Charles University in Prague
Faculty of Mathematics and Physics

ODCleanStore

Linked Data management tool

User Manual

Release 1.0.1
January 3, 2013

Authors: Jan Michelfeit
Dušan Rychnovský
Jakub Daniel
Petr Jerman
Tomáš Soukup

Supervisor: RNDr. Tomáš Knap

Contents

1. Introduction

The advent of Open Data[1] and Linked Data[2] accelerates the evolution of the Web into an exponentially growing information space[3] where the unprecedented volume of data will offer information consumers a level of information integration and aggregation agility that has up to now not been possible. Data consumers can now "mashup" and readily integrate information in myriads of applications.

Indiscriminate addition of information, however, comes with inherent problems, such as the provision of poor quality, inaccurate, irrelevant or fraudulent information. All will come with an associate cost of the data integration which will ultimately affect data consumer's benefit and Linked Data applications usage and uptake.

To overcome these issues, we present a framework enabling management of Linked Data – data cleaning, linking, transformation and quality assessment – and providing applications with a possibility to consume the stored cleaned and integrated data, which reduces the costs of application development.

1.1 What is ODCleanStore

In short, ODCleanStore is a server application for management of Linked Data – it stores data in RDF, processes them and provides integrated views on the data.

ODCleanStore accepts arbitrary RDF data through a webservice (together with provenance and other metadata). The data is processed by *transformers* in one of a set of customizable *pipelines* and stored to a persistent store. The stored data can be accessed again through a webservice. Linked Data consumers can send queries and custom query policies to this webservice and receive (aggregated/integrated) RDF data relevant for their query, together with information about provenance and data quality. Overview of ODCleanStore is depicted on Figure 1.1.

ODCleanStore is developed at the Charles University in Prague, Faculty of Mathematics and Physics as part of the OpenData.cz initiative and the LOD2.eu project and published as a free software under Apache License 2.0. The project is hosted at SourceForge at

http://sourceforge.net/p/odcleanstore/.

[1] http://opendatahandbook.org/

[2] http://www.w3.org/standards/semanticweb/data; http://linkeddata.org/

[3] See the Linked Open Data Cloud at http://richard.cyganiak.de/2007/10/lod/

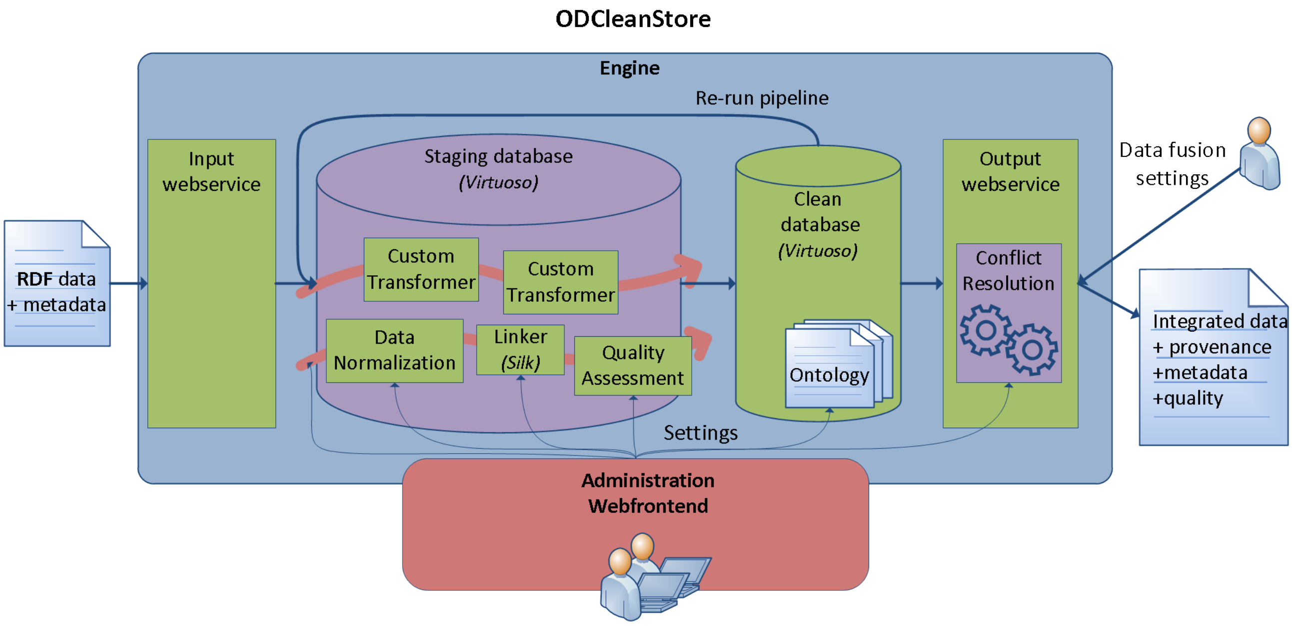


Figure 1.1: Overview of ODCleanStore architecture

1.2 How to Read This Document

This document is a user manual with basic description od ODCleanStore and detailed instructions on how to access and work with the application from the perspective of a user. Chapters 1 and 2 give a basic description of what ODCleanStore is and how it works, while Chapter 3 describes user roles and will guide you to other parts of this manual relevant for your user role.

If more detailed information is needed, please refer to related documents "Administrator's & Installation Manual" and "Programmer's Guide".

1.3 Linked Data Framework

The goal of the OpenData.cz initiative is to build an open data infrastructure in The Czech Republic. It would provide public data in a form that allows access to anyone at any time and allows to combine it freely. This would allow the creation of applications that the public really needs.

ODCleanStore is a part of the Linked Data Framework developed under the OpenData.cz initiative. The main three parts of the framework are *Data Acquisition* module, *Data Aggregation and Cleaning* module and *Data Visualization and Analysis* module.

The Data Acquisition module[4] will be able to crawl webpages and scrape structured data from webpages and other sources (such as XLS spreadsheets). This data is converted to RDF and sent to the Data Aggregation and Cleaning module represented by ODCleanStore. ODCleanStore processes the data, stores it and provides access to it. The Visualization and Analysis module will query ODCleanStore and provide a human-friendly interface to end users.

[4] http://strigil.sourceforge.net/

1.4 Examples of Deployment

ODCleanStore is planned to be deployed together with the Data Acquisition module represented by project Strigil[4] which would feed up-to-date data to ODCleanStore. However, thanks to the use of standard formats for communication with the input/output webservices, ODCleanStore can be deployed with any other third-party application for data feeding or consuming.

In general, ODCleanStore is intendend to be used whenever there are multiple sources of (semi-)structured data convertible to RDF that need to be integrated. ODCleanStore can be used for academic purposes, "mashup" applications, or even deployed in an enterprise environment.

A real-world deployment is planned for storing public contracts data published by the public administration of the Czech Republic as part of the OpenData.cz initiative. Another deployment will be for internal use in students' projects at the Charles University in Prague.

2. How It Works

ODCleanStore consists of *Engine*, *Input Webservice* and *Output Webservice* (both run as part of the Engine), and administration webfrontend. The Engine processes incoming and stored data using *transformers*. A transformer is a pluggable Java class implementing a defined interface; several transformers ship with ODCleanStore, such as Quality Assessment, Linker or Data Normalization.

2.1 Data Lifecycle

The lifecycle of data inside ODCleanStore is as follows:

1. RDF data (and additional metadata) are accepted by Input Webservice and stored as a named graph to the *dirty database*. Data can be uploaded by any third-party application registered in ODCleanStore.
2. Engine successively processes named graphs in the dirty database by applying a pipeline of transformers to it; the applied pipeline is selected according to the input metadata.
3. Each transformer in the pipeline may modify the named graph or attach new related named graphs (such as a named graph with mappings to other resources or results of quality assessment).
4. When the pipeline finishes, the augmented RDF data are populated to the *clean database* together with any auxiliary data and metadata created during the pipeline execution.
5. Data consumers can use Output Webservice to query data in the clean database. Output Webservice provides several basic types of queries – URI query, keyword and named graph query; in addition, metadata about a given named graph can be requested. The response to a query consists of relevant RDF triples together with their provenance information and quality estimate. The query can be further customized by user-defined conflict resolution policies.
   Data in the clean database can be also queried using the SPARQL query language. While SPARQL queries are more expressive, there is no direct support for provenance tracking and quality estimation.
6. When transformer rules change, the administrator may choose to re-run a pipeline on data already stored in the clean database. Copy of this data is created in the dirty database where it is processed by the pipeline. After that, the processed version of data replaces the original in the clean database.

2.2 Administration Frontend Features

The administration webfrontend enables

- management of user accounts,
- management of pipelines, transformers and transformer rules,
- management of ontologies,
- monitoring of inserted data and the state of Engine,

- management of other settings, such as default conflict resolution policies for queries.

2.3 Summary of Features

- Administration in a simple web interface.
- Input and Output Webservices communicate in standard formats - Input Webservice accepts RDF/XML or TTL, Output Webservice provides results in HTML, TriG and RDF/XML formats.
- Highly customizable pipelines for incoming data processing. Different pipelines can be used for different data sources.
- Data can be processed before they are stored to a persistent store but also when they are already stored if neccessary.
- Ships with several predefined transformers for use in data-processing pipelines: Data Normalization (transformations of data), Quality Assesment (estimates quality of data based on a set of rules), Linker (links RDF resources representing the same entity or otherwise related). All these transformers can be managed in the web administration interface.
- Support for ontology management. Mappings between ontologies can be defined in order to integrate heterogeneous data. Also, rules for transformers can be automatically generated from ontologies.
- Data consumers can query for all data about a given resource or use the keyword search.
- Response to a query includes provenance information and quality estimate of each RDF triple in the result. More provenance metadata can be requested. Conflicts that arise when integrating data are solved at query time according to user-defined policies.

3. User Roles

Data consumers accessing Output Webservice (see Section 5.3) do not need to have an account in ODCleanStore; these users have a special role User (USR). Other users working with ODCleanStore need to have an account and their permissions are based on the roles they are assigned. This chapter describes all the roles recognized by ODCleanStore.

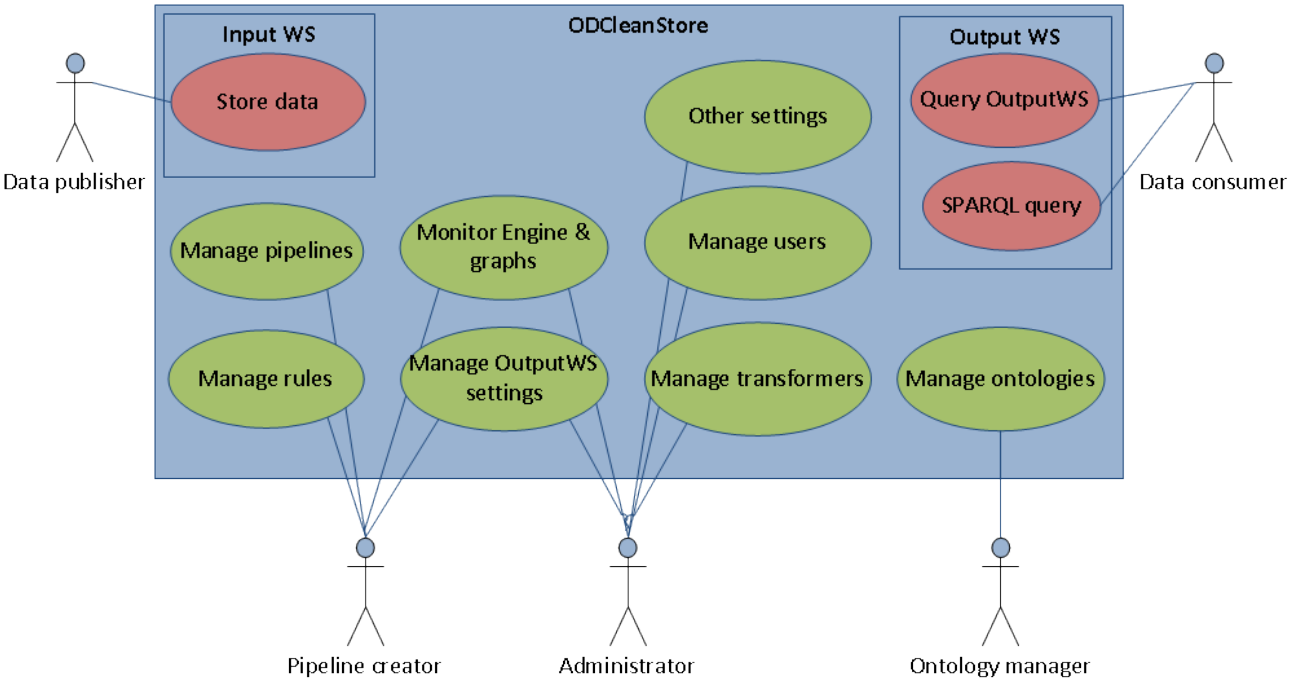


Figure 3.1: Overview of roles in ODCleanStore

3.1 Administrator (ADM)

Administrator has privileges to manage user accounts, assign roles and manage system-wide settings such as

- transformers that can be used in pipelines created by pipeline creators,
- settings of Output Webservice (default aggregation policies, etc.),
- URI prefixes that can be used in settings and queries.

