voiD guide

Using the Vocabulary of Interlinked Datasets

29 January 2009

This version:
$Id: index.html 61 2009-01-29 13:42:53Z michau $
Latest version:
http://rdfs.org/ns/void-guide
Authors:
K. Alexander (Talis)
R. Cyganiak (DERI)
M. Hausenblas (DERI)
J. Zhao (University of Oxford)

Abstract

voiD is a vocabulary and a set of instructions that enables the discovery and usage of linked datasets. A dataset is a collection of data, published and maintained by a single provider, available as RDF, and accessible, for example, through dereferenceable HTTP URIs or a SPARQL endpoint. Based on the voiD vocabulary this document explains how to use voiD in a practical setup, for both data consumers and data providers.

Table of Contents

Scope

This document, the voiD guide, is one of the two core resources of voiD; the other one is the normative **voiD vocabulary** document [voiD vocabulary]. The voiD guide aims at dataset publishers and maintainers as well as administrators (collectively known as data publishers), and on the other hand developers, that is, people interested in using linked datasets in order to for example build applications on top of it (data users). Further, parties that operate an indexing or crawling service are also in scope of our target users. For reading this document, we assume that people are familiar with basic concepts about Web of Data, such as URIs and RDF, and have some basic knowledge about certain widely-used vocabularies such as Dublin Core (DC) or Friend of a Friend (FOAF).

Introduction

The Web of Data (WoD) aims at providing technologies to support tasks that are tedious, time-consuming or otherwise non-human-friendly. Two technologies form the base of the WoD, namely URIs and RDF. URIs enable the worldwide, universal addressing of resources (such as people, documents, etc.); RDF provides a common data model that defines how to express properties and relations between resources. These resources could be published as a collection of data, i.e., a dataset, by anyone, in a distributed manner onto the Web. Such datasets already form a huge pool of real-world data available on the WoD. Still, it is an open issue on how to effectively and efficiently find the right data for the right task [WOD Discovery].

The Vocabulary of Interlinked Datasets (voiD) therefore aims to provide a vocabulary to bridge data publishers and data users, so that users can find the right data for their tasks more easily using the voiD description about a linked dataset. With discovery of datasets we mean the identification of datasets given certain attributes, trying to answer the question: *given a set of attributes, which available resources match the desired set and what is their location?*

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to

be interpreted as described in [RFC2119]. All examples in this document are informative, and are not meant to be interpreted as normative requirements.

*Note: Feedback on this document should be sent to **void-rdfs-internals[AT]googlegroups.com**. We also try to be available via IRC, on #swig channel at irc.freenode.net. You may further want to subscribe to the !voiD group on identi.ca in order to receive latest news and/or to discuss voiD.*

1. Describing Datasets

The basic functionality that voiD offers is describing a dataset. A **dataset** in voiD (`void:Dataset`) is a collection of data, which is:

- published and maintained by a **single provider**, and
- available as **RDF**, and
- **accessible**, for example, through dereferenceable HTTP URIs or a SPARQL endpoint.

Note: A voiD dataset `void:Dataset` is a subclass of a DCMI dataset (`dcmi:Dataset`). Consult the DCMI type vocabulary [DCMI type voc] for further information.

In the following we describe the most common cases for using voiD along with reusing terms from other, widely deployed vocabularies such as Dublin Core or FOAF. Throughout the document, the following namespaces are used:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix scovo: <http://purl.org/NET/scovo#> .
@prefix void: <http://rdfs.org/ns/void#> .
```

All examples in this document are written in the Turtle RDF syntax [Turtle]. To keep the examples simple and consistent, we will assume that we are creating a file called `void.ttl` which will contain a voiD description of the dataset under discussion in the example. We will further assume that the empty prefix is bound to the file like this:

```
@prefix : <#> .
```

This allows us to quickly mint new identifiers in the local namespace: `:MyDataset`, `:DBpedia` and so on. For further information about publishing and deploying voiD descriptions see section 4. Publishing voiD descriptions of datasets.

1.1 General Dataset Metadata

Let us assume that we are describing a well-known linked dataset, DBpedia. We will first mint an identifier, and type it as a `void:Dataset`:

```
:DBpedia a void:Dataset .
```

Next, we will add some basic information about the dataset: its **name**, a short **description**, its **homepage**, and links to some **example resources** that give a quick impression of the kind of data available in the dataset. This can be considered the most basic information that should be provided in any voiD description.

```
:DBpedia a void:Dataset ;
         dcterms:title "DBPedia" ;
         dcterms:description "RDF data extracted from Wikipedia" ;
         foaf:homepage <http://dbpedia.org/> ;
         void:exampleResource <http://dbpedia.org/resource/Berlin> ;
         void:exampleResource <http://dbpedia.org/resource/Physics> ;
         void:exampleResource <http://dbpedia.org/resource/Ludwig_van_Bee
```

Note: As `foaf:homepage` is an Inverse Functional Property (IFP), providing the homepage URI allows one to connect different descriptions of DBpedia that are provided in different places on the Web.

Additional metadata can be provided. [Table 1] below lists some RDF properties that are especially useful and recommended, many from the Dublin Core Metadata Terms. Other properties can also be used, if not listed here.

`foaf:homepage`	The homepage of the dataset. Note, this must be different from the homepage of the creator or publisher to avoid incorrect 'smushing'. If the dataset has no homepage dedicated to it, then `foaf:page` can be used instead.
`void:exampleResource`	Link to some representative entity described within the dataset. This is intended to give users a quick impression of the kind of data contained in the dataset.
`void:uriRegexPattern`	A regular expression pattern that matches one or more URI in the dataset. The pattern should use the same regular expression syntax as SPARQL, which uses the syntax definition of XML Schema 2: Regular Expressions.
`dcterms:title`	The name of the dataset.
`dcterms:description`	A textual description of the dataset.
`dcterms:creator`	An entity, such as a person, organisation, or service, that is primarily responsible for creating the dataset. The creator should be described as an RDF resource, rather than just providing the name as a literal.
`dcterms:publisher`	An entity, such as a person, organisation, or service, that is responsible for making the dataset available. The publisher should be described as an RDF resource, rather than just providing the name as a literal.
`dcterms:contributor`	An entity, such as a person, organisation, or service, that is responsible for making contributions to the dataset. The contributor should be described as an RDF resource, rather than just providing the name as a literal.
`dcterms:source`	A related resource from which the dataset is derived. The source should be described as an RDF resource, rather than as a literal.
`dcterms:date`	A point or period of time associated with an event in the life-cycle of the resource. The value should be formatted and data-typed as an `xsd:date`.
`dcterms:created`	Date of creation of the dataset. The value should be formatted and data-typed as an `xsd:date`.
`dcterms:issued`	Date of formal issuance (e.g., publication) of the dataset. The value should be formatted and datatyped as an `xsd:date`.
`dcterms:modified`	Date on which the dataset was changed. The value should be formatted and datatyped as an `xsd:date`.

Table 1.: Using Dublin Core Metadata Terms in voiD for general dataset metadata.

The next example shows a more complete description of DBpedia that uses many of the properties above:

