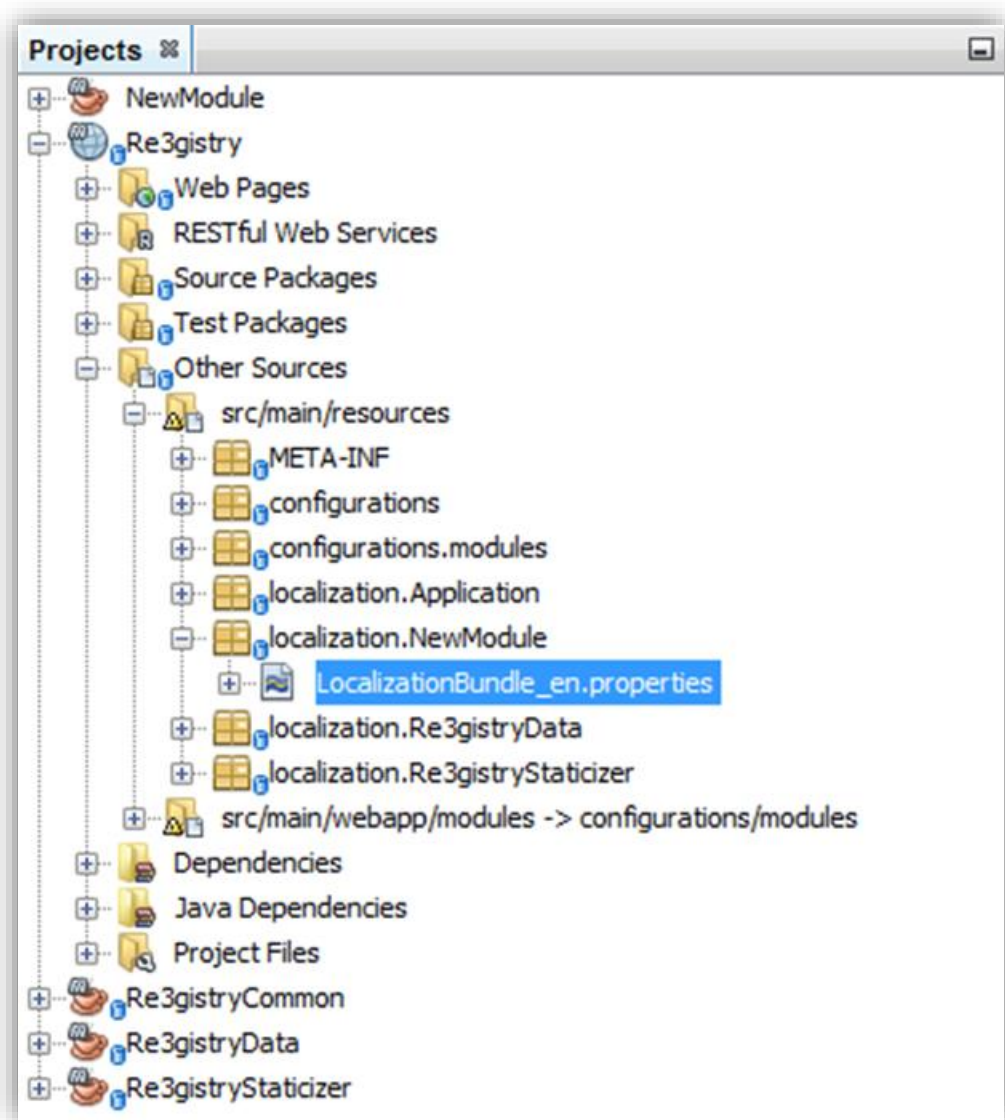


Figure 81: Module's localization properties file

7.3.6.8. 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.

