Data Dictionary

- Application architecture

Version management

|  |  |  |  |
| --- | --- | --- | --- |
| No | Date | Changes | Author |
| *0.1* | *24.03.2003* | *First Draft* | *J. Heinlaid* |
| *0.2* | *09.04.2003* | *Updated incorporating changes requested by project leader* | *J. Heinlaid* |
| *0.3* | *12.12.2003* | *Updates documenting developments done within SA-6* | *J. Heinlaid* |
| *0.4* | *02.04.2004* | *Documentation of changes since the end of SA-6* | *J. Heinlaid* |
| *0.5* | *09.11.2004* | *Documentation of DD releases 2.2, 2.3 and 2.4 in SA-7* | *J. Heinlaid* |
| *0.6* | *27.01.2007* | *Migrated to ODT* | *Søren Roug* |
| *0.61* | *13.12.2007* | *Copied XML-RPC methods from methods document* | *Søren Roug* |
| *0.7* | *11.09.2015* | *Re-modelling of the document to highlight the architectural aspects.* | *Michael Norén* |

Contents

[1 Introduction 3](#_Toc429734078)

[1.1 Scope 3](#_Toc429734079)

[2 Business requirements 4](#_Toc429734080)

[2.1 Business process support 4](#_Toc429734081)

[2.2 Supported user roles 4](#_Toc429734082)

[2.3 Functional requirements 5](#_Toc429734083)

[2.4 Information model 5](#_Toc429734084)

[3 Application architecture 6](#_Toc429734085)

[3.1 Overview 7](#_Toc429734086)

[3.2 General architecture information 7](#_Toc429734087)

[3.3 Presentation layer 7](#_Toc429734088)

[3.4 Business layer 8](#_Toc429734089)

[3.5 Service layer 8](#_Toc429734090)

[3.6 Data layer 10](#_Toc429734091)

[3.7 Security 28](#_Toc429734092)

[3.8 Logging and error management 29](#_Toc429734093)

[3.9 Collaboration with other software 29](#_Toc429734094)

[3.10 Source code structure 30](#_Toc429734095)

[4 Infrastructure architecture 32](#_Toc429734096)

[4.1 Clients 33](#_Toc429734097)

[4.2 The DD application 33](#_Toc429734098)

[4.3 External services 33](#_Toc429734099)

# Introduction

Data Dictionary (DD) is a web-based data semantics registry for the Reportnet. It holds definitions of datasets, tables and data elements exchanged over the Reportnet, whether by humans or applications. These are called data definitions and DD provides functions to import such definitions in a machine-readable form, manage them through a web-based user interface and export them both in machine and human readable formats. An API for other applications to perform selected operations is also included.

Data Dictionary follows the common standards, technologies and recommendations in data semantics field as much as concrete business needs make it possible. The main standard of those is ISO 11179.

## Scope

The DD implementation covers a database model supporting about ~70% of the data semantics functionality suggested by ISO11179 ([http://metadata-standards.org/11179](http://metadata-standards.org/11179/)):

* Part 1 (Framework for the specification of data elements)
* Part 2 (Classification for Data Elements)
* Part 3 (Basic Attributes of Data Elements)
* Part 5 (Naming and Identification Principles for Data Elements)

Part 4 about "Rules and Guidelines for the Formulation of Data Definitions" and part 6 about "Registration of Data Elements" are not directly followed, as found cumbersome for DD needs.

Note that the above listed implemented parts have not been implemented to their fullest. It would be more appropriate to say that they have been used as the main source of inspiration for coming up with DD functionality and data model. But some parts have been copied almost one-to-one, like for example the basic set of attributes of data definitions (part 3).

On top of the ISO functionality-compliant data model, a user interface is implemented for

* managing definitions of datasets,
* managing definitions of tables,
* managing definitions of data elements,
* managing definitions of metadata (i.e. attributes in DD terminology)

Transparently for the user, support for representing any of these definitions in XML Schema format has been implemented. However, only a fraction of XML Schema specification is supported, because building a web-interface for managing XML Schema to its full specs was not the object of DD. Support is implemented for:

* Declaration of elements
* Element annotations
* Value domain annotations
* Simple types
* Restrictions based on simple types
* Complex types

This XML Schema fraction is supported both in the UI and export function. Note that aggregate data elements are not supported in DD. XML Schema complex types, sequences and choices have been used to represent the hierarchical structure of datasets > tables > data elements.

To enable comfortable data definition process, a trivial Import tool based on MS Access is implemented for managing definitions in a flat table format. The tool generates XML of what has been entered into its tables and the XML is then finally imported into DD database.

DD is supplemented with documents helping to support its usage, installation and understanding:

* User Guide document for data reporters and data definers.
* Installation guide for those about to install the software.
* The current document, to help understand how DD is built.

# Business requirements

This section covers an overview of the functional requirements that the system should support, and the roles that will perform those tasks.

## Business process support

Data Dictionary provides support for two of the main processes related to the reporting on the different thematic obligations. One is an implementation of the specifications for reporting from the legislation through definition of datasets, reference data and XML schema generation. The second is supporting the actual reporting by providing the dataset definitions, reference data, XML schemas, and various templates for supplying the requested information.

## Supported user roles

The main roles supported by DD are:

* Data suppliers: These are the users who, according to reporting obligations, are responsible for reporting data over Reportnet.
* Data definers: These are the users who have the responsibility, permissions and skills to define the data for data suppliers to deliver.
* Administrators: These are the users who support the other roles by assigning permissions and maintenance of certain shared content.

## Functional requirements

The functional requirements are here grouped by the needs of the different roles.

### All users

All users should be able to search and browse three kinds of data – datasets, tables and data elements, illustrated by the following use-cases:

* Search datasets, tables, elements, schemas, schema sets, vocabularies, vocabulary concepts.
* Browse the list of datasets, tables, elements, schemas, schema sets, vocabularies, vocabulary concepts, and namespaces.
* View details for datasets, tables, elements, schemas, schema sets, vocabularies, and vocabulary concepts.
* View allowed values.
* Export XML schemas for table definitions.
* Export templates for tables and datasets in MS Excel, MS Access and Open document format.
* Export vocabularies in various formats.
* Export technical specifications for datasets.

### Data definers and administrators

Data definers and DD administrators both needs to have access to a more extensive set of operations beyond the use-cases available for all users:

* Define a new dataset, table or element
* Import a dataset definition
* Edit the definitions of a dataset, table, or element
* Edit the data model of a dataset
* Edit complex attributes for a dataset, or table definition
* Edit suggested and allowed values

### Administrators

The operations accessible only for DD administrators should be:

* Browse attribute definitions
* View details for attribute definitions
* Define new attributes
* Edit the definition of attributes, and complex attribute fields
* Manage online help
* Contents clean-up

## Information model

The main information elements that DD need to handle, in order to realise the above requirements are:



# Application architecture

This section covers how the software is designed to support the business requirements, and also the architectural requirements on the software.

## Overview

The overall system architecture of the Data Dictionary consists of three main layers. The presentation layer, providing both access through a human user- and a machine service interface. These interfaces are realised in components residing in the business layer, which in turn use the data layer for access to the underlying database- and external services providing data to the application. Throughout the application layers, security and logging components are used.

## General architecture information

To decouple the different application modules and make them easier to test, the DD application has each object specify its dependencies, which then are injected by the help of the [Spring framework](https://spring.io/), following the pattern of [dependency injection (DI)](https://en.wikipedia.org/wiki/Dependency_injection) upon instantiation.