```
:DBpedia a void:Dataset ;
         dcterms:title "DBPedia" ;
         dcterms:description "RDF data extracted from Wikipedia" ;
         foaf:homepage <http://dbpedia.org/> ;
         void:exampleResource <http://dbpedia.org/resource/Berlin> ;
         void:exampleResource <http://dbpedia.org/resource/Physics> ;
         void:exampleResource <http://dbpedia.org/resource/Ludwig_van_Bee
         void:uriRegexPattern "http://dbpedia.org/resource/.+" ;
         dcterms:contributor :FU_Berlin ;
         dcterms:contributor :University_Leipzig ;
         dcterms:contributor :OpenLink_Software ;
         dcterms:contributor :DBpedia_community ;
         dcterms:source <http://dbpedia.org/resource/Wikipedia> ;
         dcterms:modified "2008-11-17"^^xsd:date .

:FU_Berlin a foaf:Organization ;
           rdfs:label "Freie Universität Berlin" ;
           foaf:homepage <http://www.fu-berlin.de/> .

 # Similar descriptions of the other contributors go here
```

1.2 Categorise Datasets

When someone wants to select a dataset, one of the fundamental questions is, what does the dataset actually offer? There are datasets such as DBpedia that cover quite a range of topics, whereas there are others that only talk about a certain domain (books, places, etc.).

In voiD, the `dcterms:subject` property should be used to categorise or describe the topic covered by datasets.

1.2.1 General Case

For the general case, we recommend the use of a DBpedia resource URI (`http://dbpedia.org/resource/XXX`) to categorise a dataset, where XXX stands for the thing which best describes the main topic of what the dataset is about.

Two examples are given below. DBLP is a computer science bibliography database, and Geonames offers data about places. We define this in voiD:

```
:DBLP a void:Dataset;
      dcterms:subject <http://dbpedia.org/resource/Computer_science> ;
      dcterms:subject <http://dbpedia.org/resource/Journal> ;
      dcterms:subject <http://dbpedia.org/resource/Proceedings> .

:Geonames a void:Dataset;
          dcterms:subject <http://dbpedia.org/resource/Location> .
```

In the case of DBLP, we add triples saying that it offers data about proceedings, journals and computer science. Note: It is assumed that the intersection of all resources defines the topic.

1.2.2 Domain specific

DBpedia might not contain the concepts for describing some domain specific datasets. For example, there are no exact DBpedia resource URIs for describing that a dataset is about "*in situ* hybridisation images", or that another one is about "UniProt Genes". We encourage publishers to describe such datasets using concepts widely adopted in their own communities, so that they can not only capture precisely the categorisation of their datasets but also ensure that these datasets could be connected with other relevant data from their domains. For example, we could define that:

```
:Bio2RDF a void:Dataset ;
         foaf:homepage <http://bio2rdf.org/> ;
         dcterms:subject <http://purl.uniprot.org/core/Gene> .
```

We also encourage data publishers to use SKOS Concepts whenever possible to capture the topics of their datasets.

1.3 Announcing the license of a dataset

The `dcterms:license` property SHOULD be used to to point to the license under which a dataset has been published. This is crucial because without a license, potential users of the dataset do not know under which terms they can use it.

To allow automatic analysis of datasets, it is important to use canonical identifiers for well-known licenses. The list below provides canonical URIs for some licenses that have been used to publish RDF datasets.

- Creative Commons Public Domain Dedication, http://creativecommons.org/licenses/publicdomain/
- Other Creative Commons licenses, http://creativecommons.org/licenses/
- GNU Free Documentation License, http://www.gnu.org/copyleft/fdl.html
- Open Data Commons Public Domain Dedication and License, http://www.opendatacommons.org/odc-public-domain-dedication-and-licence/

The example below states that the DBpedia dataset is published under the terms of the GNU Free Documentation License.

```
:DBpedia a void:Dataset ;
         dcterms:license <http://www.gnu.org/copyleft/fdl.html> .
```

Note: Licensing of datasets is a complex issue. Datasets are collections of facts rather than creative works, and different laws apply. Most licenses such as Creative Commons or the GPL are based on copyright and are designed to protect creative works, but not databases, and applying them to datasets might not have the desired legal result. Meanwhile, efforts such as Open Data Commons, Science Commons and Health Commons are developing dedicated licenses for data, and the OpenDOAR project provides tools allowing open-access digital repositories to create customised policies.

1.4 Describing partitioned datasets

Sometimes data publishers partition their datasets into physical or logical subsets in order to obtain better performance or to reflect different provenance of dataset parts. A third-party may also only harvest or re-publish a subset of the original dataset, e.g. only the gene synonym information from a large genomic database, which is sufficient for its application. Using `void:subset` to describe a dataset's relationship with another dataset can create an awareness about the origin of this dataset and that it is a subset of another one.

```
:genomicDS void:subset :genenameDS .  # :genenameDS contains only the ger

:bio2rdf void:subset :proteinDS .  # :proteinDS contains only the proteir
```

1.5 Technical Description (Features)

To express a certain technical feature of a dataset, such as formats in which the data is available, there are two ways to realise this. In the simple case use `void:feature` along with a blank node. If one, for example, wants to say “this dataset is available as RDF/XML”, simply using the MIME type solves this:

```
:DBpedia void:feature [ dcterms:format "application/rdf+xml" ; ] .
```

However, there may be use cases where you want to talk about the feature in greater detail. In this case, create a new instance for a feature using `void:TechnicalFeature` and connect it to the dataset using `void:feature`:

```
:RDFXML a void:TechnicalFeature;
        rdfs:label "RDF/XML" ;
        rdfs:comment "Available as linked data in RDF/XML format" ;
        dcterms:format "application/rdf+xml" .