In addition, the administrator is authorized to edit pipelines and rules created by pipeline creators.

More information, e.g. about adding transformers, can be found in the related document Administrator's & Installation Manual.

Most relevant sections of this document: Chapter 4 Administration Frontend.

3.2 Ontology Creator (ONC)

The ontology creator can import and edit ontologies registered in the system. The ontology creator is also responsible for inserting mappings (`owl:sameAs` links) between ontologies.

Most relevant sections of this document: Section 4.6 Ontology Management.

3.3 Pipeline Creator (PIC)

The pipeline creator can create input data processing pipelines. This includes creating new pipelines, assigning transformers to them (Section 4.2) and also creating rules for the transformers (Section 4.3). In addition, pipeline creator can monitor state of graphs sent to ODCleanStore and errors that occur during pipeline processing (Section 4.4).

Every pipeline creator is allowed to create custom pipelines and rule groups for predefined transformers. The pipeline creator has a read-only access to other creators' pipelines and rules (and can use such rules in custom pipelines), however rules and pipelines can only be edited by their author. The only exception is the administrator, who can edit arbitrary pipelines and rule groups.

The same principle applies for inserted graphs management – pipeline creator can delete or re-run pipeline for graphs that were processed by a pipeline created by this pipeline creator, while administrators are authorized for manipulation with all graphs.

Most relevant sections of this document: Sections 4.2 Pipeline Management, 4.3 Transformer Rules, 4.4 Engine & Inserted Graphs Monitoring and 6.3 Executing Pipelines on the Clean Database.

3.4 Data Producer (SCR)

The data producer can use Input Webservice (Section 5.2) to insert new data to ODCleanStore. The system keeps track of which data were inserted by which data producer.

Most relevant sections of this document: Sections 5.2 Data Producer and 6.1 Input Processing.

3.5 Data Consumer (USR)

The data consumer can use Output Webservice (Section 5.3) to ask queries over the data in the clean database. This role is special in that users in this role do not need to have an account (any user using the Output Webservice is automatically assigned the USR role).

Most relevant sections of this document: Sections 5.3 Data Consumer and 6.2 Stored Data Structure.

4. Administration Frontend

4.1 Administration Frontend Overview

Administration Frontend is the tool for managing ODCleanStore. It covers configuration of all standard components. It is restricted to authorized users only.

The administration frontend controls various entities, allows the user to set different attributes and perform actions on those entities. Several terms and designations are used repeatedly in the frontend, however, their meanings do not change, therefore make sure to be familiar with them as they might not be described hereafter.

Common attributes

Label A unique human readable identifier of the related entity.

Description A description for user's purposes and better comprehension of semantics of the related entity.

Author The username of a the originator / creator of the related entity.

Common actions

Delete Remove the related entity irrevertably from the system.

Detail View details and form for editing the related entity. For some entitis shows also entities related to the edited entity.

Rerun affected graphs Queues all graphs affected by the entity for pipeline processing, i.e. the graphs will be processed again by their respective pipeline.

The frontend is divided into several separate sections of logicaly related controls. The main menu bar at the top of the page can be used to switch between those sections.

Figure 4.1: Main menu

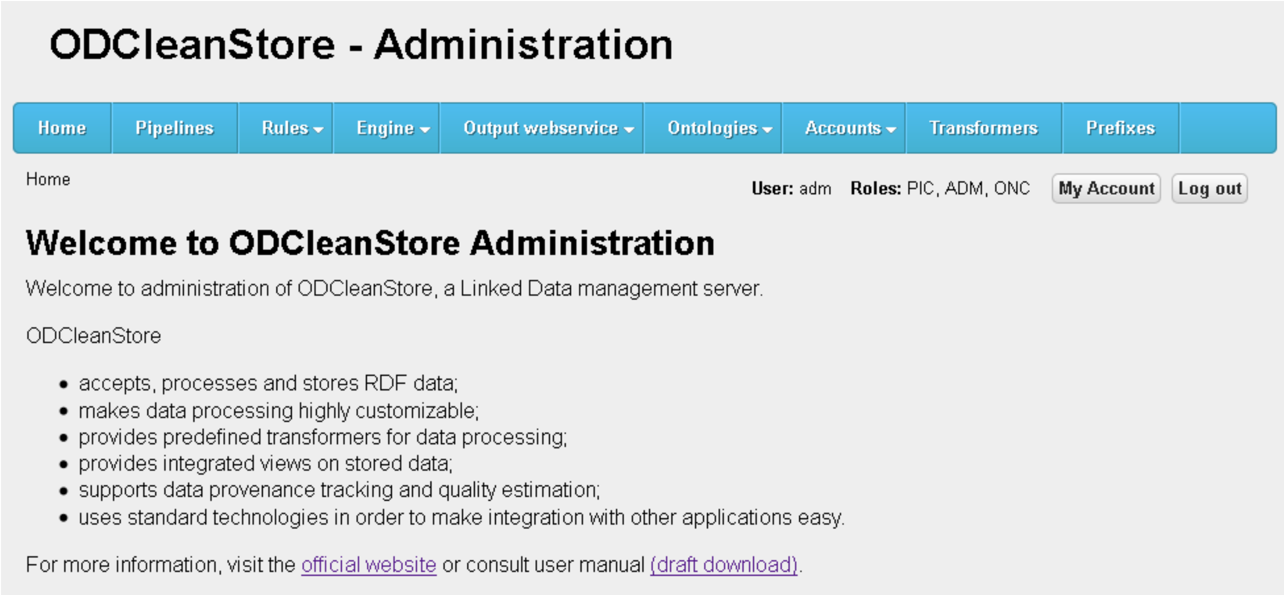


Figure 4.2: Administration Frontend after login

4.2 Pipeline Management

New incoming data (in form of a named graph) accepted by Input Webservice are passed through a pipeline consisting of transformers. In this section of the administration frontend it is possible for the user to specify different pipelines. Individual pipelines can incorporate different already existing transformers. To edit the structure of a pipeline, view its detail.

Individual Transformer Instances

In a detail page of any particular pipeline there is a list of transformers assigned to it. Each assignment composes of these fields:

Required	Field	Description
required	transformer/label	an existing transformer label
required	configuration	configuration passed to an instance of the above selected transformer
implied	allow to be run on clean DB	As the importance of data modification that the pipelines can cause differs based on what database it is running upon, it is left for the user to decide whether a concrete transformer should be allowed to run on clean database (in addition to running on dirty database). Some transformations do not even make sense when working with clean database.
required	place in pipeline before	Determines when the transformer will be run in respect to other transformers in the pipeline

The detail page of the assigned Quality Assessment, Data Normalization, Linker transformers allows user to specify what rule groups are assigned to the transformer in the related pipeline.

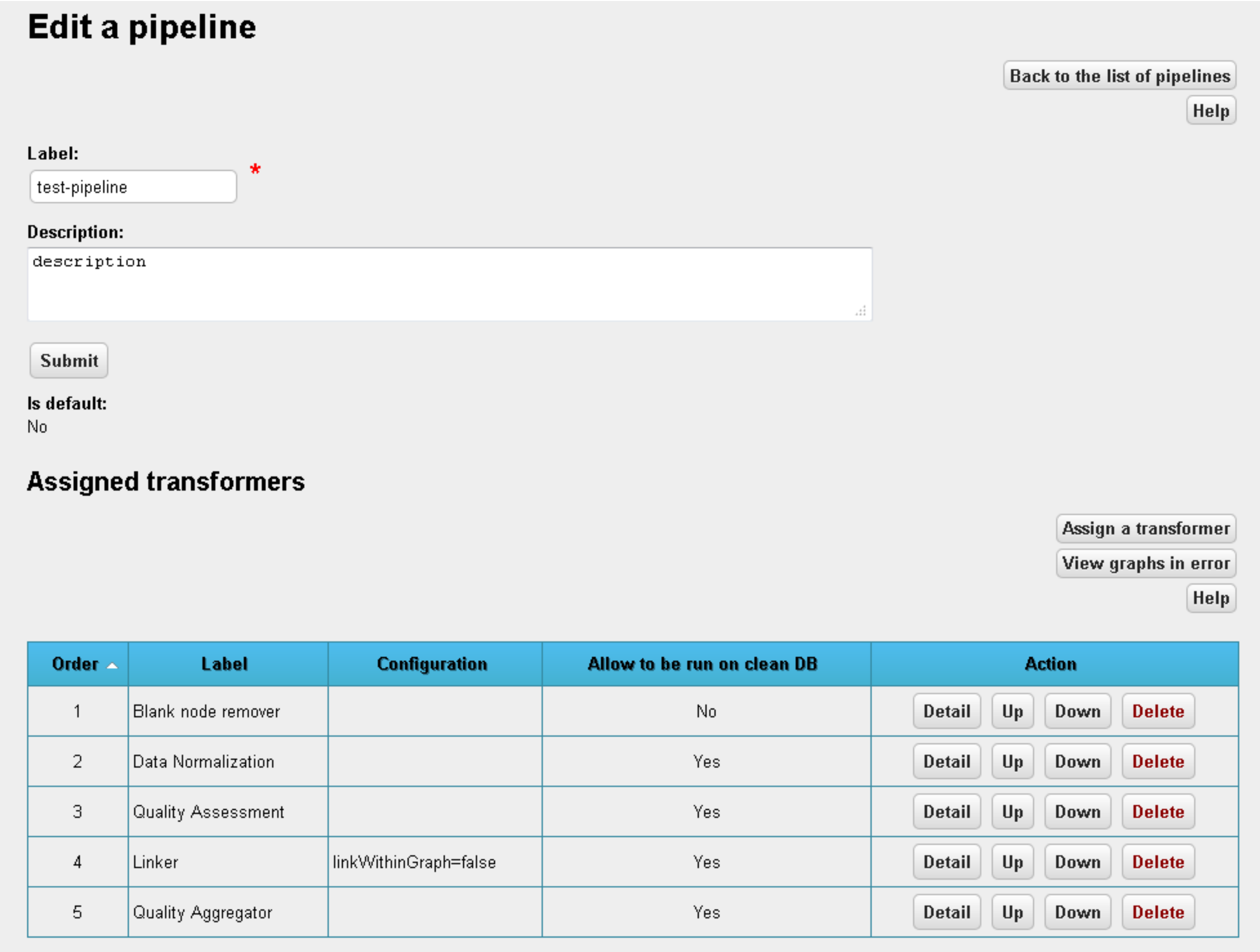


Figure 4.3: Pipeline editing

4.2.1 Predefined Transformers

Several transformers are included in ODCleanStore by default. This section provides their overview.

4.2.1.1 Quality Assessment

This transformer assigns a quality indicator to the processed named graph based on data properties contained in it. It will be further described in section 4.3.1.

4.2.1.2 Quality Aggregator

This transformer assigns a quality indicator to the publisher of the processed named graph based on quality of all graphs stored in the database and sharing this publisher. It will be further described in section 4.3.1.

4.2.1.3 Data Normalization

This transformer can be used to modify data contained in the processed graph. The main reason to allow modifications is to be able to cope with situations when data from different sources have different forms. It is also useful to preprocess data to better suit the rest of the transformation process and future queries of other users. For more information, see section 4.3.2.

4.2.1.4 Linker

Purpose of this transformer is to identify related information and create links that represent the relation. To find out how to control this transformer see section 4.3.3.

4.2.1.5 Blank Node Remover

This transformer replaces all blank nodes in `payload` named graph with unique URI resources. The transformer guarantees that occurrences of the same blank node withing the transformed graph (and only this graph) will be assigned the same URI.

The URIs generated in place of blank nodes have form *<prefix><randomUUID>-<node number>*. The prefix may be given in `Configuration` field of the transformer instance as "`uriPrefix=`*<URI prefix>*" on a single line. If the prefix is not specified, the concatenation of `input_ws.named_graphs_prefix` configuration option value and "getResource" is used as the default value.

4.3 Transformer Rules

There are a few types of transformers predefined for most common data handling in pipelines. Namely:

- *Quality Assessessment* transformer (Section 4.3.1)
- *Data Normalization* transformer (Section 4.3.2)
- *Linker* transformer (Section 4.3.3)

These transformers are configured through groups of rules. Each instance of any of these predefined transformers can accept multiple groups of rules. That way it is possible to simply assign all interrelated rules to a certain instance of a transformer while it is still possible to avoid duplication of rules in different groups.

Figure 4.4: Example of a transformer rule group overview page

4.3.1 Quality Assessment

Quality Assessor

Quality Assessor is a special transformer that assigns a score to each graph based on coefficients of different patterns present in the graph to reflect to what degree the data contained in it comply to a certain policy. The resulting Quality Assessment score is used at query time to calculate quality of results – see Section 5.3.7.

To be able to configure individual instances of Quality Assessor a group of rules needs to exist. To create one enter the Quality Assessment section reachable from Rules submenu.

Here the user can prepare groups of rules to be assigned to instances of Quality Assessor. Each group is identified by its label and can (and should) come with a description of its semantical significance.

On the detail page, one can specify individual rules contained in the related group. Each rule consists of a `GroupGraphPattern`[1] filter, quality decrease coefficient and description, as described in Table 4.1.