## Presentation layer

The DD application presentation layer is implemented with the help of the [Stripes framework](http://www.stripesframework.org/), using the [Model – View – Controller (MVC) pattern](https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller), to separate the concerns of presentation logic, business logic and data access. When a HTTP request comes in, it is handled by a controller (Action Bean) that return a view template (JSP page) populated with data. Although it is not strictly enforced throughout the whole application, as in some cases the view template directly interacts with the data access layer.

## Business layer

The business layer, where the parts directly concerning the business domain represents the model in MVC, is implemented in regular Java business objects.

## Service layer

DD provides a service interface in the form of an XML-based web service endpoint. The name of the service is DataDictService and it is provided by the eionet.meta.outservices.OutService class. For details about the API, see the DD XML-RPC service descriptor file (src/main/webapp/DataDictService.xml). To call the service interface, an http post request should be sent to the URL http://<data dictionary URL>/rpcrouter, and in the body of the request both the method to call, and the necessary parameters should be specified as in this example:

<methodCall>

<methodName>**DataDictService.getDatasetWithReleaseInfo**</methodName>

<params>

<param><value><string>**tbl**</string></value></param>

<param><value><string>**8374**</string></value></param>

</params>

</methodCall>

Example in Java (include uit-client.jar, xmlrpc.jar into the class path):

**import** eionet**.**rpcclient**.**ServiceClients**;**

**import** eionet**.**rpcclient**.**ServiceClientIF**;**

**import** eionet**.**rpcclient**.**ServiceClientException**;**

**import** java**.**util**.**Vector**;**

public class TestClass **{**

public static void main **(**String args**[])** **{**

**try** **{**

String srvName **=** "DataDictService"**;**

String methName **=** "getObligationsWithDatasets"**;**

String srvUrl**=**"http://dd.eionet.europa.eu/rpcrouter"**;**

Vector params **=** **new** Vector**();**

ServiceClientIF srv **=** ServicesClients**.**getServiceClient**(**srvName**,** srvUrl**,** ServiceClientIF**.**CLIENT\_TYPE\_XMLRPC**);**

Object value **=** srv**.**getValue**(**methName**,** params**);**

System**.**out**.**println**(**methName **+** "value=" **+** value**);**

**}**

**catch** **(**Exception e**)** **{**

System**.**out**.**println**(“**Error “ **+** e**.**toString**());**

**}**

**}**

**}**

### Methods provided

#### getDatasetWithReleaseInfo

This method returns dataset release info for the given object id and object type in the form of a hash table with the dataset release info.

**Parameters**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| objType | String | Type of object - “tbl” for tables or “dst” for datasets. |
| objId | String | Identifier of the table or dataset. |

**Returns**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| version | String | Version of the table, in the user interface called CheckInNo. |
| objId | String | ? |
| status | String | ? |
| isLatestReleased | String | ? |
| date | String | A Unix timestamp of the moment the definition was created. |
| shortname | String | Name commonly used when referring to a table or dataset. |
| identifier | String | Identifier for the table or dataset. |
| tableIds | Vector of string | IDs of tables belonging to the same dataset. |
| Id | String | ? |

#### getDSTables

This method returns all tables of all released datasets, including historic versions.

**Parameters**

**None.**

**Returns**

The return type is a Vector of Hash tables, where each Hash table represents one table with the following keys:

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| tblId | String | The table's numeric identifier. |
| identifier | String | The table's alphanumeric (i.e. logical) identifier in DD database. |
| shortName | String | The table's short name. |
| dataSet | String | The short name of the dataset where this table belongs to. |
| dateReleased | String | The release date of the dataset where this table belongs to. |

The caller should know that each of the above keys may be missing.

#### getObligationsWithDatasets

Returns a list of structures where every structure represents one obligation whose metadata is given by two keys, see below.

**Parameters**

**None.**

**Returns**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| http://purl.org/dc/elements/1.1/identifier | String | The value of the obligation ID is given as an URI. |
| <http://purl.org/dc/elements/1.1/title> | String | The value of the obligation title is given as string literal. |

#### getParametersByActivityID

Data elements in DD have a static attribute indicating if the element is to be considered a ROD parameter. For identifying parameters by a given reporting activity, the DD first associates the given activity ID with a concrete dataset and then returns all elements that are defined as ROD parameters and defined within the concrete dataset. And this is what this very method does. As an input, it takes a STRING containing the ID of the reporting activity. As an output, it returns an ARRAY of STRUCTs where each STRUCT represents one ROD parameter matching to the given reporting activity. Every such STRUCT may have the keys listed below.

**Parameters**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| raID | String | Reporting obligation id (reporting activity id). |

**Returns**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| elm-name | String | Contains the parameter’s short name. |
| tbl-name | String | Contains the short name of the table where the parameter belongs to. |
| dst-name | String | Contains the short name of the dataset where the parameter belongs to. |

The returned ARRAY is sorted first by dst-name, then by tbl-name and then by elm-name. An example STRUCT in the array is as follows:

**{**

**'elm-name' : 'Country code',**

**'tbl-name' : 'Stations',**

**'dst-name' : 'Water'**

**}**

## Data layer

The objective of the data layer is to provide the other layers with access to the underlying data source, e.g. a database. From the view templates (JSP pages) a data access object is often instantiated and provides methods for retrieving and manipulating the data. Another pattern used is that the controller instantiates the data access objects and provides the results from those into the view template. The data access objects come in to flavours, either they are specialised objects that implement specific data access methods for a certain domain object, or they are just general data access objects.

Concrete access to the database is performed either by using objects that implement a common data access interface for the specific database server technology used, or by using objects that directly issue the SQL statements to the database server (i.e. no database server specific code).

### Database model

The database model for DD is and will therefore be described in two parts. The first one presents the tables involved in dealing with the most important content, like definitions of datasets, tables, elements and the value domains of elements. The second one presents tables that are sort of helper tables or tables containing administrative information.

#### Main tables



#### Attribute

The Attribute table stores the values for the simple attributes defined in the table m\_attribute.

|  |  |
| --- | --- |
| Column | Description |
| DATAELEM\_ID | The field containing the ID of the object for which the attribute value goes. A simple attribute could be relevant for a dataset, table or element. So DATAELEM\_ID identifies the dataset, table or element the attribute goes for. The field’s name could be actually something like PARENT\_ID or OWNER\_ID, but for historical reasons the name is DATAELEM\_ID. |
| M\_ATTRIBUTE\_ID | Indicates the id of the simple attribute and is related to the same-named field in M\_ATTRIBUTE table. |
| PARENT\_TYPE | Indicates if the ID in DATAELEM\_ID is ID of a dataset, table or element. Possible values (respectively) are ‘DS’, ‘T’, and ‘E’. |
| VALUE | Holds the value of M\_ATTRIBUTE\_ID attribute for the given DATAELEM\_ID and PARENT\_TYPE. It is of text type, meaning it could be as long as the database engine has been configured to allow text fields to be. |

#### Complex\_attr\_field

The complex\_attr\_row table is used to identify for which attribute and for which object the complex attribute value row goes. The values of the row’s fields are stored into complex\_attr\_field table.