:DBpedia void:feature :RDFXML .
```

Note: voiD does not include a collection of pre-defined `void:TechnicalFeature` instances for particular technologies. Therefore, to describe technical features, it is necessary to define your own `void:TechnicalFeature` instances as in the examples above. This is perhaps most useful for larger collections of voiD descriptions, where feature instances can be re-used across multiple voiD descriptions.

1.6 SPARQL endpoints, URI lookup, and RDF data dumps

There are many different ways in which clients can access the actual data that is contained in a dataset. If the entities described in the dataset are identified by HTTP URIs, then it is a reasonable assumption that resolving such a URI will return an RDF description of the resource.

A SPARQL endpoint that provides access to a dataset via the SPARQL protocol can be announced using `void:sparqlEndpoint`:

```
:DBpedia void:sparqlEndpoint <http://dbpedia.org/sparql> .
```

Note: It is assumed that the default graph of the SPARQL endpoint is the dataset itself.

Besides the SPARQL protocol, a simple URI lookup protocol for accessing a dataset is supported by voiD:

1. Take the URI of a resource **R** that is described in the dataset
2. Urlencode the URI, and append it to the dataset's **URI lookup endpoint**
3. Perform an HTTP GET request on the resulting concatenated URI
4. The HTTP response is expected to be an RDF description of **R**.

Note: The HTTP request should be performed with an HTTP Accept header that indicates the formats supported by the requesting client, e.g. `"Accept: application/rdf+xml"` for a client that only supports RDF/XML.

The following example shows how the Sindice API could be described as a voiD dataset with a URI lookup endpoint:

```
:Sindice void:uriLookupEndpoint <http://api.sindice.com/v2/search?qt=terr
```

If an RDF dump of the dataset is available, then its location can be announced using `void:dataDump`:

```
:GenenameDS void:dataDump <http://openflydata.org/flyted/dump/> .
```

Several values of this property can provided if the dataset is split into multiple dumps.

We do not prescribe the format of such data dumps, but clients should expect dumps to be in any of the formats that are supported by the Semantic Sitemaps specification.

1.7 Vocabularies used in a dataset

Every RDF dataset uses one or more RDFS vocabularies or OWL ontologies. The vocabulary provides the terms (classes and properties) for expressing the data. The `void:vocabulary` property can be used to list vocabularies used in a dataset:

```
:LiveJournal a void:Dataset;
             void:vocabulary <http://xmlns.com/foaf/0.1/> .
```

This dataset uses the FOAF vocabulary.

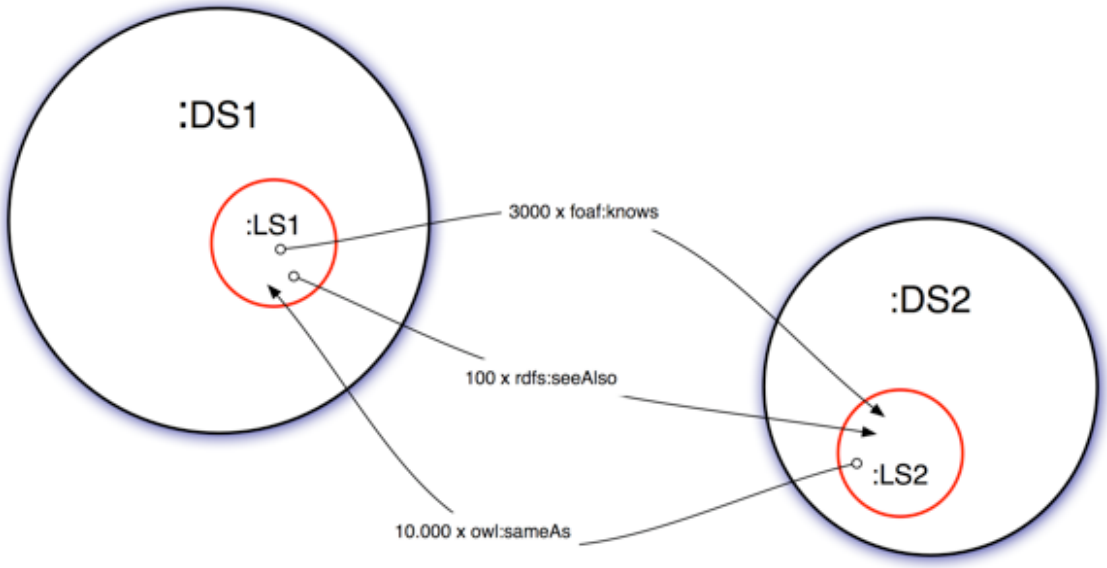
The vocabulary should be identified with the URI that's used with `rdfs:isDefinedBy` in the RDFS/OWL specification of the vocabulary. *Note: It is not necessary to list all vocabularies. Typically, only the most important vocabularies will be listed, especially those that can be useful in querying the dataset.*

To express that particular classes and properties occur in the dataset, statistics (see section 3. Expressing dataset statistics) can be used.

2. Describing Dataset Interlinking

Now, as we are able to describe the basic characteristics of datasets we want to express how datasets are interlinked. In voiD, the interlinking is modelled using a class, `void:Linkset`. This means, the interlinking in voiD is a first-class citizen. This modelling enables a flexible and powerful way to talk in great detail about the interlinking between two datasets, such as how many links there exist, which kind of links (e.g. `owl:sameAs` or `foaf:knows`) are present, or stating who claims these statements.

A **linkset** in voiD (`void:Linkset`) is a subset of a voiD dataset (`void:Dataset`), used for storing triples to express the interlinking relationship between datasets. In each interlinking triple, the subject is a resource hosted in one dataset and the object is a resource hosted in another dataset. The following figure depicts this:



In voiD we differentiate between the following orthogonal dimensions when talking about the interlinking:

- classic LOD vs. 3rd-party, which differs in terms of where the interlinking statements are kept. In the first case the interlinking triples, i.e. a linkset, are hosted in one of the two involved datasets, while in the latter case there is a third dataset involved that contains the interlinking triples, i.e. the linkset;
- non-directed vs. directed, which addresses the issue if someone is interested in stating the direction of the interlinking or not (for example with `owl:sameAs`)

We basically use three properties for interlinking descriptions:

- `void:subset` to signal where the triples are physically hosted (read: a dataset `:DS` has a linkset `:LS`);
- `void:target` to declare an interlinking target;
- `void:linkPredicate` to talk about the type (RDF predicate type) of the interlinking.

In case of a directed interlinking, voiD provides two properties, `void:subjectsTarget` and `void:objectsTarget`, which are sub-properties of `void:target`.

The following [Table 2] renders the possibilities along the two orthogonal dimensions:

3rd-party	`:DSC void:subset :LS .`
	`:LS void:target :DSA ;`
	`  void:target :DSB .`
		`:DSC void:subset :LS .`
		`:LS void:subjectsTarget :DSA ;`
		`  void:objectsTarget :DSB .`

classic LOD	`:DSX void:subset :LS .`
	`:LS void:target :DSX ;`
	`  void:target :DSY .`
		`:DSX void:subset :LS .`
		`:LS void:subjectsTarget :DSX ;`
		`  void:objectsTarget :DSY .`

	non-directed	**directed**

Table 2.: Interlinking description capabilities of voiD.

In the following we will show the different possibilities in detail. For example queries (that is how to find certain datasets) see section 5.4 Example Queries.

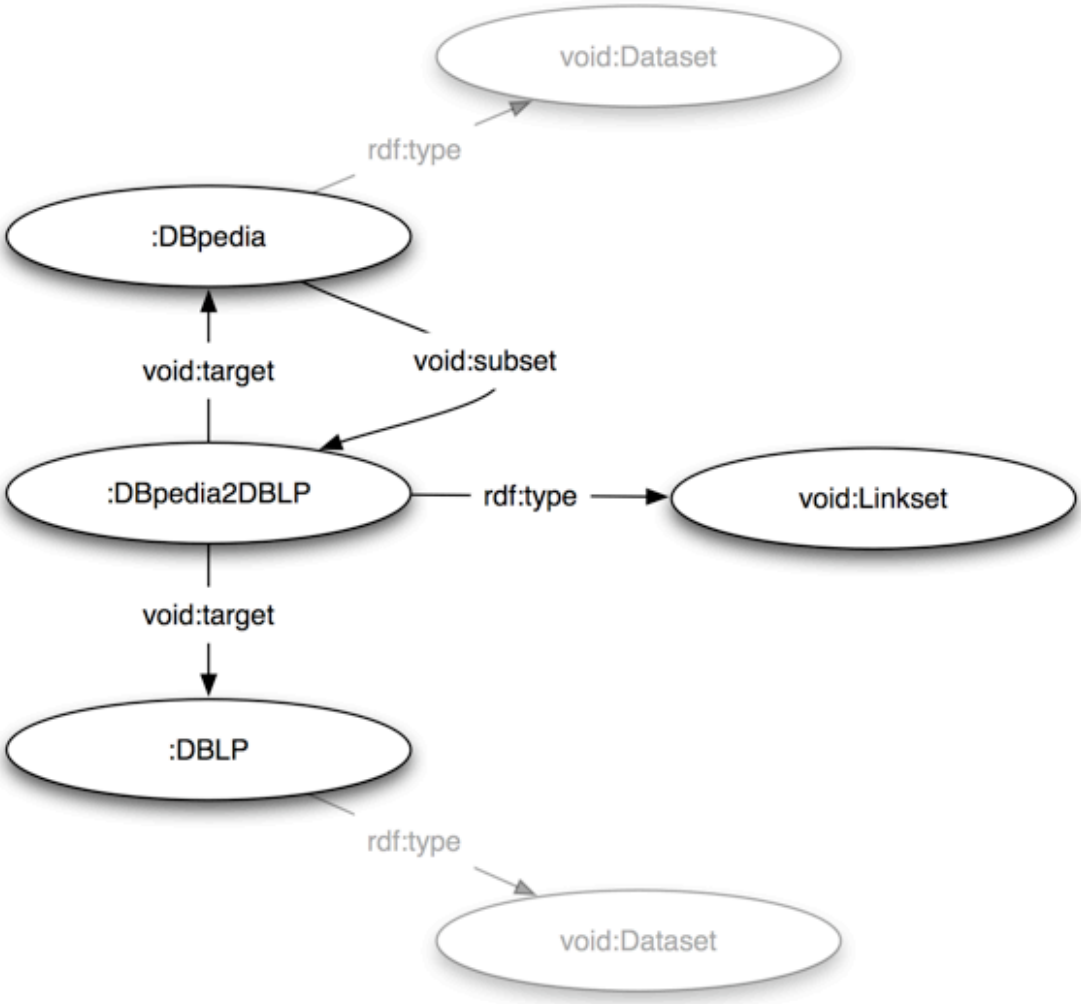
2.1 Classic LOD vs. 3rd-party Interlinking

The simplest interlinking case (the **classic LOD** case), found in many of the current LOD clouds, is, that two datasets are interlinked and one of the datasets actually hosts the interlinking triples. Let us assume we want to model the interlinking we find in DBpedia regarding to its links to DBLP in voiD:

```
:DBpedia void:subset :DBpedia2DBLP .