	Filter	Coefficient	Description
	`GroupGraphPattern [GROUP BY ... [HAVING ...]]`	$x \in [0,1]$	description
e.g.:	`{{?s anatomy:limbs ?o} FILTER (?o > 4)}`	0.4	Too many limbs

Table 4.1: Quality Assessment rule fields

Any snippet of SPARQL[2] to which "`SELECT * FROM ... WHERE`" can be prepended is a valid filter and describes a property of a named graph that the author of the rule finds defective.

Quality Aggregator

Quality Aggregator is a special transformer that accumulates quality score values of all the graphs corresponding to one publisher. It then calculates an average value and assesses this aggregated quality to the publisher.

4.3.2 Data Normalization

Data Normalizer is a special type of transformer aimed to be applied early in the whole data evaluation process to simplify work of other transformers. Its main goal is to remove inconsistencies in forms the data is provided in.

In the Data Normalization section reachable from Rules submenu, one can prepare groups of rules to be assigned to instances of Data Normalizer. Each group is identified by its label and can (and should) come with a description of its semantical significance.

The detail page of a group serves the user as means of specification of individual rules contained in the selected group. Each rule is essentially a sequence of *SPARUL*[3] modifications

[1] http://www.w3.org/TR/rdf-sparql-query/#rGroupGraphPattern
[2] http://www.w3.org/TR/rdf-sparql-query/#grammar
[3] http://www.w3.org/Submission/SPARQL-Update

put by `MODIFY`, `INSERT` and/or `DELETE`. New rule represents an empty sequence upon its creation.

Similarly as with the rules themselves the detail page of a rule allows the user to construct any arbitrary sequence of modifications. Components of the rule (members of the sequence) can be added by specifying their type (either `MODIFY`, `INSERT` or `DELETE`), modification (SPARUL snippet stripped off of initial `MODIFY` / `INSERT INTO` / `DELETE FROM` clauses); e.g.,

```
{?s ?p1 ?o2} WHERE {?s ?p1 ?o1.  ?o1 ?p2 ?o2.}.
```

Expectedly triples (`?s ?p1 ?o2` in the example) are inserted into (deleted from) the current graph when the type of the component is `INSERT` (`DELETE`). Effects are immediate in respect to consecutive applications of other components of the same rule or other rules to the graph. Another example would be:

```
{?s ?p ?o} WHERE {GRAPH $$graph$$ {SELECT ?s ?p 'Y' AS ?o WHERE {?s ?p 1}}},
```

where `$$graph$$` in `GRAPH $$graph$$` is a place holder for name of the graph being currently processed and can be used for subqueries that need to be enclosed in `GraphGraphPattern`[4].

Note that when there is no subquery, the `$$graph$$` placeholder is optional and it is not necessary to use the placeholder at all.

4.3.3 Linker

Linker is a special trasformer. Its main purpose is to interlink URIs which represent the same real-world entity by generating `owl:sameAs` links. It can be also used for creating other types of links between differently related URIs.

To be able to configure individual instances of Linker a group of rules needs to exist. To create one enter the Linker subsection of the Rules management page.

Here the user can prepare groups of rules to be assigned to instances of Linker. Each group is identified by its label and can (and should) come with a description of its semantical significance.

On the detail page, one can specify individual rules contained in the related group. Fields to be filled in for each rule are described in the table at the end of this section. For further details of their meaning see Silk-LSL specification[5].

Linkage rule an be created in Silk Workbench[6] and its `LinkageRule` element copy-pasted into corresponding field. More convenient way is to import the whole rule from XML file using the "Choose file" and "Import" buttons. For further editation in Silk Workbench the rule can be exported to XML file again using the "Export" button.

Created rule will not produce any links, until it has its outputs assigned. This can be done on the rule detail page after submitting a new rule or when editing an existing one. Two types of outputs can be assigned to a linkage rule, database outputs and file outputs.

[4] http://www.w3.org/TR/rdf-sparql-query/#rGraphGraphPattern
[5] http://www.assembla.com/wiki/show/silk/Link_Specification_Language
[6] https://www.assembla.com/spaces/silk/wiki/Silk_Workbench

Required	Field	Description
required	label	
optional	description	
required	Link type	Type of the link to create (typically `owl:sameAs`)
optional	Source SPARQL restriction	Restriction on URIs from the transformed data, written in SPARQL.
optional	Target SPARQL restriction	Restriction on URIs from the clean database, written in SPARQL.
required	Linkage rule	Linkage rule itself, written in Silk-LSL. XML fragment `<LinkageRule>...</LinkageRule>` is expected.
optional	Filter threshold	Real number, serves as a global threshold, links with lesser confidence will not be sent to any output.
optional	Link limit	Defines the number of links originating from a single data item. Only the n highest-rated links per source data item will remain after the filtering. If no limit is provided, all links will be returned.

Database output serves for storing generated links into the clean database. Minimal and maximal confidence of links to be stored can be specified as real numbers.

File output serves for storing generated links into a file. Minimal and maximal confidence of links to be stored can be specified as real numbers. Filling in filename is required, files will be stored into the transformer directory on the server, their names will be prefixed by identifiers of graphs beeing processed. Two file formats are supported, NTRIPLES[7] and ALIGNMENT[8].

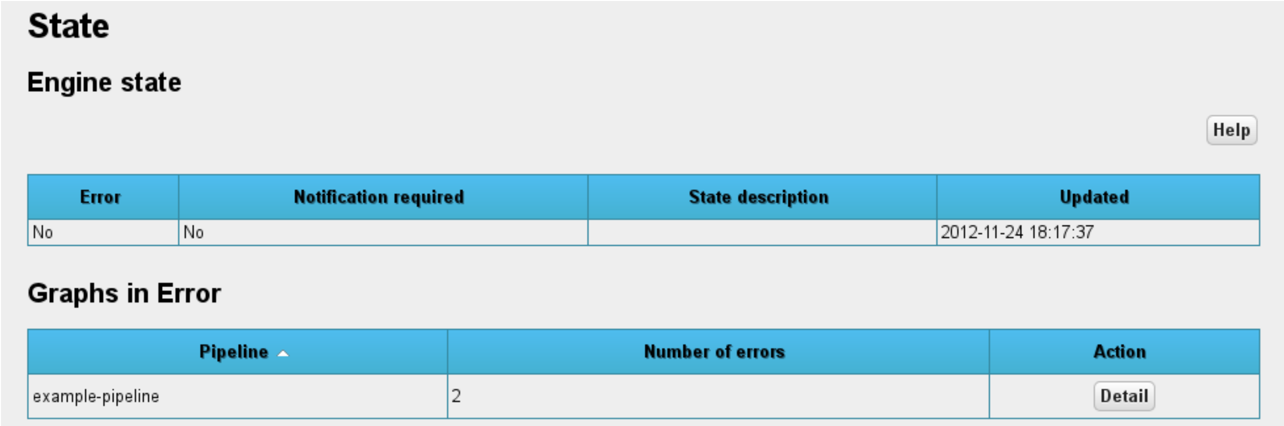
4.4 Engine & Inserted Graphs Monitoring

There is a section dedicated to monitoring an overall state of the engine and graphs stored in the database. It can be found by selecting *Engine* from the main menu and then choosing one of the subsections.

The *State* subsection displays all errors reported by the engine. Be it any failure of the engine itself or a data processing error related to only some of the graphs. The view on this page shows a simplified and well-arranged information about number of erroneous graphs. More exhaustive information will be displayed on the detail pages corresponding to individual pipelines. Each graph processed by the selected pipeline can then be processed by the pipeline once again with the rerun button (this transformation will reflect current state of the pipeline configuration) or it can be deleted. Graphs that are in clean database can in addition be accepted as they are despite the errors if the user considers them irrelevant. There are also shortcut buttons that allow to perform all of these actions on all graphs in one step. The clean database restriction for the accept action still applies so some graphs may not be affected. All of the actions can be invoked by administrator and the author of corresponding pipeline.

[7] http://www.w3.org/2001/sw/RDFCore/ntriples/

[8] http://alignapi.gforge.inria.fr/format.html



State

Engine state

Help

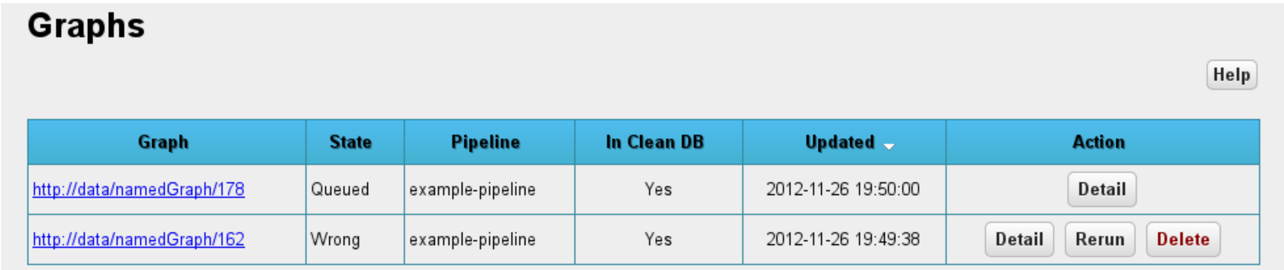
Error	Notification required	State description	Updated
No	No		2012-11-24 18:17:37

Graphs in Error

Pipeline	Number of errors	Action
example-pipeline	2	Detail

Figure 4.5: Engine state overview page

The *Graphs* subsection captures the content of the graph database. The table of all graphs contains their identifiers in form of `URI`, states, pipelines that processed them, information about residence in clean or dirty database, and timestamp of the last update. Each of the graphs can be deleted or rerun in which case it is processed by the pipeline in its current state. The `URI` is a link to output webservice and serves as detailed source of information about the graph.



Graphs

Help

Graph	State	Pipeline	In Clean DB	Updated	Action
http://data/namedGraph/178	Queued	example-pipeline	Yes	2012-11-26 19:50:00	Detail
http://data/namedGraph/162	Wrong	example-pipeline	Yes	2012-11-26 19:49:38	Detail Rerun Delete

Figure 4.6: Engine graphs overview page

4.5 Output Webservice

The output webservice configuration covers default policies for data aggregation. See section 5.3.

The configuration specifies one global behaviour and then it offers the user to override that behaviour for specific properties.

The properties specified in the Label properties section are treated by query execution component as human readable labels of different entities. Simply add new label property by using "add a new property" button and remove any by use of corresponding delete button.

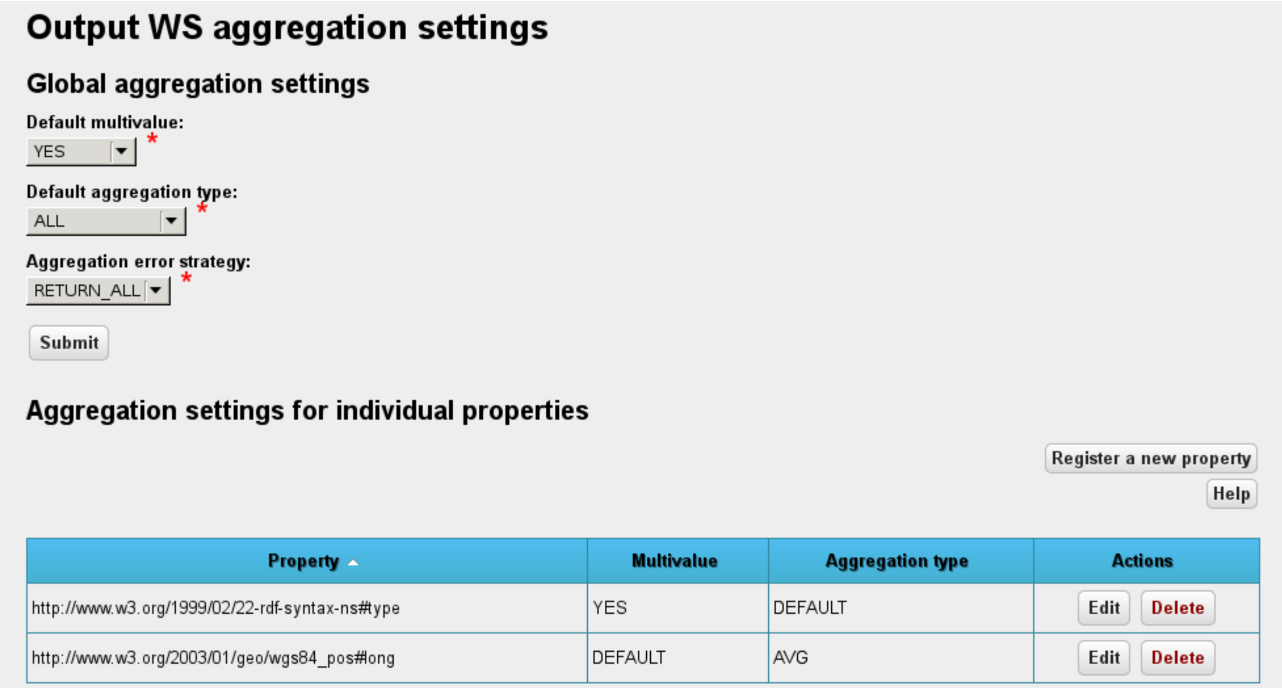


Figure 4.7: Output WS aggregation properties page

4.6 Ontology Management

Ontologies can be used to produce common rules for Quality Assessment, Data Normalization transformers. To load one into the storage one can provide an explicit definition through a text field or by uploading a file containing a valid RDF/XML or TTL ontology definition. The process of rules generation will automatically take place upon ontology submission.