|  |  |
| --- | --- |
| Column | Description |
| M\_COMPLEX\_ATTR\_FIELD\_ID | Identifies the complex attribute field we’re talking about. It is in foreign key relation with M\_COMPLEX\_ATTR\_FIELD\_ID in M\_COMPLEX\_ATTR\_FIELD table. |
| ROW\_ID | The ID of the complex attribute row. It is in foreign key relation with ROW\_ID in COMPLEX\_ATTR\_ROW table. |

#### Complex\_attr\_row

A complex attribute value is a row of values of the attribute’s fields. Rows are managed in the complex\_attr\_row table and the values of attribute fields are stored in complex\_attr\_field table.

|  |  |
| --- | --- |
| Column | Description |
| HARV\_ATTR\_ID | If the given complex attribute value row is linked to a concrete harvested value row then it is identified by the HARV\_ATTR\_ID field, which is in foreign key relation with the same-named field in HARV\_ATTR table (see more on complex attribute harvesting in the User Guide document). |
| M\_ COMPLEX\_ATTR\_ID | Identifier for the complex attribute. It is in foreign key relation with the same-named field in M\_ COMPLEX\_ATTR table. |
| PARENT\_ID | Identifies the object for which the row goes. It could be a dataset, table or element. PARENT\_TYPE tells you which one it is. A value of ‘DS’ indicates a dataset, a value of ‘T’ indicates a table and a value of ‘E’ indicates an element. |
| POSITION | Not used. |
| ROW\_ID | The primary key and unique identifier of rows. DD uses it to refer to concrete rows. The PARENT\_ID + PARENT\_TYPE + M\_COMPLEX\_ATTR\_ID must be unique. |

#### Dataelem

This table holds the definitions of elements, both common and non-common. Every row in this table represents a concrete version of a logically unique element.

| Column | Description |
| --- | --- |
| DATAELEM\_ID | The unique identifier and primary key of a row. |
| DATE | A Unix timestamp of the moment the definition was created. |
| GIS | Indicates the element’s GIS type (see the chapter of DD concepts). If the value is NULL, it means the element is not a GIS element at all. |
| IDENTIFIER | Together with PARENT\_NS, the unique identifier of an element. |
| IS\_ROD\_PARAM | Indicates if the element is to be considered a parameter by ROD (Reporting Obligations Database) when the latter asks for the list of parameters by given reporting obligation ID (for more see the chapter about XML-RPC bridge below). Possible values are ‘Y’ and ‘N’, the latter being the default. |
| PARENT\_NS | Holds the ID of the namespace of the table where the given element belongs to. If this is a common element, the value of will be NULL. |
| REG\_STATUS | Indicates the registration status of the definition of a common element. For non-common elements this field is irrelevant. For possible values see the chapter about registration statuses in the User Guide. |
| SHORT\_NAME | Holds the attribute that is called Short name in the UI. A short name is a character string used to operationally identify elements (i.e. how users and data definers commonly refer to a table). A short name is not used as a unique identifier in DD. For example, several non-common elements in a table can have the same short name. Similarly, several common elements can have the same short name. |
| TOP\_NS | The ID of the corresponding namespace of the dataset where the non-common element belongs to. For common elements the value of TOP\_NS is NULL. Namespaces are kept in the NAMESPACE table. |
| USER | The name of the user who created the definition. |
| VERSION | Together with IDENTIFIER & PARENT\_NS, the unique identifier of an element definition’s concrete version. In the user interface version is called CheckInNo. There is no unique SQL restriction on IDENTIFIER + PARENT\_NS + VERSION due the way business logic was implemented. |
| WORKING\_COPY | Indicates if the definition corresponding to this row is a working copy. Possible values are ‘Y’ and ‘N’, the latter being the default. |
| WORKING\_USER | Holds the name of the user who’s working with the element’s definition. |

#### Dataset

This table holds the definitions of datasets. Every row in this table represents a concrete version of a logically unique dataset definition.

| Column | Description |
| --- | --- |
| CHECKEDOUT\_COPY\_ID | The ID of the copy/version from which this copy/version was created. |
| CORRESP\_NS | The ID of the dataset’s corresponding namespace. Namespaces are kept in the NAMESPACE table. |
| DATASET\_ID | The unique identifier and primary key of a row. |
| DATE | A Unix timestamp of the moment the definition was created. |
| DELETED | Contains the name of the user who has non-permanently deleted the concrete dataset definition. If the value of this field is NULL then the dataset is not deleted. If the value contains a concrete username, it means the user has non-permanently deleted the dataset definition. The latter means that the definition does not appear in DD UI, except for the page of non-permanently deleted dataset definitions, from where it can be restored back into “live”. If a dataset definition is deleted permanently, the corresponding row is deleted from DATASET table. |
| DETAILED\_VISUAL | File name of uploaded detailed image to show the dataset structure as a figure. |
| DISP\_CREATE\_LINKS | Holds a number indicating the so-called “Create ...” links that should be displayed for this dataset definition in user interface (see the User Guide for more on this). Every possible “Create ...” link has a weight. If the division of DISP\_CREATE\_LINKS by a link’s weight is not a multiplicand of 2, then that link should not be displayed for this dataset definition in user interface. |
| IDENTIFIER | The unique identifier of a dataset. |
| REG\_STATUS | Holds the dataset definition’s registration status. For possible values see the chapter about registration statuses in the User Guide. |
| SHORT\_NAME | A short name is a character string used to operationally identify datasets (ie how users and data definers commonly refer to a dataset). A short name is not used as a unique identifier in DD. For example, several datasets can have the same short name. |
| USER | The name of the user who created the definition. |
| VERSION | The unique identifier of a dataset definition’s concrete version is assembled by IDENTIFIER and VERSION. Remember that in user interface (UI) the version is called CheckInNo. Due to restrictions from the way the business logic is implemented, there is no unique SQL restriction on IDENTIFIER + VERSION. |
| VISUAL | File name of uploaded image to show the dataset structure as a figure. |
| WORKING\_COPY | Indicates if the definition corresponding to this row is a working copy. Possible values are ‘Y’ and ‘N’, the latter being the default. |
| WORKING\_USER | Holds the name of the user who’s working with the dataset’s definition. |

#### Ds\_table

This table holds the definitions of tables. Every row in this table represents a concrete version of a logically unique DD table.

|  |  |
| --- | --- |
| Column | Description |
| CORRESP\_NS | The ID of the table’s corresponding namespace. Namespaces are kept in the NAMESPACE table. |
| DATE | A Unix timestamp of the moment the definition was created. |
| IDENTIFIER | Identifier for the table. The unique identifier of a table is IDENTIFIER + PARENT\_NS. |
| NAME | The table’s name. |
| PARENT\_NS | Holds the ID of the corresponding namespace of the dataset where the given table belongs to. |
| SHORT\_NAME | A short name is a character string used to operationally identify tables (i.e. how users and data definers commonly refer to a table). A short name is not used as a unique identifier in DD. For example, several tables in a dataset can have the same short name |
| TABLE\_ID | The unique identifier and primary key of a row. |
| USER | The name of the user who created the definition. |
| VERSION | Version of the table, in the user interface called CheckInNo. Due to restrictions from the way the business logic was implemented, there is no unique SQL restriction on IDENTIFIER + PARENT\_NS + VERSION. The unique identifier of a table definition’s concrete version is assembled by IDENTIFIER + PARENT\_NS + VERSION. |
| WORKING\_COPY | Indicates if the definition corresponding to this row is a working copy. Possible values are ‘Y’ and ‘N’, the latter being the default. |
| WORKING\_USER | Holds the name of the user who’s working with the table’s definition. |