Figure 82: 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>
```

7.3.6.9. Step 9

The module structure is now complete. 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 .

There are two 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.

The main structure of the modules can be understood by having 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 localisation 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).

Index of keywords

A

Addition, 23, 25, 26, 31
Additional fields, 23
administration panel, 19, 36, 48, 49, 54, 63, 64, 65
Apache SHIRO, 45
Application.properties, 41, 42, 45, 46, 101, 102, 104
authentication method, 45
Autocomplete, 67

B

base URI, 45
Build projects, 106

C

changelog, 15
Clarification, 23, 26, 27, 31
collection, 17, 24, 26, 28, 29, 30, 31, 32, 56, 63, 64, 65, 78, 105
conf.php, 59, 64, 66
configuration file, 44, 45, 57, 58, 60, 61, 66, 83, 96, 100
contact point, 69
content-negotiation, 12, 34, 55, 56, 57
core.properties, 63
costumisation, 33
costumised registers, 71
costumised registry, 68
create-table.sql, 39
create-tables.sql, 39, 68, 106
CSV, 11, 13, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 32
customisation, 62, 66, 99
customised attributes, 24
Customised attributes, 19

D

data consistency, 21
data localisation, 19
data management module, 20
Data management module, 21
data staticization module, 20
data storage, 32
Database, 19, 38, 50, 99, 103, 106
database-initialization.sql, 39, 68, 106
database-localization.sql, 39, 73, 106
deployer module, 20, 35, 79
descriptor, 25, 26, 27, 67, 83, 84, 86, 88, 89, 91, 95, 97
descriptor file, 67, 89, 91, 95
download, 13
drop-table.sql, 39

E

ECAS, 11, 13, 37, 38, 45, 46, 104, 105, 106
email notification, 32, 52
endpoint, 64
Errors, 30
external items, 15
externally governed items, 23
Externally governed items, 23

F

federation, 13, 15, 67
feedback, 13

G

GUI, 19
gui-languages, 34

H

HTTP header, 56
HTTP server, 60

I

ignore warning, 30, 32, 50
import, 13, 17, 19, 20, 21, 22, 30, 32, 36, 43, 50, 51, 74, 77, 78, 102, 103
import data file, 19, 21
Import data file, 22, 74
index, 15, 24, 34, 37, 64, 97
indexing, 34
information model, 16
INSPIRE registry, 12, 15, 17, 19, 38, 85, 87, 91, 94, 95
inspire-example, 38, 44, 50, 60, 61
Installing, 37, 58, 62
Internal items, 23
Invalid, 18, 74
Invalidation, 23, 28, 31
item, 17
item keys, 34
itemclass, 17, 22, 23, 25, 26, 27, 28, 29, 30, 33, 71, 72, 73, 74, 76, 78, 83, 87, 90, 91, 94, 106

L

language mapping file, 34
Language representations, 19
libraries, 38, 46, 62, 99, 100

License, 14
localisation, 19, 34, 73, 106
log, 25, 28, 29, 30, 43, 60, 103, 115
logcfg.xml, 41, 43, 101, 114
logger.xml, 59, 60
logging system, 60
logs, 60, 102, 103

M

Main fields, 23
mandatory fields, 23, 24
master language, 22, 70, 73
master xml files, 32
module, 20, 21, 24, 32, 34, 35, 57, 79, 99, 100, 101, 102, 103, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116
multilingualism, 19, 69, 73

N

neutral-example, 38, 44, 50, 52, 53, 60, 61, 66, 68, 76, 78, 81

O

open source, 13, 15, 101

P

persistence.xml, 42
POM, 11, 99, 102, 104, 109, 114
procedure status, 36

R

Re3gistry 1.3, 15
Re3gistryData.properties, 43
Re3gistryDeployer.properties, 44, 54
Re3gistryStaticizer.properties, 43
readme.md, 38
recursive, 28
register, 17
register extension, 23
registry, 16
Registry core module, 20, 21
RegistryData.properties, 42, 102
RegistryDeployer.properties, 42, 79
RegistryStaticizer.properties, 42, 102
relationship, 17, 72
RESTful, 11, 13, 20, 34, 55, 56, 99
RESTful web service, 20, 34, 55, 56, 99
Retired, 18, 74
Retirement, 23, 30, 31
RoR, 13, 15, 67
RSS, 44

Run full export, 52

S

schema.xml, 63
separator, 23
server, 12, 20, 35, 36, 41, 43, 53, 55, 56, 57, 58, 60, 61, 79, 102
SHIRO, 46
shiro.ini, 46, 49
solr, 34, 44, 53, 55, 62, 63, 64, 65, 102
solr.xml, 62, 63
source code, 99
SQL scripts, 38, 106
standard attributes, 18, 19
Staticisation, 32
status, 17, 18, 23, 25, 26, 36, 51, 52, 70, 71, 73, 74, 83, 84, 86, 87, 89, 90, 92, 93, 94, 95, 97
Submitted, 18, 74
successor, 27, 28, 29, 31, 77
Superseded, 18, 74
Supersession, 23, 27, 29, 31
system architecture, 20
System requirements, 37, 55

T

Tomcat, 12, 37, 41, 42, 43, 44, 45, 46, 47, 48, 62, 63, 99

U

URI, 11, 13, 15, 17, 23, 24, 26, 45, 57, 69, 71, 72, 79
uriname, 24, 33, 56, 69, 70, 71, 72
uuid, 69, 70, 71, 72

V

Valid, 18, 74, 87, 90, 94
var, 34, 57, 58, 59

W

Warnings, 30
web service, 13, 16, 19, 38, 55, 56, 57, 62, 65
web.xml, 46
webapp, 19

X

XSLT, 11, 32, 33, 34, 44, 52, 78

Z

ZIP, 21, 22