Figure 4.8: Ontologies page

Another benefit of storing ontologies in the database is gain of ability to map properties from one ontology to properties from another with `owl:sameAs`, `owl:equivalentProperty`, `rdfs:subPropertyOf`, `rdfs:subClassOf` or a custom `URI` of any other property. This can be further used by Conflict Resolution component to produce more precise results. Such mapping can be added in the section Ontology mappings reachable from Ontologies submenu. In that section a pair of ontologies needs to be selected to restrain to a specific set of properties. After submitting the pair of ontologies a new form is presented where individual properties can be mapped. After filling in `URI`'s of source and target properties, selecting a relation type and submitting a mapping is created. From that point on it will be considered during conflict resolution.

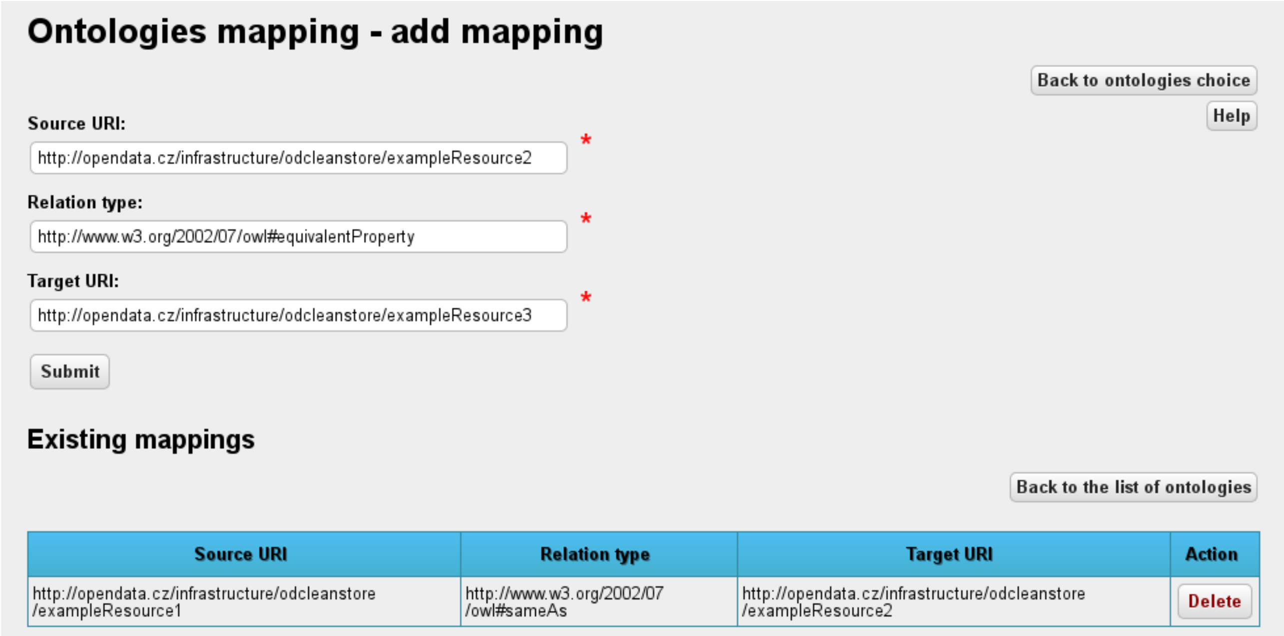


Figure 4.9: Ontologies page

4.7 Accounts

As it has been already mentioned in chapter 3 the frontend verifies user privileges before providing access to different configurations and content. To be able to maintain user roles and permissions that are implied for individual users the frontend administration provides this section. All registered accounts will be displayed in a table with all the information (username, e-mail address, first and second name, roles assigned to the account). The administrator can assign roles and reset password from this overview and related editable pages. For editing the roles simply use *Roles* button and for reseting the password use *New password* button which will prompt you for confimation and then generate a new password and send it via the e-mail address to the user in question if the confirmation is given.

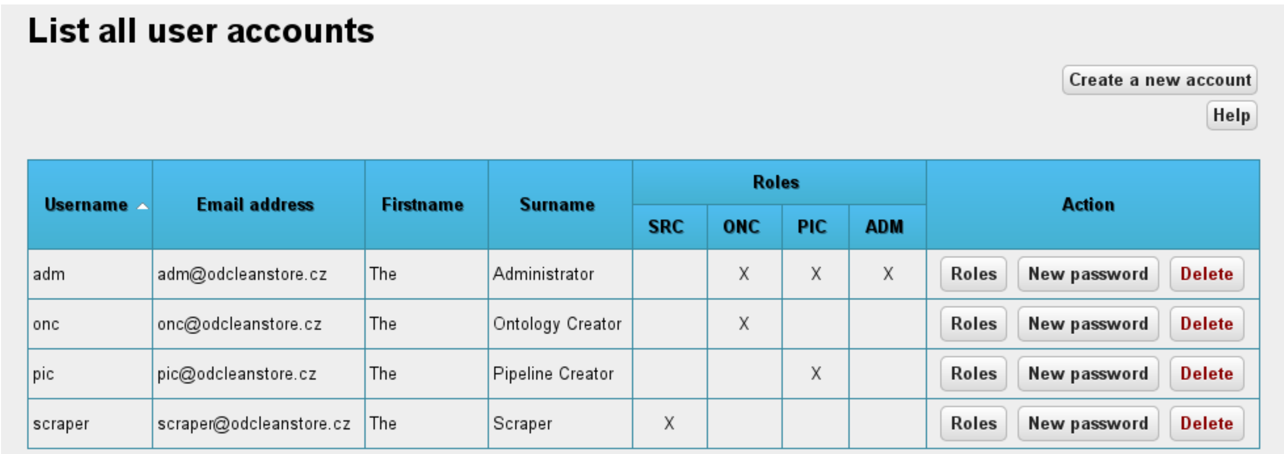


Figure 4.10: Accounts page

My Account

This section is reachable with *My Account* button below the main menu after a succesful login. It displays current user's name, first and second real name, and e-mail address. It is also possible to change the current password through a page to which *Edit my password* redirects.

4.8 Transformer Management

Transformer is a component responsible for data refinement, cleaning, aggregation and other transformations applied to incoming or stored data.

The transformer management screen allows registered users to add, edit or remove transformers. These can be then added to *pipelines* (Section 4.2).

Each transformer definition consists of:

Required	Field	Description
required	Working directory	An (arbitrary) directory dedicated to instances of this transformer. Files may be stored in it.
required	JAR path	Path to a Java Archive containing the transformer declaration.
required	Full classname	Name of the class implementing the transformer. Note that it isn't possible to edit this value at a later time.

If *JAR path* is set to "." then it is handled as a special case and *Full classname* is treated as a built-in transformer.

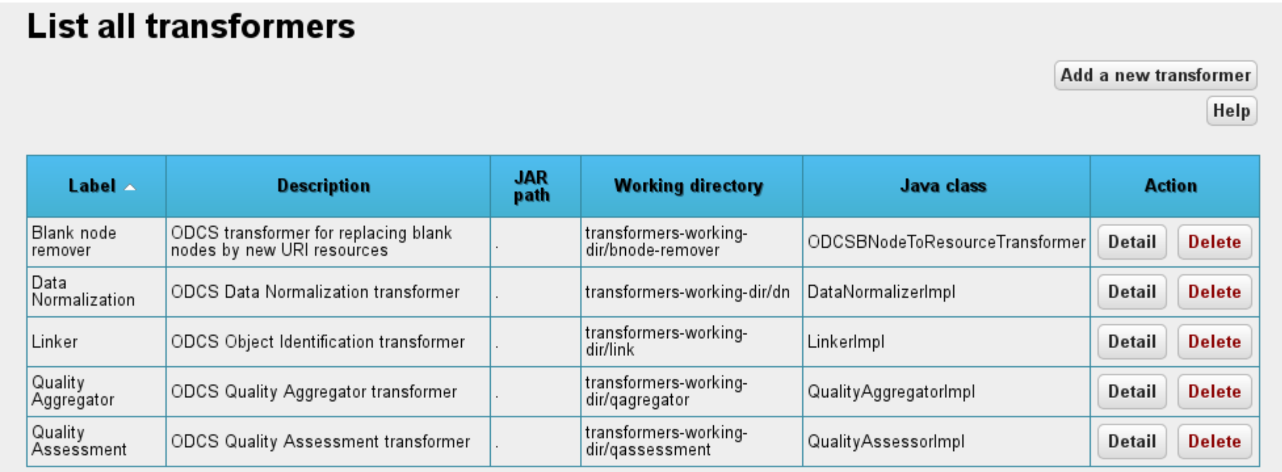


Figure 4.11: Transformers page

4.9 Prefixes

To avoid obligation of full manual URI expansion in *transformer* rules or queries it is possible to maintain set of global *RDF* prefixes that storage recognizes. These can be added with *add a new prefix* button and removed by *delete* button next to the desired target of removal.

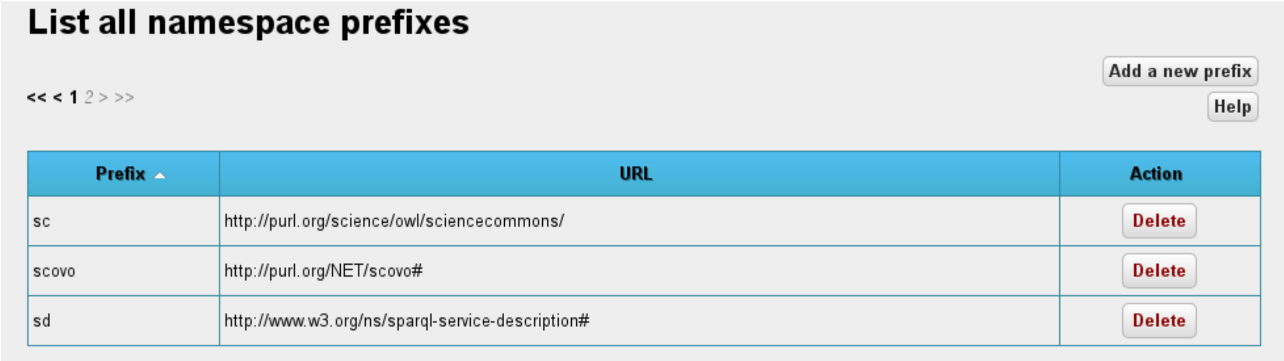


Figure 4.12: Prefixes page

4.10 Configuration Example

In this section a basic concept of ODCleanStore configuration will be illustrated.

It is necessary to log into the frontend with credentials given during the installation. All of the following operations will be possible to be done with the initial user account.

There need to be transformers for the storage to be able to handle incoming data. ODCleanStore comes with its built in transformers that are accessible from the frontend right after the installation. Custom transformers need to be added at this point.

Add the ODCSPropertyFilterTransformer by following steps:

Prepare Transformer

1. Choose `Transformers` from the frontend menu

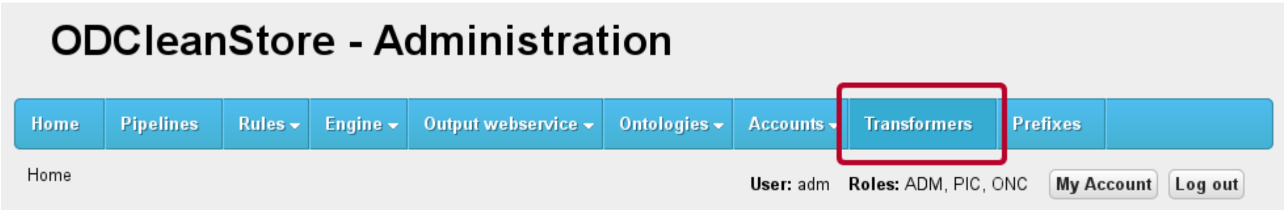


Figure 4.13: Navigating to transformers page using the main menu

2. Click "Add a new transformer"

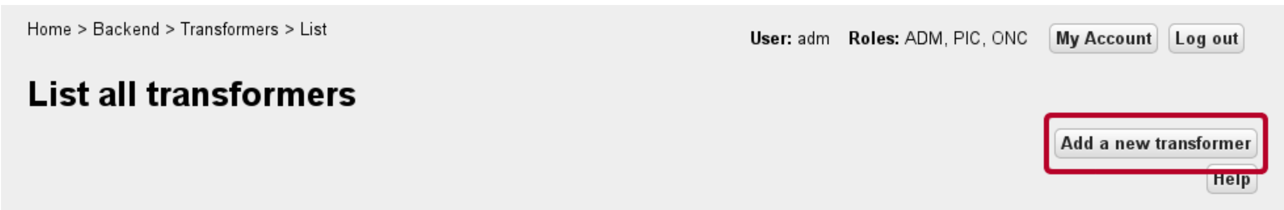


Figure 4.14: Accessing the new transformer definition page

3. Fill in label of your choice
4. Describe its purpose
5. Select the path to the backend JAR
6. Fill in the classname

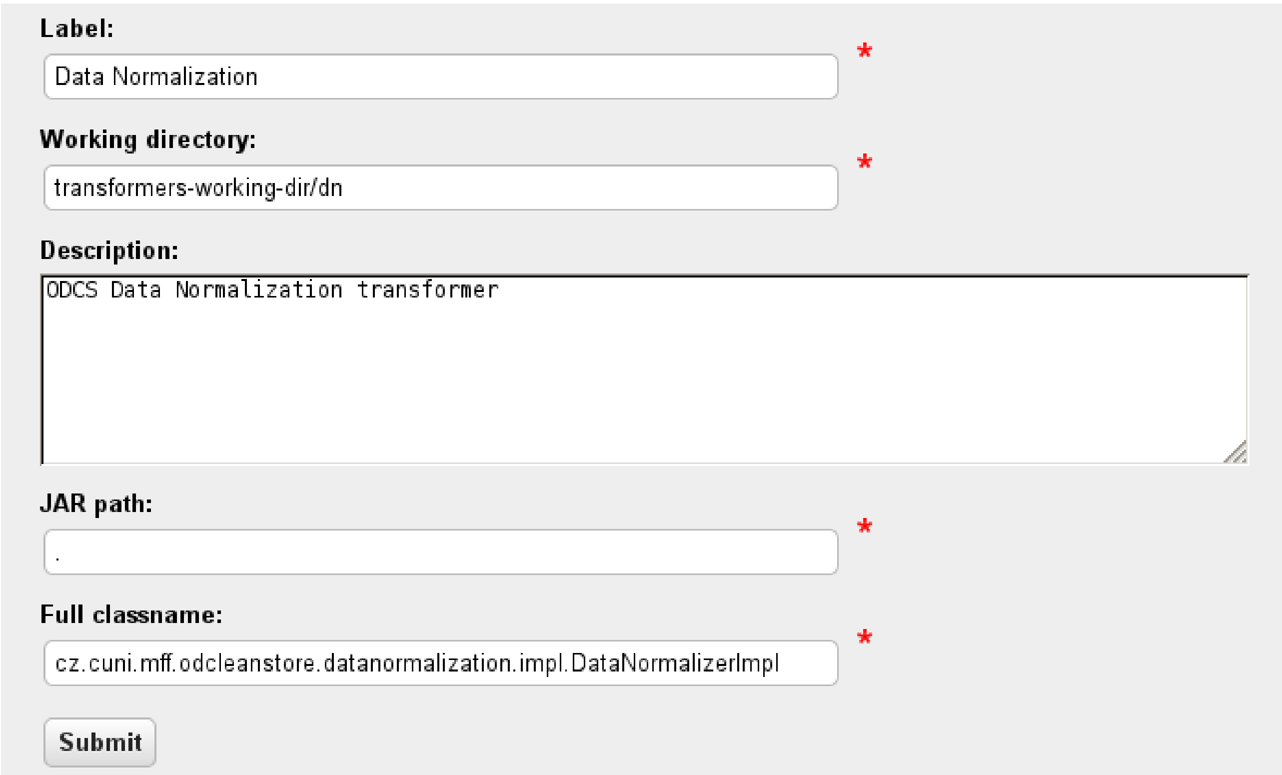


Figure 4.15: Transformer definition after filling in all necessary information

Prepare Rules for Standard Transformer

7. Choose `Rules/Data Normalization` from the frontend menu

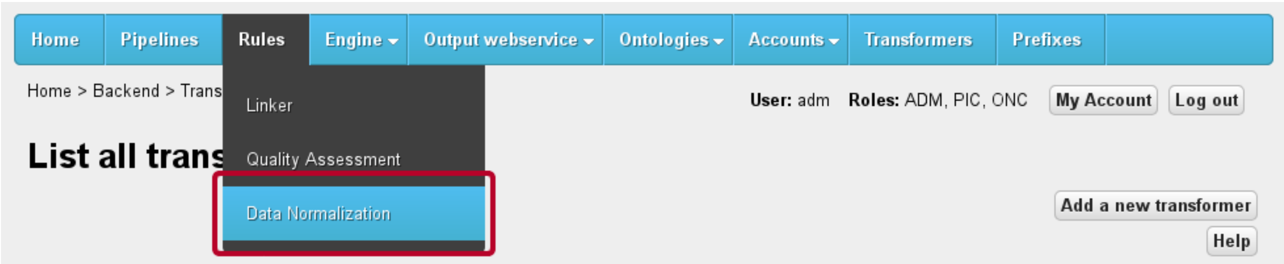


Figure 4.16: Navigating to the rule group management section

8. Click "Add a new rules group"

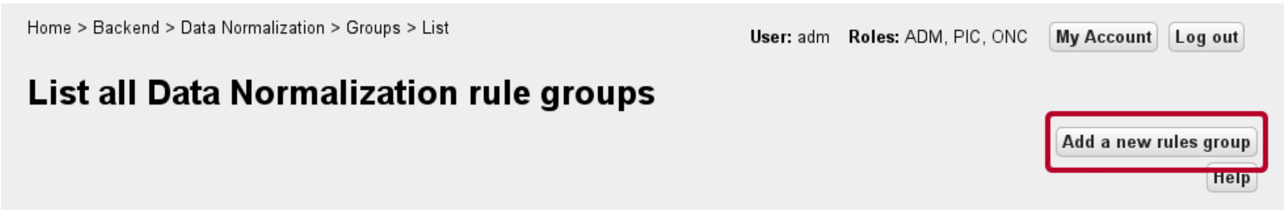


Figure 4.17: Proceeding to define a new group

9. Fill in necessary information and submit it

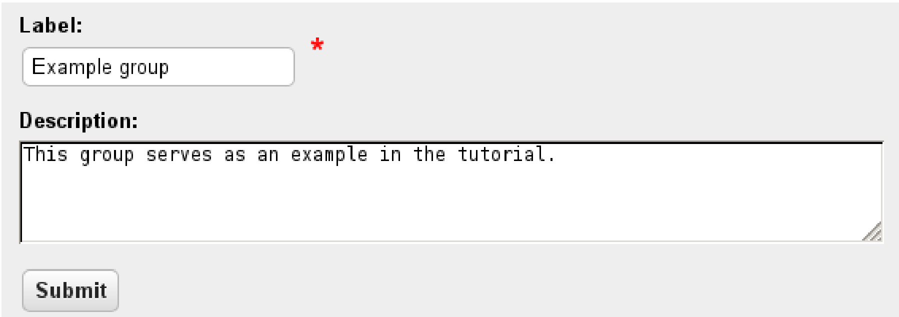


Figure 4.18: Definition of a new rule group

10. Click "Add a new raw rule"

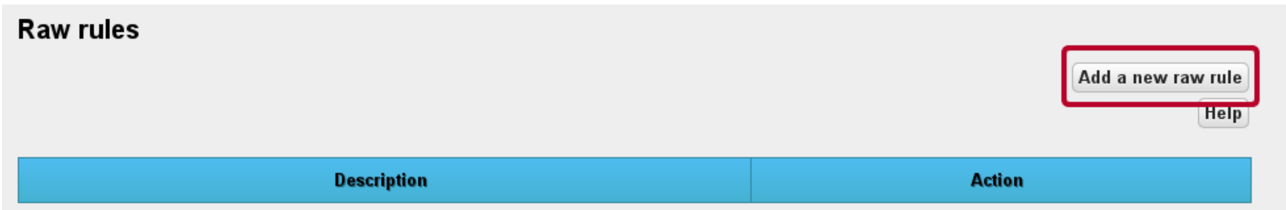


Figure 4.19: Adding a rule to the new group

11. Fill in its description and submit

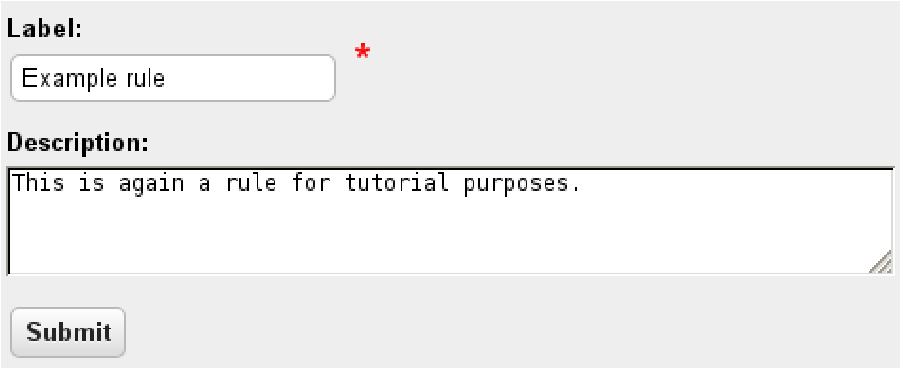


Figure 4.20: Filling in the necessary information

12. Click "Add a new rule component"

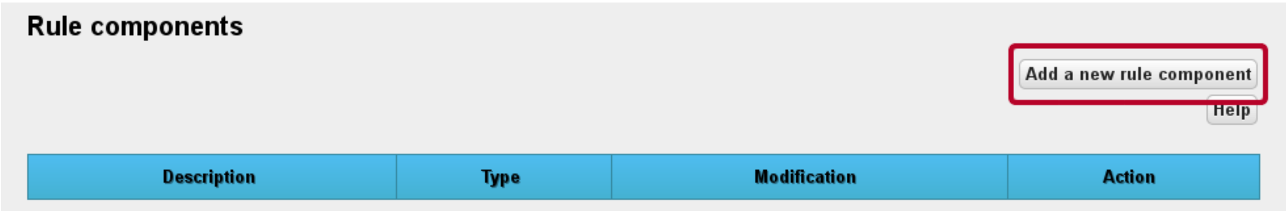


Figure 4.21: Proceeding to definition of individual components

13. Choose the type of the data transformation (`MODIFY`/`INSERT`/`DELETE`)
14. Specify the triples that will modify the graph
15. Describe the meaning of this transformation and submit it

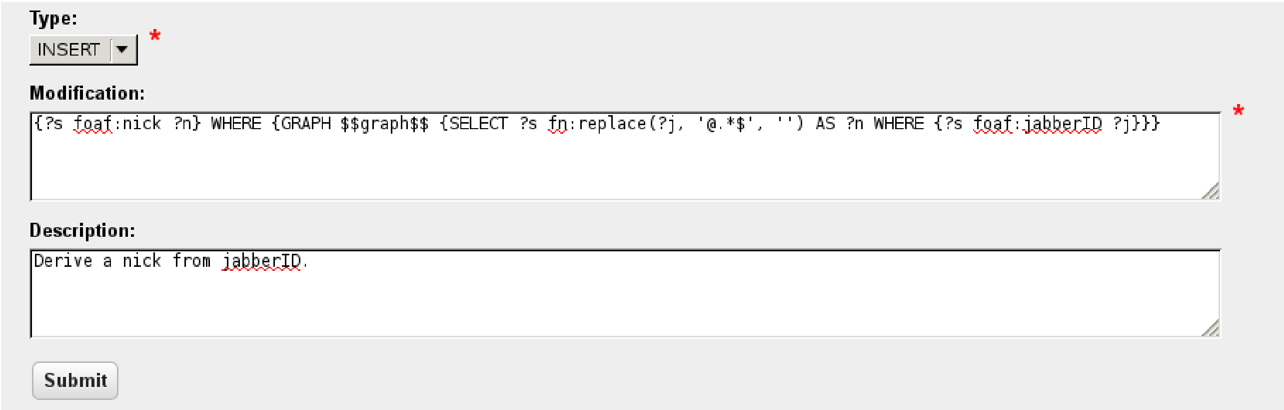


Figure 4.22: Definition of a new Data Normalization component

16. Repeat until the rule is complete

Prepare Pipeline

17. Choose `Pipelines` from the main menu

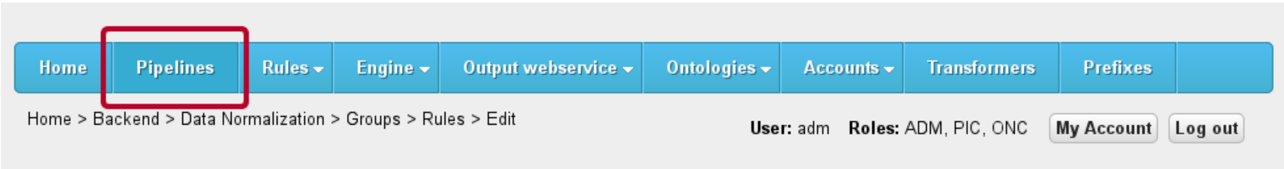


Figure 4.23: Navigating to the pipelines management section

18. Click "Add a new pipeline", fill in the label and description and submit it
19. Click "Assign a transformer"

Figure 4.24: Proceed to assignment of transformers to the pipeline

20. Select one of the transformers
    If there is no option then go back to the **Prepare Transformer** *section*
    To be able to assign rule groups select one of the standard transformers (QualityAssessor, DataNormalizer, Linker)
21. Fill in the configuration needed by the transformer
22. Allow or disallow running on clean DB
23. Select place in the pipeline