#### Dst2tbl

Versioning of dataset definitions requires versioning of table definitions as well, only the latter happens transparently for the users. A concrete version of a table’s definition can participate in many versions of the definition of its parent dataset. And since a dataset definition, in turn, may contain definitions of several tables, there is a many-to-many relationship between DATASET and DS\_TABLE tables. This table implement that relationship.

|  |  |
| --- | --- |
| Column | Description |
| POSITION | Indicates the table’s position in the dataset. It is relevant only when displaying the list of tables in a dataset. |
| DATASET\_ID | The ID of the dataset. |
| TABLE\_ID | The ID of the table. |

#### Fk\_relation

Since tables in a reported dataset might have foreign key relations to each other, DD enables to define them as well. A foreign key relation between two tables is handled simply by defining a foreign key relation between some of their elements. On both sides several elements can participate in the relation. These foreign key relations are managed in this table.

|  |  |
| --- | --- |
| Column | Description |
| A\_CARDIN | Indicates the cardinality’s side represented by the element in A\_ID. |
| A\_ID | Contains the DATAELEM\_ID of the element for which the relation was specified. |
| B\_CARDIN | Indicates the cardinality’s side represented by the element in B\_ID. Possible values of both sides can be '0' (zero), '1' (one), '+' (one or many), '\*' (zero or many). |
| B\_ID | Contains the DATAELEM\_ID of the element that was selected to be a foreign key for the element for which the relation was specified. |
| DEFINITION | Contains comments on the given relation, if any specified. |
| REL\_ID | Unique identifier for the foreign key relation. |

#### FXV

A data element can be defined to have values only from a pre-fixed value set. Also, a dynamic attribute can be defined to have pre-fixed values only. These value sets are kept and managed in the FXV table.

|  |  |
| --- | --- |
| Column | Description |
| DEFINITION | The meaning of the given value. |
| FXV\_ID | Unique identifier for the rows in this table. |
| IS\_DEFAULT | Indicates if the given fixed value is the default. Possible values are ‘Y’ and ‘N’, the latter being the default. |
| OWNER\_ID | The ID of the object for which the value goes. |
| OWNER\_TYPE | The type of the object for which the value goes, possible values are ‘elem’ (data element) and ‘attr’ (dynamic attribute). |
| SHORT\_DESC | Similar to DEFINITION, but in a shorter way that can be conveniently displayed in value lists in UI. |
| VALUE | Contains the value itself. The field is of TEXT type, so it can be as long as the configuration of the data engine allows it to be. |

#### Harv\_attr

Complex attributes can be associated with a harvester that harvests the attribute’s values from an outer source over HTTP. Such a harvester acts automatically in the background and over certain interval, thus making sure the harvested values are always up to date. This is a useful feature for complex attributes that represent some administrative information that might be a subject to frequent change. For example if an organisation’s metadata is used a complex attribute’s value in many dataset definitions then the users don’t need to go through all those definitions and change the organisation’s metadata every time it changes. If the attribute has been associated with a value-harvester then the latter does such tedious work for the user. Harvested complex attribute values are managed in the HARV\_ATTR table. Every row in this table represents a harvested complex attribute row.

|  |  |
| --- | --- |
| Column | Description |
| HARV\_ATTR\_ID | The harvested value row’s ID as given by the outer source that is harvested. |
| HARVESTED | A Unix timestamp indicating the time of the harvest session when the given row was harvested. This is needed because one and the same row might have been harvested at different sessions and the business logic makes sure that always the latest is kept and used. |
| HARVESTER\_ID | A string by which the user identifies the harvester. It is recommended that the string is chosen so that you understand what the harvester is about. The field is used as a foreign key to the same field in M\_COMPLEX\_ATTR table. |
| LOGICAL\_ID | A combination of HARVESTER\_ID + HARV\_ATTR\_ID and is used as foreign key to the HARV\_ATTR\_ID field in COMPLEX\_ATTR\_ROW table. |
| MD5KEY | Every row is uniquely identified this MD5KEY field, which is a combination of HARVESTER\_ID + HARV\_ATTR\_ID + HARVESTED, and is used as foreign key to the HARV\_ATTR\_MD5 field in HARV\_ATTR\_FIELD table. The latter holds the values of the fields of the given harvested complex attribute row. |

#### Harv\_attr\_field

This is the table where the values of the fields of the harvested complex attribute rows are kept.

|  |  |
| --- | --- |
| Column | Description |
| HARV\_ATTR\_MD5 | Identifier for the harvested row, which is a foreign key to MD5KEY in HARV\_ATTR table. |
| FLD\_NAME | Identifier for the harvested row field, which is a foreign key to the NAME in M\_COMPLEX\_ATTR\_FIELD table. |
| FLD\_VALUE | The harvested field’s value. |

#### Inference\_rule

This table defines the semantic relationships between data elements, such as skos:broader/skos:narrower relationships.

|  |  |
| --- | --- |
| Column | Description |
| DATA\_ELEM | Foreign key to the DATAELEM\_ID in the Dataelem table. |
| TARGET\_ELEM | Foreign key to the DATAELEM\_ID in the Dataelem table for the target element. |
| RULE | The name of the rule, e.g. “owl:inverseOf”. |

#### M\_attribute

The set of possible data definition attributes is flexible, where users can define new definition attributes at any time and they can remove existing ones as well. At the same time, some attributes are static, hard-coded into the tables of the database model. These are attributes like IDENTIFIER, SHORT\_NAME, IS\_ROD\_PARAM, etc. The ones that users can dynamically define and remove, are called static dynamic attributes and their definitions are kept in the tables M\_ATTRIBUTE (definitions of simple attributes) and M\_COMPLEX\_ATTR (definitions of complex attributes).