:DBpedia2DBLP rdf:type void:Linkset ;
              void:target :DBpedia ;
              void:target :DBLP .
```

The above example basically states that DBpedia and DBLP are interlinked and that these links are contained in the DBpedia dataset. The according RDF graph would then render as follows:

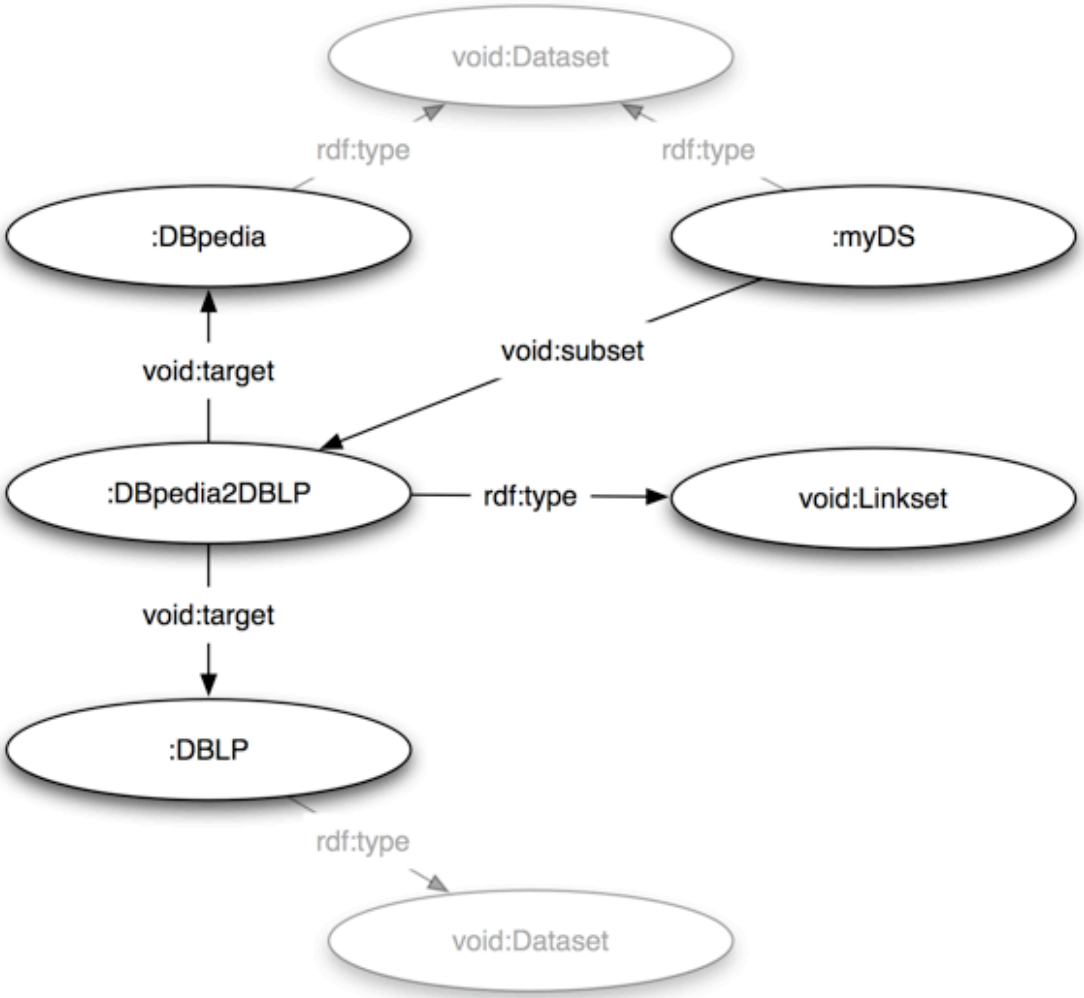


Further, there may be cases where a third dataset contains the data that talks about the interlinking between two datasets (**3rd-party** case). Let us assume a provider (`:myDS`) keeping track of the interlinking between DBpedia and DBLP (maybe offering some more detailed information, such as statistics, etc.) wants to express this using voiD:

```
:myDS void:subset :DBpedia2DBLP .

:DBpedia2DBLP rdf:type void:Linkset ;
              void:target :DBpedia ;
              void:target :DBLP .
```

The above example states that DBpedia and DBLP are interlinked, and that these links are contained in the `:myDS` dataset. `:myDS` could also contain interlinking information about many other datasets. The RDF graph for this example is:



2.2 Non-directed vs. Directed

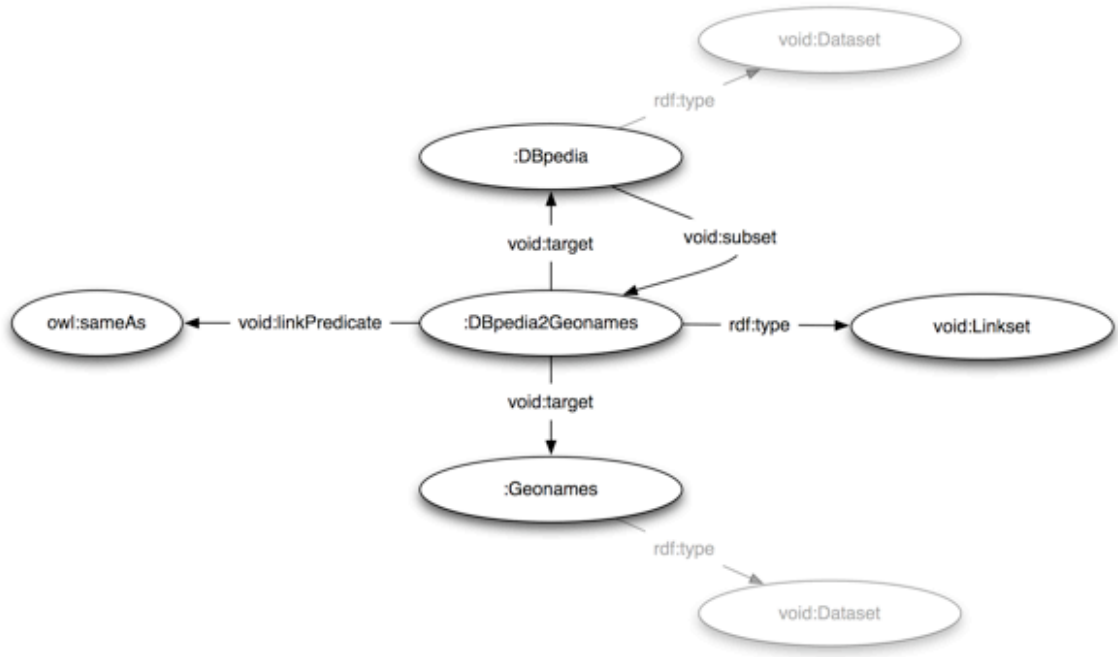
Depending on the type of interlinking, one may be interested in stating the direction of the interlinking.

In the **non-directed** case (i.e., the interlinking predicate is symmetric, such as `owl:sameAs`) you can simply use `void:target` (possibly along with `void:linkPredicate` to further specify the interlinking):

```
:DBpedia void:subset :DBpedia2Geonames .