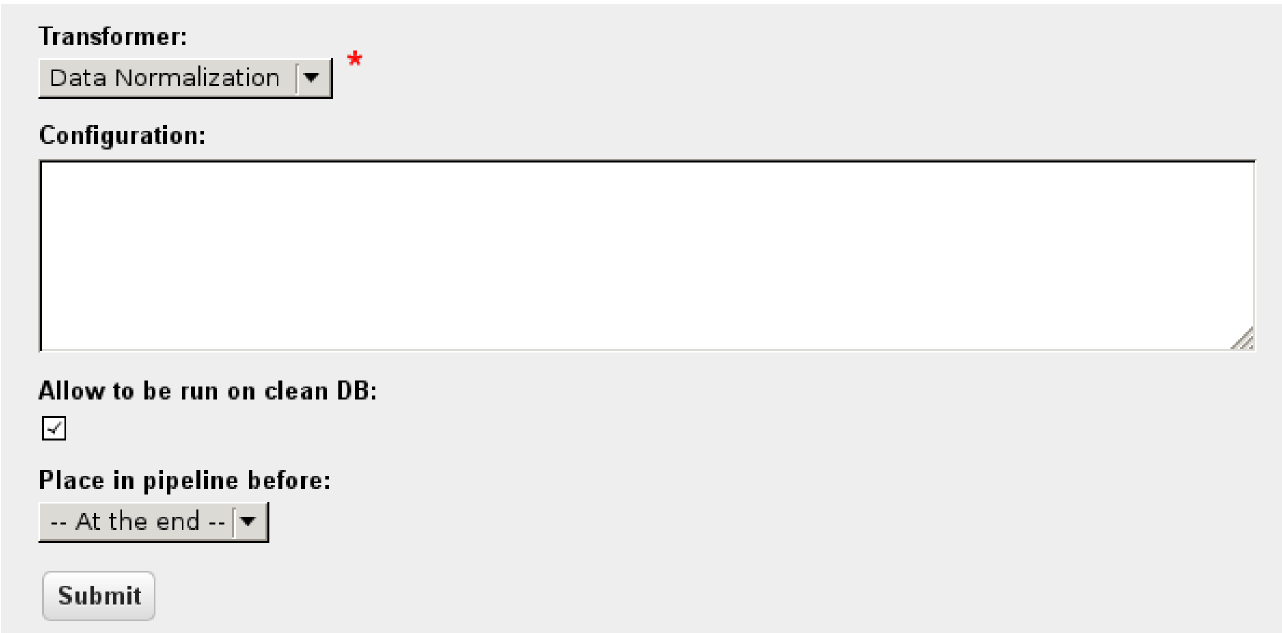


Figure 4.25: Assigning a new instance of previously defined transformer

24. Click “Assign a group” to assign a group of Data Normalization rules

Figure 4.26: Continuing by assigning rule groups to the transformer instance

25. Select the group created earlier

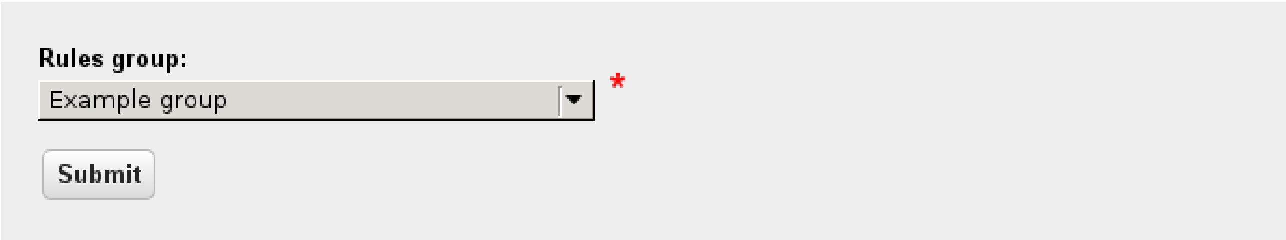


Figure 4.27: Selecting the desired group to be assigned

5. Web Services

5.1 Web Services Overview

ODCleanStore communicates with third-party applications via webservices. Data producers can store data to ODCleanStore through *Input Webservice*, while data consumers may use *Output Webservice* to query the stored & processed data. In addition, stored data can be accessed through a public SPARQL endpoint. Input Webservice requires authorization, Output Webservice and the SPARQL endpoint do not.

5.2 Data Producer

New data can be stored to ODCleanStore through Input Webservice, a SOAP multithreaded webservice that accepts RDF data serialized as RDF/XML[1] or TTL[2] and additional metadata. The webservice requires authorization with a valid user name and password.

The location of Input Webservice can be configured by the `input_ws.endpoint_url` configuration option (see Administrator's & Installation Manual); by default, it is:

<*host*>`:8080/inputws`

See Section 6.1 for more information about how the inserted data are processed and stored.

5.2.1 Request parameters

Table 5.1 enumerates parameters of Input Webservice. All the parameters are required.

Name	Description	Type
`user`	user login name	*string*
`password`	user password	*string*
`payload`	data to insert serialized as RDF/XML or TTL	*string*
`metadata`	metadata about `payload`	see Table 5.2

Table 5.1: Input Webservice parameters

5.2.1.1 Metadata

Table 5.2 lists fields that the `metadata` parameter consists of.

Each request is identified by a unique UUID generated on the client side and sent in the `uuid` field. The client side is responsible for generating different UUIDs for new requests. UUID doesn't change during the whole message transfer nor in case of a repeated request after an exception.

The `dataBaseUrl` field is the base URI for `payload`.

[1] http://www.w3.org/TR/rdf-syntax-grammar/

[2] TTL or Turtle – Terse RDF Triple Language; http://www.w3.org/TeamSubmission/turtle/

Name	Description	Type	Cardinality
`uuid`	UUID string unique for the current request	UUID	1
`dataBaseUrl`	base URI for resolution of relative URIs in `payload`	URI	1
`source`	location of where the data were retrieved from	URI	1..*
`publishedBy`	identifier(s) of the publisher(s) of the data	URI	1..*
`license`	license(s) under which the data are published	URI	0..*
`provenance`	additional provenance metadata serialized as RDF/XML or TTL	RDF/XML or TTL	0..1
`pipelineName`	identifier of the pipeline that should process the inserted data	*string*	0..1
`updateTag`	distinguisher of set of graphs that update each other	*string*	0..1

Table 5.2: Input Webservice `metadata` fields

The `source` field is a list or URIs the data were retrieved from. Typically, this would be URI(s) of webpage(s) the data were scraped from but in general it can be any URI.

The `publishedBy` field is a list of URIs representing the publisher of the data. It can be a well known URI, or, for example, the host part of the source URI (e.g. `http://en.wikipedia.org/` for data scraped from the English Wikipedia).

The `license` field may specify URI(s) representing the license(s) under which `payload` contents and any additional metadata being inserted are published.

The optional `provenance` field can contain additional RDF provenance metadata about contents of `payload`.[3] Base URI for the provenance metadata is the URI of the named graph where `payload` is stored in ODCleanStore.

The optional `pipelineName` field can contain a string identifier of an existing pipeline in ODCleanStore that should be used to process the inserted data. If omitted, the default pipeline is used.

The optional `updateTag` field serves as a distinguisher of data that update an already inserted version of data. If one named graph is to be considered an update of another named graph, both of them must have the same value of `updateTag`. For more information about how updates are detected, see Section 6.1, step 6.

5.2.2 Exceptions

Error during a request to Input Webservice is indicated by throwing an exception. Table 5.3 summarizes exceptions that can occur. In case of such exception, no data or `uuid` value for the interrupted request are stored.

[3]The suggested vocabulary for these purposes is W3P (`http://code.google.com/p/od-w3p/`).

Exception	Code	Description
SERVICE_BUSY	1	Service busy – occurs when maximum limit of concurrent connections is exceeded
BAD_CREDENTIALS	2	Bad credentials (invalid `user` or `password`)
NOT_AUTHORIZED	3	Not authorized – user doesn't have SCR role assigned
DUPLICATED_UUID	4	Duplicate uuid – another request with the same `uuid` value has already successfully finished
UUID_BAD_FORMAT	5	Wrong format of the `uuid` field
UNKNOWN_PIPELINENAME	6	No pipeline with name as given in `pipelineName` exists
OTHER_ERROR	7	Other error; when a new transmission with the same `uuid` as the current `uuid` is started before the current transmission finishes, OTHER_ERROR is thrown; only the new transmission will continue
FATAL_ERROR	8	Fatal error
METADATA_ERROR	9	Invalid metadata – a field has a wrong format or a required field is missing

Table 5.3: Input Webservice exceptions

5.2.3 Java API

Third-party applications can access Input Webservice directly, or use the Java client library provided in ODCleanStore distribution. Add `odcs-inputclient-`*version*`.jar` library to your project and use class `OdcsService` to access Input Webservice programmatically.

Listing 5.1 gives an example of how the client library can be used.

```
try {
  File payloadFile = new File("data.rdf");

  final int BUFFER_SIZE = 1024 * 4;
  char[] buffer = new char[BUFFER_SIZE];
  int count = 0;
  StringBuilder provenancePayload = new StringBuilder();
  InputStreamReader provenanceReader = new InputStreamReader(
          new FileInputStream("provenance-metadata.rdf"), "UTF-8");
  while (-1 != (count = provenanceReader.read(buffer, 0, BUFFER_SIZE))) {
      provenancePayload.append(buffer, 0, count);
  }
  provenancePayload.close();

  Metadata metadata = new Metadata(UUID.randomUUID());
  metadata.setDataBaseUrl(new URI("http://en.wikipedia.org/wiki/Berlin"));
  metadata.getSource().add(new URI("http://en.wikipedia.org/wiki/Berlin"));
  metadata.getPublishedBy().add(new URI("http://en.wikipedia.org"));
  metadata.getLicense().add(new URI("http://creativecommons.org/licenses/by-sa/3.0/"));
  metadata.setPipelineName("examplePipeline");
  metadata.setUpdateTag("example");
  metadata.setProvenance(provenancePayload.toString());
```

```
  OdcsService service = new OdcsService("http://localhost:8088/inputws");
  service.insert("username", "password", metadata, payloadFile, "UTF-8");
} catch (Exception e) {
  e.printStackTrace();
}
```

Listing 5.1: Example usage of Input Webservice client library

5.3 Data Consumer

A consumer of data stored in ODCleanStore can query the database through Output Webservice. The Output Webservice can be queried for data about a given URI resource, queried by keywords, queried for contents of a given named graphs or queried for metadata of a named graph. Conflicts in data returned in response to a query are resolved and the data are fused using policies provided by the user or by the administrator.

Additionally, the user can access the data in the clean database directly using the SPARQL endpoint powered by Virtuoso.[4] This way the data consumer can use the full power of the SPARQL query language, however conflict resolution and provenance tracking is not supported for this type of queries.

Output Webservice

The Output Webservice is a REST webservice which can be accessed using both GET and POST HTTP methods equivalently. The port where the webservice resides can be configured by the `output_ws.port` configuration option (see Administrator's & Installation Manual); by default, it is on port `8087`.

5.3.1 Types of queries

The Output Webservice can be queried for:

1. a resource URI – *URI query*
2. keyword(s) – *keyword query*
3. named graph contents – *named graph query*
4. named graph metadata – *metadata query*

Table 5.4 lists where each type of query can be accessed by default. The exact address can be configured.

More information is available in the Query Execution specification.

5.3.2 Request format

Table 5.5 lists (either GET or POST) parameters than can be used with the URI, keyword and named graph queries. The `uri` parameter is required for URI query, `kw` parameter for keyword

[4] http://virtuoso.openlinksw.com/

Query	URI	Example of a query
URI	<*host*>`/uri`	http://localhost:8087/uri ?uri=http%3A%2F%2Fexample.com
Keyword	<*host*>`/keyword`	http://localhost:8087/keyword?kw=keyword
Named graph	<*host*>`/namedGraph`	http://localhost:8087/namedGraph ?uri=http%3A%2F%2Fexample.com
Metadata	<*host*>`/metadata`	http://localhost:8087/metadata ?uri=http%3A%2F%2Fexample.com

Table 5.4: Types of queries

query. Other parameters are optional.

Name	Description	Possible values	Default value
`uri`	searched URI; *used only with URI and named graph query*	*string*	*N/A*
`kw`	searched keyword(s); *used only with keyword query*	*string*	*N/A*
`format`	format of the response	`html`, `trig`, `rdfxml`	`html`
`aggr`	default aggregation method	*string*	`ALL`
`es`	error strategy – handling of values for which aggregation fails	`IGNORE`, `RETURN_ALL`	`RETURN_ALL`
`multivalue`	default multivalue setting	0, 1	0
`paggr[`*property*`]`	aggregation method for the given property; example: `paggr[rdfs%3Alabel]=ANY`	*string*	*N/A*
`pmultivalue[`*property*`]`	multivalue setting for the given property; example: `pmultivalue[rdf%3Atype]=1`	0, 1	*N/A*

Table 5.5: URI, keyword and named graph query parameters

Table 5.6 lists parameters that can be used with the metadata query.

For all queries, parameters and values are case-sensitive. Property names may be either full URIs, or prefixed names (e.g. `rdfs:label`). Available prefixes are managed in the administration frontend (see section 4.9).

For more information about aggregation settings, see the corresponding section of Conflict Resolution specification.