|  |  |
| --- | --- |
| Column | Description |
| DATA\_TYPE | ? |
| DEFINITION | Contains the definition of the attribute. This is free text explaining the essence and the meaning of the attribute. In UI, the value of this field is called Definition. |
| DISP\_MULTIPLE | Indicates if this simple attribute can have multiple values at a time. For example an attribute like Keyword could have several values at the same time. Possible values are ‘0’ (no multiple values allowed and this is the default) and ‘1’ (multiple values allowed). |
| DISP\_MULTIPLE | Determines if the attribute can have multiple values |
| DISP\_ORDER | Reflects the position at which the attribute should be displayed in the overall attributes list and the data definition views. In UI, the value of this field is called Display order. |
| DISP\_TYPE | Reflects how the given attribute should be displayed in web forms. Possible values are ‘text’ (corresponding to <input type=”text”> in HTML), ‘textarea’ (corresponding to <textarea> in HTML), ‘select’ (corresponding to <select> in HTML) and ‘image’. The latter is a special simple attribute type, because it means the attribute is displayed as an image and the attribute’s value is the relative path to the image in DD visuals directory (more in chapters below). In UI, the value of this field is called Display type. |
| DISP\_WHEN | Indicates for which data definitions the attribute is relevant. There is 4 options here: an attribute could be relevant for datasets, tables, elements with fixed values or elements with quantitative values. All 4 options have a weight and when the division of DISP\_WHEN by that weight is a multiplicand of 2 then the attribute is relevant for that particular weight. In UI, the value of this field is called Display for. The field is displayed as a set of 4 checkboxes, each corresponding to one of the above-described options. |
| DISP\_WIDTH | Is relevant only if the value of DISP\_TYPE is ‘textarea’. In such a case it indicates the height of the text area in terms of rows. In UI, the value of this field is called Display width. |
| DISP\_WIDTH | Is relevant only if the value of DISP\_TYPE is ‘text’. In such a case it indicates the width of the text box in terms of characters. In UI, the value of this field is called Display width. |
| INHERIT | Indicates the given attribute’s inheritance type. For details about attribute inheritance see the relevant chapter(s) in User Guide document. Possible values are ‘0’ (no inheritance, the default), ‘1’ (inherit attribute values from parent level with possibility to add new values) and ‘2’ (inherit attribute values from parent level with possibility to overwrite them). |
| LANGUAGE\_USED | Values in Linked Data can have a language attribute. This Boolean determines if the value can have a language attribute. |
| M\_ATTRIBUTE\_ID | The unique identifier and primary key of rows in this table. It is also used as the unique identifier of simple attributes defined. |
| NAME | Not as commonly used as SHORT\_NAME. It contains the full name of the defined attribute. In UI, the value of this field is called Name. |
| NAMESPACE\_ID | Indicates the namespace where this attribute belongs to and is related to the NAMESPACE\_ID field in NAMESPACE table (see for more below). DD comes with two pre-defined namespaces for dynamical attributes: ISO11179 attributes and Data Dictionary attributes. Attributes originating from ISO should be defined in the ISO namespace, while as attributes originating from DD should be defined in the Data Dictionary namespace. In UI, the value of this field is called Context. |
| OBLIGATION | Indicates if this attribute is mandatory, optional or conditional for the data definitions to specify. Possible values are (respectively) ‘M’, ‘O’, ‘C’. In UI, the value of this field is called Obligation. |
| RDF\_PROPERTY\_NAME | RDF property name for this attribute |
| RDF\_PROPERTY\_NAMESPACE\_ID | Foreign key to the column ID in the t\_rdf\_namespace-table, linking a namespace declaration for this RDF\_PROPERTY\_NAME. |
| SHORT\_NAME | Used by users for commonly referring to attributes. Even though there is no SQL unique restriction on this field, each attribute must have a unique SHORT\_NAME. In UI, the value of this field is called Short name. The values of SHORT\_NAME must have no white space. |

#### M\_complex\_attr

This table contains the definitions of complex attributes.

|  |  |
| --- | --- |
| Column | Description |
| DEFINITION | Contains the definition of the attribute. This is free text explaining the essence and the meaning of the attribute. In UI, the value of this field is called Definition. |
| DISP\_ORDER | *Not used?* |
| DISP\_WHEN | *Not used* |
| HARVESTER\_ID | The ID of the value-harvester associated with this complex attribute. More on this is available in the chapter about harvested complex attributes in User Guide document. This field is related to the HARVESTER\_ID field in HARV\_ATTR table (see for more below) and in UI this field is called Linked harvester. |
| INHERIT | Serves exactly the same purpose as it did in M\_ATTRIBUTE table, see the relevant explanation above. |
| M\_COMPLEX\_ATTR\_ID | The unique identifier and primary key of rows in this table. It is also used as the unique identifier of complex attributes defined. |
| NAME | Not as commonly used as SHORT\_NAME. It contains the full name of the defined attribute. In UI, the value of this field is called Name. |
| NAMESPACE\_ID | Indicates the namespace where this attribute belongs to and is related to the NAMESPACE\_ID field in NAMESPACE table (see for more below). DD comes with two pre-defined namespaces for dynamical attributes: ISO11179 attributes and Data Dictionary attributes. Attributes originating from ISO should be defined in the ISO namespace, while as attributes originating from DD should be defined in the Data Dictionary namespace. In UI, the value of this field is called Context. |
| OBLIGATION | *Not used* |
| SHORT\_NAME | Used by users for commonly referring to attributes. Even though there is no SQL unique restriction on this field, each complex attribute must have a unique SHORT\_NAME. In UI, the value of this field is called Short name. The values of SHORT\_NAME must have no white space. |

Almost all complex attributes are considered optional, except for the “SubmitOrganisation”-attribute which is mandatory for all data definitions. Definitions of tables and non-common elements inherit the values of the “SubmitOrganisation”-attribute of the parent dataset!

#### M\_complex\_attr\_field

Complex attributes consist of fields and a complex attribute value is row of values of those fields. The set of fields of a complex attribute is flexible just like the attributes themselves. New fields can be defined and removed at any time via UI. The definitions of complex attribute fields are kept in this table.

|  |  |
| --- | --- |
| Column | Description |
| M\_COMPLEX\_ATTR\_ID | Holds the ID of the complex attribute for which the field is relevant. It is in foreign key relation with the same-named field in M\_COMPLEX\_ATTR table. |
| NAME | The name of the field. This is what the users commonly use when referring to complex attribute fields. Field names within a complex attribute must be unique! |
| M\_COMPLEX\_ATTR\_FIELD\_ID | The unique identifier and primary key of the rows in this table. It is also used as the unique identifier of complex attribute fields. |
| DEFINITION | This is free text explaining the field’s meaning. |
| POSITION | Indicates the field’s position in its owner complex attribute. This is relevant for display purposes. |
| PRIORITY | Specifies weather the field should be displayed in the definitions of datasets/tables/elements in order to save space on the page. A field with priority ‘0’ (displayed as ‘High’ in user views and used as the default) is displayed, while the field with priority ‘1’ (displayed as ‘Low’ in user views) is not. In the detailed views of complex attributes of data definitions, however, all fields are always displayed. |
| HARV\_ATTR\_FLD\_NAME | Maps the given field to a field of a complex attribute value-harvester. This is relevant only if the owner attribute of this field is defined to be harvestable by some complex attribute value-harvester. For more about complex attribute harvesting see the relevant chapter in User Guide document. |

#### Namespace

This is the table where the corresponding namespaces of datasets and tables are managed. In addition, it also contains 3 pre-defined namespaces coming with the DD software package. These are the namespace where all the datasets belong (called datasets), the namespace where all the ISO-originating attributes must belong (called ISO11179 attributes) and the namespace where all the DD-originating attributes must belong (called Data Dictionary attributes).

|  |  |
| --- | --- |
| Column | Description |
| DEFINITION | Designates the meaning of the given namespace. |
| FULL\_NAME | The namespace’s full name. |
| NAMESPACE\_ID | The unique identifier and primary key of rows in this table. It is also used as the unique identifier of namespaces. |
| PARENT\_NS | If the given namespace corresponds to a dataset or a table, then PARENT\_NS contains the ID of the namespace where the given namespace belongs to. Corresponding namespaces of datasets belong into the pre-defined namespace datasets. Corresponding namespaces of tables belong into the corresponding namespaces of their parent datasets. |
| SHORT\_NAME | The short name of the given namespace. |
| WORKING\_USER | If the given namespace corresponds to a dataset, then WORKING\_USER contains the name of the user who’s working with that dataset. If the dataset is not in work, the value of WORKING\_USER is null. |

#### T\_rdf\_namespace

Attributes can be described also with an RDF property, allowing tables to be output as RDF. This table supports this by storing the RDF schema URIs and a namespace mapping.