:DBpedia2Geonames a void:Linkset ;
                  void:linkPredicate owl:sameAs ;
                  void:target :DBpedia ;
                  void:target :Geonames .
```

The voiD snippet above basically states that there are `owl:sameAs` links between DBpedia and Geonames and that these links are contained in the DBpedia dataset, which can also be seen in the RDF graph below:

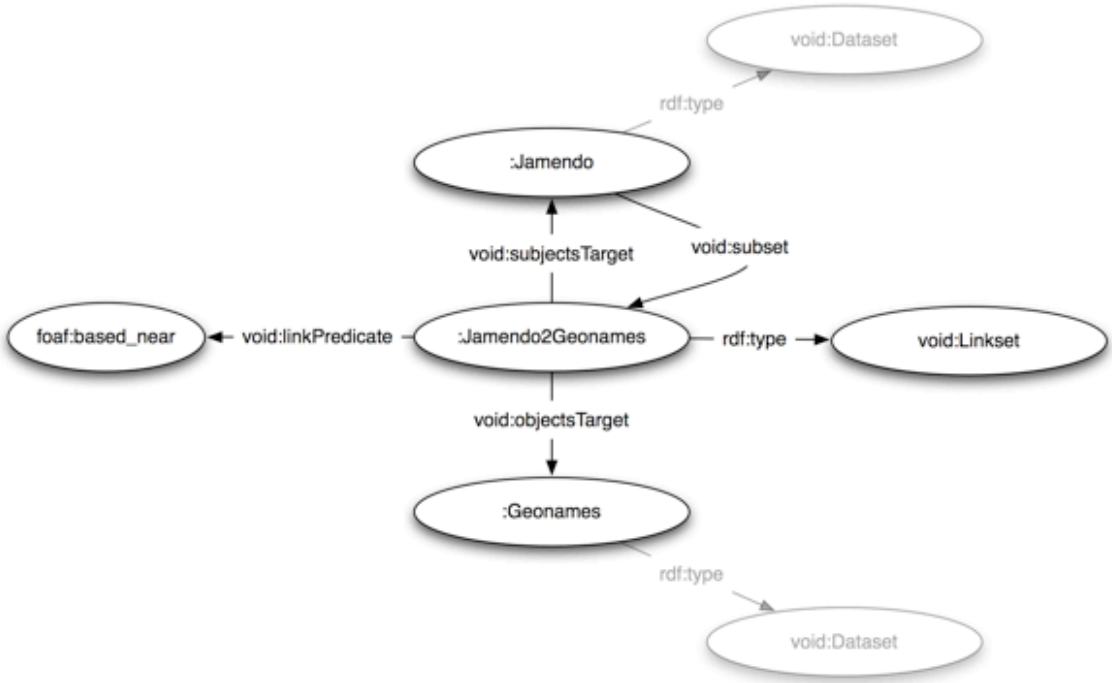


In the **directed** case (for example with `foaf:based_near`) you would use `void:subjectsTarget` and `void:objectsTarget` along with `void:linkPredicate` to state the direction of the interlinking:

```
:Jamendo void:subset :Jamendo2Geonames .

:Jamendo2Geonames a void:Linkset ;
                  void:linkPredicate foaf:based_near ;
                  void:subjectsTarget :Jamendo ;
                  void:objectsTarget :Geonames .
```

The above snippet basically states that there are `foaf:based_near` links **from** Jamendo **to** Geonames and that these links are contained in the Jamendo dataset (see also the RDF graph in the following).

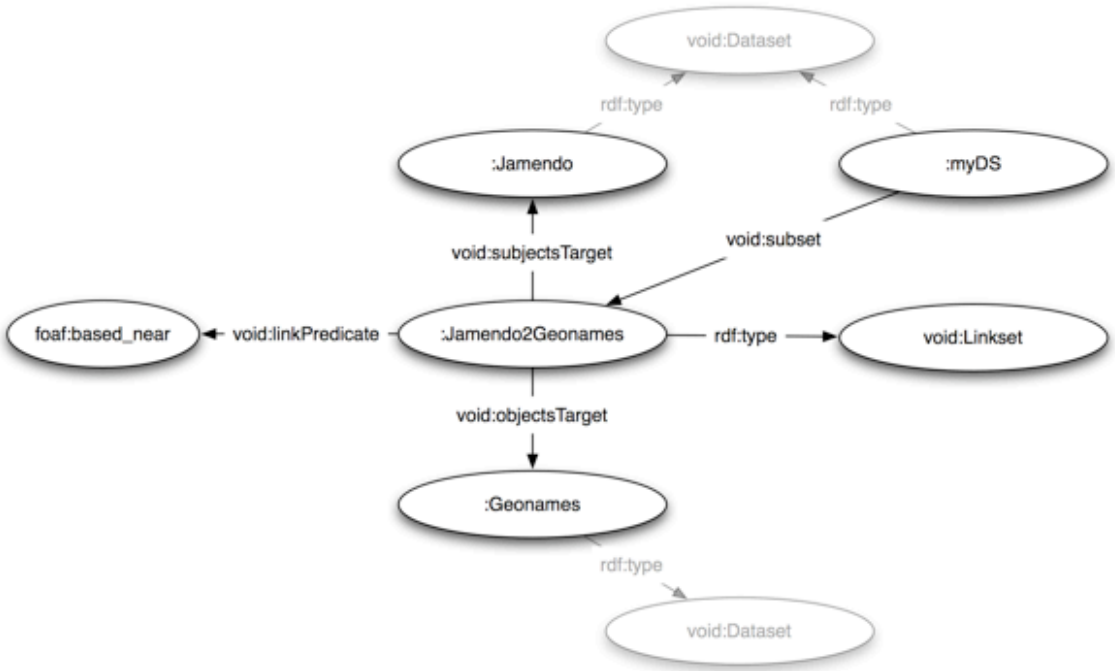


The directed case may of course as well play together with the 3rd-party interlinking such as in the following example where some dataset (`:myDS`) contains information about the interlinking between Jamendo and Geonames:

```
:myDS void:subset :Jamendo2Geonames .