Name	Description	Possible values	Default value	Required
`uri`	URI of the requested named graph	*string*	*N/A*	yes
`format`	format of the result	`html`, `trig`, `rdfxml`	`html`	no

Table 5.6: Metadata query parameters

General aggregation methods

ALL	returns all conflicting values
BEST	value with the highest aggregated quality; in case of equality, the newest timestamp is preferred
LATEST	value with the newest timestamp; in case of equality, the highest aggregate quality is preferred
ANY	returns a single arbitrary value
CONCAT	concatenation of conflicting values separated by "`;` "
NONE	returns all conflicting values including duplicities

Numeric aggregation methods

MIN	minimum of conflicting values
MAX	maximum of conflicting values
AVG	average of conflicting values
MEDIAN	median of conflicting values

Date aggregation methods

MIN	the earliest date
MAX	the latest date

String aggregation methods

SHORTEST	the shortest string
LONGEST	the longest string

Error strategy

The error strategy determines how to handle values that cannot be aggregated by the given aggregation method, e.g. when applying MEDIAN aggregation to a mix of numeric and date values.

Note that for some aggregations, an untyped literal may be converted to a numeric literal (`xsd:double`) if possible.

Multivalue parameter

The multivalue parameter determines whether differences with other conflicting values decrease quality (`multivalue=0`), or not (`multivalue=1`). Setting multivalue to false (0) is appropriate for properties with a single value (e.g. `dbprop:population`), setting it to true (1) is appropriate for propertiees with multiple possible values (e.g. `rdf:type`).

5.3.3 Query Format

5.3.3.1 URI Query

The value of the `uri` parameter must be either a full valid URI, or a prefixed name (e.g. `dbpedia:Berlin`). Available prefixes are managed in the administration frontend (see section 4.9).

5.3.3.2 Keyword Query

The `kw` parameter can contain one or more keywords separated by whitespace. If a keyword itself contains spaces, it may be enclosed in double quotes. Query Execution looks for literals that contain all of the keywords. Keywords can also contain the `*` wildcard, but they must begin with at least four non-wildcard characters if a wildcard is to be used.

Query Execution also looks for an exact match of the entire `kw` value (i.e. without any division to keywords). If the `kw` value is a number, then numeric typed literals will also match; if the `kw` value is formatted as `xsd:dateTime`[5], then `xsd:dateTime` typed literals will also match.

Special characters, such as quotes and backslashes may be filterd out from searched keyword(s).

5.3.3.3 Named Graph Query

The value of the `uri` parameter must be either a valid URI, or a prefixed name, of an existing named graph.

5.3.3.4 Metadata Query

The value of the `uri` parameter must be a valid URI of an existing named graph.

5.3.4 Results Format for URI & Keyword Queries

The result contains triples returned in response to the query, including relevant labels of URI resources in the result, and metadata for the triples.

5.3.4.1 HTML

The result in HTML format contains results in a human-readable form (Figure 5.1). It contains

[5] `http://www.w3.org/TR/xmlschema-2/#dateTime-lexical-representation`

- a table with all triples in the result together with their aggregated quality and named graphs from which the triple was selected or calculated,
- a table with metadata of named graphs occuring in the first table.

URI query for <http://dbpedia.org/resource/Berlin>. Query executed in 1.569 s.

Subject	Predicate	Object	Quality	Source named graphs
dbpedia:Berlin	dbo:country	dbpedia:Germany	0.90000	http://odcs.mff.cuni.cz/namedGraph/qe-test/berlin/dbpedia
dbpedia:Berlin	http://linkedgeodata.org/property/capital	"yes"	0.80000	http://odcs.mff.cuni.cz/namedGraph/qe-test/berlin/linkedgeodata
dbpedia:Berlin	rdfs:label	"Berlin"	0.94252	http://odcs.mff.cuni.cz/namedGraph/qe-test/berlin/linkedgeodata, http://odcs.mff.cuni.cz/namedGraph/qe-test/berlin/geonames, http://odcs.mff.cuni.cz/namedGraph/qe-test/berlin/dbpedia, http://odcs.mff.cuni.cz/namedGraph/qe-test/berlin/freebase
dbpedia:Berlin	freebase:location.geocode.longtitude	"13.402740096987914"^^xsd:double	0.82446	http://odcs.mff.cuni.cz/namedGraph/qe-test/berlin/linkedgeodata, http://odcs.mff.cuni.cz/namedGraph/qe-test/berlin/dbpedia, http://odcs.mff.cuni.cz/namedGraph/qe-test/berlin/geonames, http://odcs.mff.cuni.cz/namedGraph/qe-test/berlin/freebase
dbpedia:Berlin	rdf:type	http://schema.org/City	0.92000	http://odcs.mff.cuni.cz/namedGraph/qe-test/berlin/dbpedia, http://odcs.mff.cuni.cz/namedGraph/qe-test/berlin/freebase
dbpedia:Berlin	rdf:type	http://schema.org/Place	0.90000	http://odcs.mff.cuni.cz/namedGraph/qe-test/berlin/dbpedia
dbpedia:Berlin	rdf:type	http://umbel.org/umbel/rc/Village	0.90000	http://odcs.mff.cuni.cz/namedGraph/qe-test/berlin/dbpedia
dbpedia:Berlin	rdf:type	http://www.geonames.org/ontology#Feature	0.80000	http://odcs.mff.cuni.cz/namedGraph/qe-test/berlin/geonames

Source graphs:

Named graph	Data source	Inserted at	Graph score	License	Update tag
http://odcs.mff.cuni.cz/namedGraph/qe-test/berlin/dbpedia	http://dbpedia.org/page/Berlin	2012-04-01 12:34:56.0	0.9		
http://odcs.mff.cuni.cz/namedGraph/qe-test/berlin/error	http://example.com		0.8		
http://odcs.mff.cuni.cz/namedGraph/qe-test/berlin/freebase	http://www.freebase.com/view/en/berlin	2012-04-02 12:34:56.0	0.8		
http://odcs.mff.cuni.cz/namedGraph/qe-test/berlin/geonames	http://www.geonames.org/2950159/berlin.html	2012-04-03 12:34:56.0	0.8		
http://odcs.mff.cuni.cz/namedGraph/qe-test/berlin/linkedgeodata	http://linkedgeodata.org/page/node240109189	2012-04-04 12:34:56.0	0.8		
http://odcs.mff.cuni.cz/namedGraph/qe-test/germany/dbpedia	http://dbpedia.org/page/Germany	2012-04-05 12:34:56.0	0.9		

Figure 5.1: Example of HTML output for URI query for `dbpedia:Berlin`

5.3.4.2 TriG

If the `format` parameter is set to `trig`, the result contains triples (quads) serialized in the TriG[6] format. The result includes:

- triples returned in response to the query, each one placed in a unique named graph
- aggregated quality (`odcs:quality`) and source named graphs (`odcs:sourceGraph`) of the above triples; subjects of these statements are the unique named graphs where the respective triples are placed
- metadata of source named graphs; they may include where the data were extracted from (`w3p:source`), Quality Assesment score of the named graph (`odcs:score`) and of its publisher (`odcs:publisherScore`), the publisher of the data (`w3p:publishedBy`), timestamp (`w3p:insertedAt`), license (`dc:license`), update tag (`odcs:updateTag`)

[6] http://www4.wiwiss.fu-berlin.de/bizer/trig/

- metadata about the query response itself – a title (`dc:title`), date (`dc:date`), number of result triples (`odcs:totalResults`), the query (`odcs:query`) and link to each result item (`odcs:result`)

An example:

```
@prefix :       <#> .
@prefix odcs:   <http://opendata.cz/infrastructure/odcleanstore/> .
@prefix w3p:    <http://purl.org/provenance#> .
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix dbpedia: <http://dbpedia.org/ontology/> .

<http://opendata.cz/infrastructure/odcleanstore/query/results/1> {
  <http://dbpedia.org/resource/Berlin> rdfs:label "Berlin"@en .
}

<http://opendata.cz/infrastructure/odcleanstore/query/results/2> {
  <http://dbpedia.org/resource/Berlin> dbpedia:populationTotal
    "3420768"^^<http://www.w3.org/2001/XMLSchema#int> .
}

<http://opendata.cz/infrastructure/odcleanstore/query/metadata/> {
  <http://opendata.cz/infrastructure/odcleanstore/query/results/1>
    odcs:quality 0.92 ;
    w3p:source <http://opendata.cz/infrastructure/odcleanstore/data/e0cdc9d7-e2d8-4bde> ;
    w3p:source <http://opendata.cz/infrastructure/odcleanstore/data/b68e21f7-363f-4bfd> .

  <http://opendata.cz/infrastructure/odcleanstore/query/results/2>
    odcs:quality 0.8966325468133597 ;
    w3p:source <http://opendata.cz/infrastructure/odcleanstore/data/b68e21f7-363f-4bfd> .

  <http://opendata.cz/infrastructure/odcleanstore/data/e0cdc9d7-e2d8-4bde>
    odcs:score 0.9 ;
    w3p:insertedAt "2012-04-01 12:34:56.0"^^<http://www.w3.org/2001/XMLSchema#dateTime> ;
    w3p:source <http://dbpedia.org/page/Berlin> ;
    w3p:publishedBy <http://dbpedia.org/> ;
    dcterms:license <http://creativecommons.org/licenses/by-sa/3.0/> ;
    odcs:publisherScore 0.9 ;
    odcs:updateTag "dataset123".


  <http://opendata.cz/infrastructure/odcleanstore/data/b68e21f7-363f-4bfd>
    odcs:score 0.8 ;
    w3p:insertedAt "2012-04-04 12:34:56.0"^^<http://www.w3.org/2001/XMLSchema#dateTime> ;
    w3p:source <http://linkedgeodata.org/page/node240109189> .

  <http://localhost:8087/uri?uri=http%3A%2F%2Fdbpedia.org%2Fresource%2FBerlin>
    a odcs:QueryResponse ;
    dc:title "URI search: http://dbpedia.org/resource/Berlin" ;
    dc:date "2012-08-01T10:20:30+01:00" ;
    odcs:totalResults 2 ;
```

```
    odcs:query "http://dbpedia.org/resource/Berlin" ;
    odcs:result <http://opendata.cz/infrastructure/odcleanstore/query/results/1> ;
    odcs:result <http://opendata.cz/infrastructure/odcleanstore/query/results/2> .
}
```

Listing 5.2: Example of URI or keyword query response in TriG

5.3.4.3 RDF/XML

If the `format` parameter is set to `rdfxml`, then the result will be formatted in RDF/XML.[7] The returned triples contain

- triples returned in response to the query,
- metadata about the query response itself

as in case of TriG output, however no metadata about quality of triples or about source named graphs are included.

5.3.4.4 Paging of results

As of now, all results are returned on a single page. The approximate maximum number of triples in the result is 500 by default (and can be set in the configuration file, see Administrator's & Installation Manual).

5.3.5 Results Format for Named Graph Query

Named graph query selects all triples stored in the given named graph and is intended mainly for debugging purposes. The format of results for the named graph query is exactly the same as for URI or keyword queries (see Section 5.3.4). The only difference is that labels for URI resources in the result are not retrieved (unless they are contained in the named graph). Also, conflict resolution considers only the named graph and not any other conflicting (or same) values that may be stored in other graphs.

5.3.6 Results Format for Metadata Query

The result contains metadata and Quality Assessment results for a given named graph. The metadata include metadata maintained by ODCleanStore (e.g. `odcs:insertedAt`) and data from the `provenance` metadata graph. Quality Assessment is executed on the named graph at query time, with rules that would be applied to it in its respective pipeline.

5.3.6.1 HTML

The result in HTML format contains results in a human-readable form (Figure 5.2). It contains

- a table with ODCleanStore metadata,

[7] http://www.w3.org/TR/REC-rdf-syntax/

- the results of Quality Assessment, i.e. the resulting score and all Quality Assessment rules the named graph violated and thus its score was decreased by the respective coefficient (only if there is at least one Quality Assessment rule group applicable to the named graph),
- `provenance` metadata, if available.

Metadata query for named graph <http://opendata.cz/infrastructure/odcleanstore/data/adfab24d-ef92-42d6-46c83dc4eb80>. Query executed in 1.413 s.

Basic metadata:

Named graph	Data source	Inserted at	Graph score	License	Update tag
odcs-data:adfab24d-ef92-42d6-9fb7-46c83dc4eb80	http://source.com/a, http://source.com/b	2012-10-20 10:11:46.0	0.45	http://creativecommons.org/licenses/by-sa/3.0/, http://creativecommons.org/licenses/by-sa/3.0/cz/	

Total Quality Assessment score: 0.45000

Quality Assessment rule violations:

Rule description	Score decreased by
Publication date after tender deadline.	0.90000
Invalid gr:hasCurrencyValue price.	0.50000

Additional provenance metadata:

Subject	Predicate	Object
http://example.com	http://purl.org/provenance#someProperty	"Some provenance information"

Figure 5.2: Example of HTML output for metadata query

5.3.6.2 TriG

The result contains triples (quads) serialized in the TriG format. Again, the result contains ODCleanStore metadata, additional `provenance` metadata, results of Quality Assessment and also metadata about the query response itself. The meaning of used predicates is as described in Section 5.3.4.2.