|  |  |
| --- | --- |
| Column | Description |
| ID | Unique identifier of the row. |
| NAME\_PREFIX | The namespace prefix, e.g. “skos”. |
| URI | The namespace URI, e.g. "http://www.w3.org/2004/02/skos/core#". |

#### T\_schema

This table store XML Schemas developed for different reporting obligations.

|  |  |
| --- | --- |
| Column | Description |
| CHECKEDOUT\_COPY\_ID | Relevant only if WORKING\_COPY='Y', and it's the ID of the copy/version from which this copy/version was created. |
| COMMENT | If WORKING\_COPY='Y', then it's the comment that user supplied when checking out. If WORKING\_COPY='N', it's the comment that user supplied when he/she checked in. |
| CONTINUITY\_ID | Since we shall allow versioning of a schema inside a schema set (or a root-level schema), different versions of conceptually the same schema will need to be identified. So this is column helps to keep the history line just like in SCHEMA\_SET. |
| DATE\_MODIFIED | A Unix timestamp of the moment the definition was created. If WORKING\_COPY='Y', it's the date-time of creating the working copy, otherwise it's the date-time of check-in. |
| FILENAME | The schema's file name which is also its unique identifier within the set it belongs to, i.e. this is the last segment in the schema’s URL. So it contains also the “.xsd”-extension. |
| OTHER\_DOCUMENT | ? |
| REG\_STATUS | Holds the schema sets definition’s registration status. For possible values see the chapter about registration statuses in the User Guide. |
| SCHEMA\_ID | Auto-generated ID for convenience and for referring to ATTRIBUTE table. |
| SCHEMA\_SET\_ID | Foreign key to SCHEMA\_SET.SCHEMA\_SET\_ID. Shall be NULL if this is a root-level schema. |
| USER\_MODIFIED | The name of the user who created the definition. If WORKING\_COPY='N', it's the name of the user who checked it in, otherwise irrelevant. |
| WORKING\_COPY | Indicates if the definition corresponding to this row is a working copy. Possible values are ‘Y’ and ‘N’, the latter being the default. |
| WORKING\_USER | Holds the name of the user who’s working with the definition. |

#### T\_schema\_set

Schemas that logically belong together, can be collected into a schema set. For example a schema set can represent a reporting obligation, and bring together all schemas relevant to that particular obligation. Just like datasets are collections of tables, so are schema sets collections of schemas.

|  |  |
| --- | --- |
| Column | Description |
| CHECKEDOUT\_COPY\_ID | Relevant only if WORKING\_COPY='Y', and it's the ID of the copy/version from which this copy/version was created. |
| COMMENT | If WORKING\_COPY='Y', then it's the comment that user supplied when checking out. If WORKING\_COPY='N', it's the comment that user supplied when he/she checked in. |
| CONTINUITY\_ID | Since different versions of logically the same schema set can have different identifiers, this ID keeps the history line. |
| DATE\_MODIFIED | A Unix timestamp of the moment the definition was created. If WORKING\_COPY='Y', it's the date-time of creating the working copy, otherwise it's the date-time of check-in. |
| IDENTIFIER | The schema set's identifier, i.e. this will become the last segment in the schema set's URL. |
| PARENT | The full path of the parent schema set. This is the path starting after “http://dd.eionet.europa.eu/schemas”. It always starts with "/", e.g. "/eprtr/common". |
| REG\_STATUS | Holds the schema sets definition’s registration status. For possible values see the chapter about registration statuses in the User Guide. |
| SCHEMA\_SET\_ID | Auto-generated ID for convenience and for referring to ATTRIBUTE table. |
| USER\_MODIFIED | The name of the user who created the definition. If WORKING\_COPY='N', it's the name of the user who checked it in, otherwise irrelevant. |
| WORKING\_COPY | Indicates if the definition corresponding to this row is a working copy. Possible values are ‘Y’ and ‘N’, the latter being the default. |
| WORKING\_USER | Holds the name of the user who’s working with the definition. |

#### T\_site\_code

The CDDA dataflow has a site code service, where appointed Reportnet users can reserve a set of site codes for their new sites after logging into the service. This table stores the new and reserved site codes.

|  |  |
| --- | --- |
| Column | Description |
| CC\_ISO2 | ISO 2 alpha code of country the site is allocated to. |
| DATE\_ALLOCATED | Date when the site code was allocated to the country. |
| DATE\_CREATED | ? |
| DATE\_DELETED | Date the site code was flagged as deleted. |
| INITIAL\_SITE\_NAME | Name of the site from the CDDA dataset. |
| PARENT\_ISO | ISO 3 alpha code of the country from the CDDA dataset. |
| SITE\_CODE | The site code in question. |
| SITE\_CODE\_NAT | National code of the site from the CDDA dataset. |
| STATUS | Status for the site code, options are 'available', 'allocated', 'assigned', 'deleted', 'disappeared'. |
| USER\_ALLOCATED | User ID of the user allocating the site code. |
| USER\_CREATED | ? |
| VOCABULARY\_CONCEPT\_ID | Foreign key to the vocabulary\_concept table, where the site name is located. |
| YEARS\_DELETED | A year, or a semicolon separated list of years when the site record with the given SITE\_CODE has been flagged for deletion by the data reporter. Multiple years indicate errors made by the national data reporters (just for information at this stage). |
| YEARS\_DISAPPEARED | A year when the site record with the given SITE\_CODE was not reported in the CDDA dataset, presumably incorrectly deleted by the data reporter. |

##### Tbl2elem

Versioning of dataset definitions requires versioning of the definitions of its sibling tables and non-common elements as well. Only the latter happens transparently for the users. And so it follows that a concrete version of a non-common element’s definition can participate in many versions of the definition of its parent table. And since a table definition, in turn, may contain definitions of several non-common elements, it appears that there is a need for a many-to-many relationship between DS\_TABLE and DATAELEM tables. The need also follows from the fact, that a common element might be re-used in many tables. Hence an associative table is needed and that’s what the TBL2ELEM is.

|  |  |
| --- | --- |
| Column | Description |
| DATAELEM\_ID | Dataelem table reference. |
| MANDATORY | ? |
| MULTIVAL\_DELIM | ? |
| POSITION | Indicates the element’s position in the table. It is relevant only when displaying the list of elements in a table. |
| PRIM\_KEY | ? |
| TABLE\_ID | DS\_Table table reference. |

Notice that the TBL2ELEM table is used for representing table-to-element relations both for common and non-common elements.

#### Vocabulary

The definition of a vocabulary, e.g. the container for a set of concepts (values).

|  |  |
| --- | --- |
| Column | Description |
| BASE\_URI | Base URI for the vocabulary, each concept will a concept URI made up of this URI + the concept notation value. |
| CHECKEDOUT\_COPY\_ID | Relevant only if WORKING\_COPY='Y', and it's the ID of the copy/version from which this copy/version was created. |
| CONCEPT\_IDENTIFIER\_NUMERIC | If true then the system will propose a new identifier as MAX+1 when creating a new concept |
| CONTINUITY\_ID | Since different versions of logically the same vocabulary can have different identifiers, this ID keeps the history line. |
| DATE\_MODIFIED | A Unix timestamp of the moment the definition was created. If WORKING\_COPY='Y', it's the date-time of creating the working copy, otherwise it's the date-time of check-in. |
| FOLDER\_ID | Foreign key to the vocabulary set, saying this vocabulary belongs to that set. |
| IDENTIFIER | Textual identifier for the vocabulary, is not unique. A unique identifier for a vocabulary is made up FOLDER\_ID + IDENTIFIER + WORKING\_COPY. |
| LABEL | A label describing the vocabulary. |
| NOTATIONS\_EQUAL\_IDENTIFIERS | If the NOTATION value should be equal to the IDENTIFIER value for all concepts in the vocabulary. |
| REG\_STATUS | Status of the vocabulary, options are 'Draft', 'Public draft', 'Released', and the default is 'Draft'. |
| USER\_MODIFIED | The name of the user who created the definition. If WORKING\_COPY='N', it's the name of the user who checked it in, otherwise irrelevant. |
| VOCABULARY\_ID | Numeric primary key |
| VOCABULARY\_TYPE | Textual value for what type of vocabulary this is, default is “Common”. |
| WORKING\_COPY | Indicates if the definition corresponding to this row is a working copy. Possible values are ‘Y’ and ‘N’, the latter being the default. |
| WORKING\_USER | Holds the name of the user who’s working with the definition. |

#### Vocabulary2elem

Manages the many-to-many relationship between the vocabulary table and the dataelem table. It defines which elements are to be shown on the page, where the user can create a new concept.

|  |  |
| --- | --- |
| Column | Description |
| DATAELEM\_ID | Foreign key to the dataelem table |
| VOCABULARY\_ID | Foreign key to the vocabulary table |

Both columns together form the primary key of rows in this table.

#### Vocabulary\_concept

The vocabulary concepts holds the individual concepts in a vocabulary, i.e. all the values. It only holds core attributes – identifier, notation and label.

|  |  |
| --- | --- |
| Column | Description |
| CREATION\_DATE | Timestamp of when the concept was created. |
| DEFINITION | Text where the concept can be defined. |
| IDENTIFIER | Textual identifier for a concept, uniquely identifies a concept together with VOCABULARY\_ID. |
| LABEL | The label describing the concept. |
| NOTATION | Short code as an identifier for the concept |
| OBSOLETE\_DATE | If the concept is made obsolete, this field holds a timestamp of when that happened. |
| ORIGINAL\_CONCEPT\_ID | When a concept is modified, a copy is written into the vocabulary\_concept table with ORIGINAL\_CONCEPT\_ID set. |
| VOCABULARY\_CONCEPT\_ID | Numeric primary key |
| VOCABULARY\_ID | Foreign key to the vocabulary table. |

#### Vocabulary\_concept\_element

Concepts can have additional properties, which are made up of elements defined in the dataelem table, and this table connects those definitions with the concepts and their values.

|  |  |
| --- | --- |
| Column | Description |
| ATTR\_MD5 | Unique hashed key value to avoid duplicate concept element values. The hash is computed on the vocabulary\_concept\_id, dataelem\_id, related\_concept\_id or new.element\_value, and the language. |
| DATAELEM\_ID | Foreign key to the dataelem table for the definition of the concept’s attribute. |
| ELEMENT\_VALUE | Textual/numeric value for the concept. |
| ID | Numeric primary key |
| LANGUAGE | Language code for the value, e.g. when the concept has values in multiple languages. |
| LINK\_TEXT | Not used |
| RELATED\_CONCEPT\_ID | Foreign key to the vocabulary\_concept table, to be used when the concept’s value is another concept, and not a string etc. |
| VOCABULARY\_CONCEPT\_ID | Foreign key to the vocabulary\_concept table. |

#### Vocabulary\_set

A set of vocabularies allows grouping related vocabularies together.

|  |  |
| --- | --- |
| Column | Description |
| ID | Numeric primary key |
| IDENTIFIER | Textual unique identifier for the set. |
| LABEL | Name of the schema set shown in the user interface. |

#### Supporting tables



#### Acl\_rows & acls

These two tables are needed to implement the Access Control Lists (ACL) mechanism from an external Java library in DD, see <https://github.com/eea/eionet.acl> for more information.

#### Cache

For every dataset, table and element definition users can download differently formatted outputs (the so-called “Create ...” links in UI), unless the data definers have prohibited them for certain definitions. Since the generation of those outputs can take considerable amount of time (mainly for PDF and MS Excel documents), data definers have the possibility to cache outputs in at least these two formats. The caching means simply that the generated output is stored into DD host machine’s file system and so later it can be retrieved for quick download. Metadata on such cached outputs (i.e. files) is kept in CACHE table. All these cached files are always stored into the directory whose path is configured by the “doc-path” parameter in DD’s configuration in web.xml (see for more in Installation Guide).

|  |  |
| --- | --- |
| Column | Description |
| ARTICLE | The cached file’s type. The values can be 'pdf' (the default), 'xls' (MS Excel), 'xform' (XForm on top of XML), 'xmlinst' (an empty XML instance file for the data reporting). While all these types can be cached, only PDF and XLS are supported in UI. |
| CREATED | Holds the Unix timestamp of the moment the file was cached. |
| FILENAME | The cached file’s name. |
| OBJ\_ID | The ID of the object for which the file goes. |
| OBJ\_TYPE | Identifies the object’s type, either ‘dst’ (dataset, the default), ‘tbl’ (table), or ‘elm’ (element). |

The primary key of rows is formed by OBJ\_ID + OBJ\_TYPE + ARTICLE.

#### Databasechangelog & databasechangeloglock

These two tables are needed for implementing [Liquibase](http://www.liquibase.org/), which is an external library used for tracking, managing and applying database schema changes.

#### Doc

For every dataset, table and element definition data definers can upload informative documents that can later be downloaded by data reporters for further guidelines. These documents are stored into a directory whose path is configured by the “doc-path” parameter in DD’s configuration in web.xml (see for more in Installation Guide). The metadata on those documents are in this table.

|  |  |
| --- | --- |
| Column | Description |
| ABS\_PATH | The document’s full Unix path. |
| MD5\_PATH | The MD5-hash of the document’s path. |
| OWNER\_ID | The ID of the object for which the document goes. |
| OWNER\_TYPE | Identifies the object’s type, which could be ‘dst’ (dataset, the default), ‘tbl’ (table), or ‘elm’ (element). |
| TITLE | The document’s title (displayed in UI as a hint about the document’s contents). |

Primary key of the rows in DOC table is formed by OWNER\_ID + OWNER\_TYPE + MD5\_PATH. So you notice that a physically single document could be represented several times in this table, because it could be used for several different objects.

#### Documentation

This table is needed to implement the Eionet documentation module, an external library to support documentation pages, see <https://github.com/eea/eionet.doc>.

#### Dst2rod

Activities in ROD\_ACTIVITIES table are associated with concrete datasets by the help of DST2ROD table, because the relation between DATASET and ROD\_ACTIVITIES tables is of many-to-many nature.

|  |  |
| --- | --- |
| Column | Description |
| DATASET\_ID | Contains the ID of the dataset for which the association goes. |
| ACTIVITY\_ID | The associated activity. |

Both columns together form the primary key of rows in this table.

#### Hlp\_screen and hlp\_area

These two tables are needed to implement the Dynamical Online Help (DynOHelp) mechanism from an external Java library in DD, see <https://github.com/eea/eionet.helpadmin> for more information.