The `provenance` metadata are contained in one named graph and a triple <*payload-graph*>-`odcs:provenanceMetadataGraph`-<*provenance-graph*> points to it; all other data are placed in another named graph.

An example:

```
@prefix :       <#> .
@prefix odcs:   <http://opendata.cz/infrastructure/odcleanstore/> .
@prefix w3p:    <http://purl.org/provenance#> .
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dc:     <http://purl.org/dc/terms/> .

<http://opendata.cz/infrastructure/odcleanstore/query/metadata/> {
  <http://opendata.cz/infrastructure/odcleanstore/data/e0cdc9d7-e2d8-4bde>
    w3p:insertedAt "2012-04-01 12:34:56.0"^^<http://www.w3.org/2001/XMLSchema#dateTime> ;
    w3p:source <http://dbpedia.org/page/Berlin> ;
    dc:license <http://creativecommons.org/licenses/by-sa/3.0/> ;
    odcs:updateTag "dataset123" ;
    w3p:publishedBy <http://dbpedia.org/> ;
    odcs:provenanceMetadataGraph
      <http://opendata.cz/infrastructure/odcleanstore/provenanceMetadata/e0cdc9d7-e2d8-4bde>;
```

```
    odcs:score 0.72 ;
    odcs:violatedQARule <http://opendata.cz/infrastructure/odcleanstore/QARule/10> ;
    odcs:violatedQARule <http://opendata.cz/infrastructure/odcleanstore/QARule/20> .

  <http://opendata.cz/infrastructure/odcleanstore/QARule/10>
    a odcs:QARule ;
    odcs:coefficient 0.8 ;
    dc:description "Procedure type ambiguous" .

  <http://opendata.cz/infrastructure/odcleanstore/QARule/20>
    a odcs:QARule ;
    odcs:coefficient 0.9 ;
    dc:description "Procurement contact person missing" .

  <http://localhost:8087/namedGraph?uri=http%3A%2F%2Fopendata.cz
      %2Finfrastructure%2Fodcleanstore%2Fdata%2Fe0cdc9d7-e2d8-4bde>
    a odcs:QueryResponse ;
    dc:title "Metadata for named graph:
      http://opendata.cz/infrastructure/odcleanstore/data/e0cdc9d7-e2d8-4bde" ;
    dc:date "2012-08-01T10:20:30+01:00" ;
    odcs:query "http://opendata.cz/infrastructure/odcleanstore/data/e0cdc9d7-e2d8-4bde";
}

<http://opendata.cz/infrastructure/odcleanstore/provenanceMetadata/e0cdc9d7-e2d8-4bde> {
  <http://opendata.cz/infrastructure/odcleanstore/data/e0cdc9d7-e2d8-4bde>
    w3p:provenanceMetadataProperty1 "provenanceMetadataValue1".
  <http://opendata.cz/infrastructure/odcleanstore/data/e0cdc9d7-e2d8-4bde>
    w3p:provenanceMetadataProperty2 "provenanceMetadataValue2".
}
```

Listing 5.3: Example of metadata query response in TriG

5.3.6.3 RDF/XML

The result for a metadata query serialized in RDF/XML contains the same triples as in case of TriG (Section 5.3.6.2) except that triples are not divided into named graphs.

5.3.6.4 Paging of results

See Section 5.3.4.4.

5.3.7 Quality Calculation

As stated in the previous sections, an aggregate quality estimate for each triple in the result is part of the query results. The quality is expressed as a number from interval [0,1] where 0 means lowest quality and 1 highest quality.

The aggregate quality estimate is done for each result quad and is based on several factors based on real-world scenarios. The factors include quality scores of the source named graphs (as calculated by the Quality Assessor transformer, Section 4.3.1), number of graphs that agree on a value, and the difference between a value and other (conflicting) values.

Since conflicts in data are resolved by aggregating values in place of objects of resolved triples, the quality also depends on object values (and not the subject or predicate of a triple). The exact calculation depends on the aggregation method used and other aggregation settings given to Output Webservice. In short, the calculation for a value in place of an object can be outlined as:

- Quality Assessment scores of graphs that the value was selected or calculated from are taken. Depending on the aggregation method, their average or maximum is used as the initial score.
- Differences with other conflicting values are taken into consideration. The more conflicting values differ, the more the quality is decreased. This step can be turned off by setting the `multivalue` parameter to false (`0`).
- If there are multiple sources that agree on exactly the same value, the quality is increased.

This is a very crude description of the algorithm. You can find it explained in detail in section "Quality and Provenance Calculation" of Programmer's Guide.

6. Stored Data

6.1 Input Processing

When a new request is sent to Input Webservice, the stored data & metadata go through several phases.

1. First, data & metadata are validated. `Payload` data and optional `provenance` metadata (see Section 5.2.1 Request parameters) should be valid RDF/XML or TTL, all required metadata fields must have the proper cardinality and valid format. An exception is thrown and the request interrupted if validation fails.
2. If all data are valid, the request is queued, Input Webservice indicates success and the transmission successfully finishes.
3. Engine, independently on Input Webservice, successively takes requests from the input queue and processes them. RDF data from `payload` are stored to a single named graph, `provenance` metadata to a separate named graph and other metadata to another separate named graph called *metadata graph*, all in the dirty (staging) database. The format of RDF triples in the metadata graph is described in Section 6.2.
4. Because some predicates are reserved for purposes of internal metadata representation in ODCleanStore, RDF triples that contain these predicates are removed from `payload` and `provenance` named graphs. Table 6.1 lists all reserved predicates.

`odcs:score`
`odcs:publisherScore`
`odcs:scoreTrace`
`odcs:metadataGraph`
`odcs:provenanceMetadataGraph`
`odcs:sourceGraph`
`odcs:insertedAt`
`odcs:insertedBy`
`odcs:source`
`odcs:publishedBy`
`odcs:license`
`odcs:updateTag`
`odcs:isLatestUpdate`

Table 6.1: Reserved RDF predicates

5. Next, the processing pipeline is selected – if `pipelineName` was present, pipeline with the given name is used, the default pipeline is used otherwise. Engine runs each transformer in the pipeline on the stored data. Transformers can modify the inserted named graphs or attach new named graphs (*attached named graph*). See Administrator's & Installation Manual for more information about transformers.
6. Engine runs a special (automatically added) transformer, that checks if the currently processed data are an update of data already stored in the clean database. More

specifically, an inserted named graph A is considered an update of named graph B if and only if the following conditions hold:

(i) Named graphs A and B have the same update tag, or both have an unspecified (`null`) update tag.
(ii) Named graphs A and B were inserted by the same (SCR) user.
(iii) Named graphs A and B have the same set of sources in metadata.
(iv) Named graph A was inserted later than named graph B.

The `payload` named graph is marked as the latest version by adding a triple with predicate `odcs:isLatestUpdate` to the `metadata` graph. If the currently processed data update a named graph already stored in the clean database, this triple is removed for the older graph.

7. If all transformers in the pipeline finish successfully, the `payload` graph, `provenance` graph, metadata graph and any new attached graphs are moved from the dirty database to the clean database, while the respective request is removed from the queue.

6.2 Stored Data Structure

Data originating from a single request to Input Webservice can be stored in several named graphs. RDF data given in the `payload` parameter are stored in one named graph (`payload` graph). If `provenance` RDF metadata are given, they are stored in another named graph (`provenance` graph). Other metadata (such as the source of data, timestamp, etc.) are stored in yet another named graph (*metadata graph*). In addition, transformers in the respective pipeline may add more related RDF data to one or more named graphs (*attached graphs*), e.g. results of quality assessment, or mappings for resources in `payload`.

While contents of the `payload`, `provenance` and attached graphs may be arbitrary, the metadata graph has a set structure. Table 6.2 describes the structure of a metadata graph. In the table, <*payload-graph*> stands for the name of the respective `payload` graph, <*provenance-graph*> and <*metadata-graph*> analogously.

Note that transformers may add triples to the metadata graph too. For example, Quality Assessment adds these two triples:

- <*payload-graph*> – `odcs:score` – <*QA-score*>
- <*payload-graph*> – `odcs:scoreTrace` – <*QA-score-explanation*>

Subject	Predicate	Object	Cardinality
<*payload-graph*>	`odcs:metadataGraph`	<*metadata-graph*>	1
<*payload-graph*>	`odcs:provenanceMetadataGraph`	<*provenance-graph*>	0..1
<*payload-graph*>	`odcs:attachedGraph`	URIs of attached graphs	0..*
<*payload-graph*>	`odcs:insertedAt`	insertion time	1
<*payload-graph*>	`odcs:insertedBy`	name of the user who inserted the data	1
<*payload-graph*>	`odcs:source`	source of the data (values from the `source` field)	1..*
<*payload-graph*>	`odcs:publishedBy`	identifier of the publisher of the data (values from the `publishedBy` field)	1..*
<*payload-graph*>	`odcs:license`	license of the data (values from the `license` field)	0..*
<*payload-graph*>	`odcs:updateTag`	distinguisher of graph updates (value from the `updateTag` field)	0..1
<*payload-graph*>	`odcs:isLatestUpdate`	1 *Present only for the latest version of data (see Section 6.1, step 6)*	0..1

Table 6.2: RDF triples in a `metadata` graph

6.3 Executing Pipelines on the Clean Database

The pipeline creator or administrator can decide to re-run a transformer pipeline on one or more named graphs that are already in the clean database, e.g. when the respective transformer rules changed. In that case, such named graphs are queued for processing and Engine successively runs the pipeline on each queued graph:

1. First, a copy of the `payload`, `provenance`, metadata and any attached graphs is created in the dirty database.
2. The same processing pipeline that was used when the data came through Input Webservice is run on this copy. Transformers can modify any of the graphs and attach new graphs.
3. In a transaction, the old version in the clean database is deleted and the processed copy (together with any new attached graphs) is moved from the dirty database to the clean database.

A. Glossary

RDF-related

RDF
: Resource Description Framework, a language for representing information about resources in the World Wide Web[1]

RDF triple
: Statement about a resource expressed in the form of subject-predicate-object expression

URI
: Uniform Resource Identifier, identifies RDF resources

Named graph
: A set of related RDF triples (RDF graph) named with a URI[2]

RDF quad
: An RDF triple plus named graph URI (subject, predicate, object, named graph)

Ontology
: Representation of the meaning of terms in a vocabulary and of their interrelationships

OWL
: The Web Ontology Language[3]

SPARQL
: RDF query language[4]

RDF/XML
: An XML-based serialization format for RDF graphs[5]

TTL
: Turtle – Terse RDF Triple Language[6]; a human-friendly alternative to RDF/XML

Data & Data Quality

Dirty (staging) database
: Database where incoming data are stored until they are processed by a processing pipeline (e.g. clean, linked to other data, etc.)

[1] http://www.w3.org/RDF/
[2] http://www.w3.org/2004/03/trix/
[3] http://www.w3.org/TR/owl-features/
[4] http://www.w3.org/TR/rdf-sparql-query/
[5] http://www.w3.org/TR/rdf-syntax-grammar/
[6] http://www.w3.org/TeamSubmission/turtle/

Clean database
Database where incoming data are stored after they are successfully processed by the respective processing pipeline; this database can be accessed using the Output Webservice

`Payload` **graph**
Named graph where the actual inserted data, given in the `payload` parameter of Input Webservice, are stored

`Provenance` **graph**
Named graph where additional provenance metadata, given in the `provenance` field of Input Web Service, are stored

Metadata graph
Named graph where other metadata about a `payload` graph (such as source, timestamp, license, etc.) are stored

Attached graph
Named graph attached to a `payload` graph by a transformer

Named graph score
Quality of a single (`payload`) named graph estimated by the Quality Assesment component and stored in the database, expressed as a number from interval [0,1]

Publisher score
Average score of named graphs from a publisher

Aggregate quality
Quality of a triple in the results calculated by the Conflict Resolution component during query time, expressed as a number from interval [0,1]

Data Processing

Pipeline
A configurable sequence of transformers that is used to process a named graph. The pipeline to process data sent to Input Webservice can be selected explicitly, or the default pipeline is used.

Transformer
A Java class which implements the *Transformer* interface that and is registered in ODCleanStore Administration Frontend by an administrator.

Transformer instance (or transformer assignment)
Assignment of a *transformer* to a *pipeline*. A single transformer can be assigned to multiple pipelines (or even to a single pipeline multiple times), thus creating multiple transformer instances.

Rule

Some transformers included in ODCleanStore can be configured in Administration Frontend by rules. Rules are grouped together to *rule groups*.

Rule group

A group of transformer *rules*. Rule groups can be assigned to transformer instances.

User Roles

ADM

Administrator

ONC

Ontology creator

PIC

Pipeline creator

SCR

Data producer (scraper)

USR

Data consumer

B. List of Used XML Namespaces

Prefix	**URI**
odcs	http://opendata.cz/infrastructure/odcleanstore/
w3p	http://purl.org/provenance#
dc	http://purl.org/dc/terms/
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs	http://www.w3.org/2000/01/rdf-schema#
owl	http://www.w3.org/2002/07/owl#
xsd	http://www.w3.org/2001/XMLSchema#
dbpedia	http://dbpedia.org/resource/
dbprop	http://dbpedia.org/property/
skos	http://www.w3.org/2004/02/skos/core#

Table B.1: List of used XML namespaces