#### Rod\_activities

Dataset definitions in DD can be associated with corresponding reporting obligations in the Reporting Obligations Database (ROD). Metadata on reporting obligations can be downloaded on the fly from ROD at the moment the data definer starts to create the association (there’s an XML-RPC bridge for that, see 3.9.2). Metadata on associated obligations (and associated obligations only, meaning not all obligations found from ROD) is kept in the ROD\_ACTIVITIES table. For historical reasons, one can assume that an activity and an obligation is one and the same thing.

|  |  |
| --- | --- |
| Column | Description |
| ACTIVITY\_ID | Contains the ID of the obligation as received from ROD |
| ACTIVITY\_TITLE | Contains the title of the obligation as received from ROD. |
| LEGINSTR\_ID | Contains the instrument’s ID as received from ROD. |
| LEGINSTR\_TITLE | The instrument’s title as received from ROD. Since several reporting activities (i.e. obligations) can have the same title in ROD, we need some kind of way to distinct them when displayed by title in DD UI. Therefore the table also tells which legal instrument the activity belongs to. That’s because there can no two activities with the same title within a single legal instrument. |

## Security

### Authentication

For authenticating users DD uses a single sign-on mechanism, implemented with the Central Authentication Service protocol (CAS) against EIONET’s LDAP directory. Therefore DD does not carry a user database of its own.

A given username and password (given via the eionet.meta.LoginServlet servlet) are authenticated by using the utility methods coming with eionet-dir.jar. The properties for finding and connecting the EIONET LDAP directory must be set in the /datadict/public/WEB-INF/classes/eionetdir.properties-file. This is further explained in the DD Installation guide document.

### Authorisation

For checking an authenticated user’s permissions, DD uses the Eionet Access Control Lists (ACL) mechanism. Full documentation about the concept of this mechanism – how it works and how applications (including DD) can use it is available at <https://github.com/eea/eionet.acl>.

The ACL mechanism is hidden in the uit-\*.jar libraries in /datadict/public/WEB-INF/lib directory and certain properties must be configured in /datadict/public/WEB-INF/classes/uit.properties file. For details about this configuration, see the above-mentioned ACL documentation and the DD Installation guide document.

## Logging and error management

When the application encounters an error, an error is thrown and a message describing the error is logged to the application log using the log4j library. If it is deemed that the application can recover from the error, the error is then caught and the application can continue (e.g. failed user authentication).

## Collaboration with other software



### Eionet Directory

DD uses EEA's Central Authentication Service (CAS), which is linked to the Eionet user directory, for authentication of users. The CAS protocol operates over HTTP exchanging the authentication information. DD retrieves the login cookie and username after a successful login.

### Reporting Obligations Database (ROD)

The integration with ROD consists of that DD will collect data about reporting obligations by calling the ROD XML-RPC interface when a data definer creates an association between a dataset in DD and a reporting obligation in ROD. The service method called on ROD is “getActivities”.

### Dynamical Online Help (DynOHelp)

The help texts in DD can be authored using the web interface provided by the DynOHelp Help Admin Tool (HAT) tool, which will update the help texts in the corresponding DD database tables. For more information on how the tool works and interacts with a client system like DD, see the DynOHelp documentation at <https://github.com/eea/eionet.helpadmin>.

### Data Dictionary import tool

The import tool produces definitions of datasets, tables, and data elements in XML files, which can be imported into DD using the DD web interface.

### QA and Converters (XMLCONV)

Retrieves table definitions from DD over using the “getDSTables” method from the DD XML-RPC service interface.

## Source code structure

The code-base for DD is structured as following.

|  |  |
| --- | --- |
| Name | Comment |
| Acl | Access control lists for the application |
| Custom | Html header files for use in the DD web application |
| Doc | This and other documents describing DD |
| Msaccess | MS Access template file for the import application |
| Sql | Database creation scripts |
| Src/main/java | Main source code for the back-end application, described further below |
| Src/main/webapp | Main source code for the front-end web application |

The sub-directories of Src/main/java follow the DD Java class hierarchy which reflects from the following list of all DD java packages:

|  |  |
| --- | --- |
| Name | Comment |
| eionet.meta | This is the root package of DD’s Java source. Classes in this package deal with the basic objects of dd business logic, but it also contains classes representing several of DD’s Java servlets. |
| eionet.meta.dao | Data access interfaces |
| eionet.meta.dao.domain | Business domain objects |
| eionet.meta.dao.mysql | Implementation of data access interfaces for MySQL server |
| eionet.meta.dbschema |  |
| eionet.meta.doc | Used for the Eionet Documentation Module (eionet.doc) |
| eionet.meta.exports | This is the root package of DD’s export logic which is responsible for generating outputs of data definitions in different output formats (pdf, xml, etc). Classes in this package deal with output that are cachable. Sub-packages of this package are each dedicated to a concrete output format. |
| eionet.meta.exports.pdf | This package deals with generating outputs in PDF format. |
| eionet.meta.exports.schema | This package deals with generating outputs in XML Schema format. |
| eionet.meta.exports.xforms | This package deals with generating outputs in XForm format. |
| eionet.meta.exports.xls | This package deals with generating outputs in MS Excel format. |
| eionet.meta.exports.xmlinst | This package deals with generating empty XML instance files for the data to report. |
| eionet.meta.filestore | Related to the DD file store, where all user-uploaded files should be stored. |
| eionet.meta.filters |  |
| eionet.meta.harvest | Above it was described how values of complex attributes can be automatically harvested and in User Guide you can learn more about this function. But the function itself is implemented in this package. |
| eionet.meta.imp | The idea of DD Import tool is that it is capable of putting its contents into XML format and DD is capable of importing that format. The logic of this import is implemented in this package. |
| eionet.meta.inservices | DD is able to collect data about reporting obligations from ROD over XML-RPC. This package contains implementation of a framework supporting incoming XML-RPC communication with any outer source, including ROD. |
| eionet.meta.notif | To handle notifications about the country site code allocation via e-mail and UNS. |
| eionet.meta.outservices | Besides being able to download information about reporting obligations from ROD, DD is also able to provide information for ROD and for any other application for that matter. This package implements a framework for carrying out such communications. |
| eionet.meta.savers | DD web interface contains a lot of forms from where the users submit their inputs. This package contains several different handlers of submitted data, each dealing with getting the data stored into the right tables, all business rules taken into account. |
| eionet.meta.scheduled | Handles scheduling of jobs, e.g. import of DD vocabularies provided in RDF. |
| eionet.meta.schemas | Repository implementation for schema files. |
| eionet.meta.service | Data access interfaces- and implementations, and view models for common business domain objects. |
| eionet.util | This is a package containing several different utility and helper classes helping to carry out several routine operations in many places of the DD code. |
| eionet.web | Controllers etc. for the webapp. |

# Infrastructure architecture

This section covers how the software components are implemented in the physical infrastructure consisting of servers, system software, and communication protocols and networks.



## Clients

The main set of clients, which are the users of the DD web interface, the DD import tool, the DynOHelp administration tool and any clients using the DD services interface, are all communicating over HTTP to the Apache HTTP web server.

## The DD application

The requests from the clients are handled by the Apache HTTP web server, which forwards the calls over the Apache JServ Protocol (AJP) to the core of the DD application, implemented on a Tomcat Java application server. Communication with the MySQL database from the core application on the Tomcat application server is done using the JDBC (Java Database Connectivity) API over TCP/IP.

## External services

Authentication calls to the CAS server are made from the Tomcat application server over HTTP.