:Jamendo2Geonames a void:Linkset ;
                  void:linkPredicate foaf:based_near;
                  void:subjectsTarget :Jamendo ;
                  void:objectsTarget :Geonames .
```

Again, the according RDF graph may render these facts more clearly:



3. Expressing dataset statistics

Note: The mechanism described in this section is considered experimental and may be removed or significantly changed in future versions of this document.

We have discovered several issues with the mechanism as described here:

- In SCOVO, `scovo:Items` are grouped into `scovo:Datasets`, and there seems to be an implicit assumption that all items in such a dataset share the same dimensions. As described here, we attach items directly to a `void:Dataset`, which leads to mixing of items of different dimensionality. On the other hand, the correct SCOVO modelling would lead to awkwardly complex notation for simple statistics.
- We encourage the use of classes and properties in places where SCOVO requires an instance of `scovo:Dimension`. This breaks the symmetry of the SCOVO model. SCOVO would require us to create a `scovo:Dimension` for each class or property. This would be quite verbose.
- Because of the issues above, SPARQLing for statistics can be awkward. It will often require a verbose check to make sure that an item has only certain dimensions and no others.

Users of the statistics feature should be aware of these issues, and of the possibility that this mechanism might be changed or replaced in the future.

This section describes how to express statistics about a dataset, such as the number of RDF triples contained in the dataset, or the number of resources described in the dataset. We use the Statistical Core Vocabulary (SCOVO). The main concept in SCOVO is the `scovo:Item`, which records a single number or statistical value. We use the `void:statItem` property to connect a `scovo:Item` to the `void:Dataset` it describes.

3.1 Basic dataset statistics

The following example states the number of triples in the DBpedia dataset.

```
:DBpedia a void:Dataset ;
         ...
         void:statItem [
            scovo:dimension void:numberOfTriples ;
            rdf:value 212576239 ;
         ] ;
```

A statistical data item is always used together with one or more **dimensions**, to indicate what kind of value the item represents. In voiD three dimensions are defined:

`void:numberOfTriples`	The `scovo:Item` represents the number of RDF triples.
`void:numberOfResources`	The `scovo:Item` represents the number of RDF resources, usually with distinct URIs.
`void:numberOfDocuments`	The `scovo:Item` represents the number of documents that contain RDF data, such as RDF/XML documents or RDFa-annotated web pages. Non-RDF documents, such as web pages in HTML or images, are usually not included in this count.
`void:numberOfDistinctSubjects`	The `scovo:Item` represents the number of distinct subjects, that is, the count of resources in an RDF statement `s p o` in the `s` position, with duplicates removed.
`void:numberOfDistinctObjects`	The `scovo:Item` represents the number of distinct objects, that is, the count of resources in an RDF statement `s p o` in the `o` position, with duplicates removed.

Table 3.: Defined statistical dimensions of voiD.

SCOVO is extensible, and arbitrary other dimensions may be defined and used to provide statistical information about a dataset or linkset.

3.2 Statistics about linksets

The same as with datasets, statistics about linksets can be provided. Most importantly, the `void:numberOfTriples` dimension can be used to record the number of links in a linkset. Example:

```
:DBpedia2DBLP a void:Linkset;
              void:target :DBpedia;
              void:target :DBLP;
              void:linkPredicate owl:sameAs;
              void:statItem [
                 rdf:value 10000;
                 scovo:dimension void:numberOfTriples;
              ] .
```

3.3 Number of instances of a class or property

How can I say how many instances of a particular class, such as `foaf:Person`, are described in a dataset? By using the class as a second `scovo:dimension` property in addition to `void:numberOfResources`:

```
void:statItem [
   rdf:value 20000;
   scovo:dimension void:numberOfResources ;
   scovo:dimension foaf:Person ;
] .
```

The number of triples with a particular predicate can be expressed with the same approach, by using the property as a dimension together with `void:numberOfTriples` .

3.4 Attributing statistics to a source

Sometimes it can be useful to record where a statistical data point came from. Does this number come from the dataset's homepage, or from a blog post by the dataset publisher, or from a manual count? We use `dcterms:source` on the `scovo:Item` to record additional information about the number's provenance.

```
void:statItem [
   rdf:value 22000;
   scovo:dimension void:numberOfTriples ;
   dcterms:source <http://lists.example.org/Archives/2008Sep/0327.html>
] .
```

Of course, additional information about the source can be recorded:

```
<http://lists.example.org/Archives/2008Sep/0327.html>
   dcterms:title "Mail from Joe Bloggs announcing the new dataset" ;
   dcterms:date "2008-09-23"^^xsd:date .
```

4. Publishing voiD descriptions of datasets

Publishers of RDF datasets should publish a voiD description together with the dataset itself. This helps users to find and access the dataset, and to integrate it with other sources of data. This section describes how to publish such a voiD description alongside a dataset. We note that when voiD descriptions are published along with their dataset, they can be crawled and indexed by search engines and semantic indexer and data user can hence query for it based on dataset characteristics (such as categories, vocabularies used, interlinking, statistics, available under license X).

4.1 Publishing a voiD file

Publishing a voiD file means to physically deploy it on the Web in an RDF serialisation. When deploying a voiD file one has basically three options:

1. Deploy it using Turtle RDF syntax [Turtle] in a file called `void.ttl`and make it accessible by assign an URI. **Pro**: easily readable by humans. **Con**: current parser support not overly well; higher costs of setup when accessing.
2. Deploy it using RDF/XML syntax [RDF/XML] in a file called `void.rdf` and make it accessible by assign an URI. **Pro**: wide parser support. **Con**: awkward syntax.
3. Deploy it using XHTML+RDFa syntax [RDFa] in the homepage of the dataset. **Pro**: can be directly embedded into the web page that describes the dataset hence does not require the publisher to maintain a separate voiD file. **Con**: weak current parser support (incl. SPARQL consumption).

Note: We have not decided on the canonical deployment format, yet. For the casual user, Turtle may seem to be the best option currently, but implementers are free to choose their deployment format as fit with their infrastructure. We explicitly solicit feedback on this issue!

4.2 Back-links

In voiD a dataset can be published as many documents on the Web. These documents can point back to the `void:Dataset` URI using `dcterms:isPartOf` such as shown below:

```
<http://dbpedia.org/data/Berlin> dcterms:isPartOf :DBpedia .
```

If the dataset consists of resources identified by dereferenceable URIs, dereferencing a URI will redirect to a document containing a description of that resource, and the `dcterms:isPartOf` triple linking back from the document to the dataset. This is especially handy in case when applying the follow-your-nose principle where one often has a single URI as a starting point. The question is then how to find the way from this URI to the entire dataset, that is, in the first place to identify the according voiD description; see also section 5.1 Discovery via links in the dataset's documents for the discovery process.

4.3 Tools

In order to support you to create voiD descriptions, we expect tools and libraries to emerge in the near future. One example currently already available is liftSSM, an XSLT that takes a semantic sitemap in XML and creates a stub voiD description in RDF/XML. Take for example the following semantic sitemap as the starting point:

```
<?xml version="1.0" encoding="UTF-8"?>
  <urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9"
          xmlns:sc="http://sw.deri.org/2007/07/sitemapextension/scschema.xsd'
    <sc:dataset>
      <sc:datasetLabel>Example Corp. Product Catalog</sc:datasetLabel>
      <sc:datasetURI>http://example.com/catalog.rdf#catalog</sc:datasetURI>
      <sc:linkedDataPrefix slicing="subject-object">http://example.com/produc
      <sc:sampleURI>http://example.com/products/widgets/X42</sc:sampleURI>
      <sc:sampleURI>http://example.com/products/categories/all</sc:sampleURI>
      <sc:sparqlEndpointLocation slicing="subject-object">http://example.com/
      <sc:dataDumpLocation>http://example.com/data/catalogdump.rdf.gz</sc:dat
      <sc:dataDumpLocation>http://example.org/data/catalog_archive.rdf.gz</sc
      <sc:dataDumpLocation>http://example.org/data/product_categories.rdf.gz<
      <changefreq>weekly</changefreq>
    </sc:dataset>
```

```
  </urlset>
```

After applying the liftSSM transformation the resulting RDF/XML document would look like:

```
<?xml version="1.0" encoding="UTF-8"?>
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns:sc="http://sw.deri.org/2007/07/sitemapextension/scschema.xsd
           xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
           xmlns:dcterms="http://purl.org/dc/terms/"
           xmlns:foaf="http://xmlns.com/foaf/0.1/"
           xmlns:scovo="http://purl.org/NET/scovo#"
           xmlns:void="http://rdfs.org/ns/void#">
    <void:Dataset rdf:about="http://example.com/catalog.rdf#catalog">
      <dcterms:comment>Example Corp. Product Catalog</dcterms:comment>
      <foaf:homepage rdf:resource="http://example.org/dataset.html"/>
      <dcterms:subject rdf:resource="http://dbpedia.org/resource/EXAMPLE"/>
      <dcterms:creator rdf:resource="http://example.org/creator#me"/>
      <dcterms::license rdf:resource="http://creativecommons.org/licenses/by/
      <void:statItem rdf:parseType="Resource">
        <scovo:dimension rdf:resource="http://rdfs.org/ns/void#numOfTriples"/
        <rdf:value rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">0<
      </void:statItem>
      <void:exampleResource rdf:resource="http://example.com/products/widgets
      <void:exampleResource rdf:resource="http://example.com/products/categor
      <void:sparqlEndpoint rdf:resource="http://example.com/sparql"/>
      <void:dataDump rdf:resource="http://example.com/data/catalogdump.rdf.gz
      <void:dataDump rdf:resource="http://example.org/data/catalog_archive.rd
      <void:dataDump rdf:resource="http://example.org/data/product_categories
      <void:uriPattern>^http://example.com/products/$</void:uriPattern>
    </void:Dataset>
  </rdf:RDF>
```

5. Consuming voiD descriptions

When a user retrieves a document containing data that may belong to a dataset, they may well want to know about the dataset it belongs to. How can dataset publishers make this information available?

5.1 Discovery via links in the dataset's documents

For datasets that are published as a collection of RDF documents in the linked data style, the preferred mechanism of discovering an associated voiD description is the back-link mechanism described in section 4.2 Back-links. Clients should look for a `dcterms:isPartOf` triple that links the RDF document to the dataset:

```
    <document.rdf> dcterms:isPartOf <void.ttl#MyDataset>
```

5.2 Discovery via sitemap.xml

This section describes a discovery mechanism for voiD descriptions based on `robots.txt` and `sitemap.xml`. This is useful in several situations:

- Datasets where individual documents contain no back-links

- Datasets that are not available as linked data
- Where the client only knows the SPARQL endpoint of a dataset
- For checking if a given domain contains any RDF datasets

The discovery-via-sitemaps in voiD is defined as follows:

1. Given a domain name, the client gets the file `robots.txt` and searches for a line that starts with '`Sitemap :`'; the rest of that line is the URI of a sitemap. See the specifications for robots.txt and the sitemap protocol for more details.
2. The semantic sitemaps extension to the sitemap protocol defines a `<sc:dataset>` element that can have a `<sc:datasetURI>` child element. The value of that element is a URI that identifies the dataset.
3. The dataset URI is dereferenced (as always, this might involve truncating the hash part of the URI or following an HTTP redirect). The final result of the dereferencing should be a voiD description of the dataset.

5.3 Smushing based on dataset homepages

One scenario for the use of voiD is to aggregate voiD descriptions of different datasets from different providers into a single collection. In this case, the same dataset can occur multiple times with different URIs. For example, if DBpedia publishes a voiD description, then it might use this URI:

```
http://dbpedia.org/void.ttl#DBpedia
```

But if Geonames publishes a voiD description, including a linkset for their links to DBpedia, it might use a different URI:

```
http://geonames.org/void.rdf#dbpedia
```

In a voiD collection that contains both these voiD descriptions, it is often desirable to 'smush' these two URIs into a single resource. The recommended process is **IFP smushing** on `foaf:homepage` property: dataset resources that have the exact same homepage URI should be combined into a single resource.

5.4 Example Queries

The following queries allow the selection of datasets from a collection of voiD descriptions based on different criteria.

5.4.1 By Category

```
SELECT DISTINCT ?dataset
WHERE {
  ?dataset a void:Dataset .
  ?dataset dcterms:subject <http://dbpedia.org/resource/Computer_science> .
}
```

Returns all datasets that are about 'Computer Science'. See also example query 1 online.

5.4.2 By Interlinking with a Certain Dataset

```
SELECT DISTINCT ?dataset
WHERE {
  ?dataset a void:Dataset .
  ?linkset void:target ?dataset ;
                       void:target :Geonames .
}
```

Returns all datasets that contain links that interlink them with Geonames. See also example query 2 online.

5.4.3 By Interlinking with a Certain Type

```
SELECT ?dataset
WHERE {
  ?dataset a void:Dataset ;
  void:subset ?linkset .
  ?linkset void:subjectsTarget ?dataset ;
  void:objectsTarget :Geonames;
  void:linkPredicate foaf:based_near .
}
```

Returns all datasets that contain `foaf:based_near`-links to Geonames. See also example query 3 online.

5.4.4 By Vocabulary

```
SELECT ?dataset
WHERE {
  ?dataset a void:Dataset ;
  void:vocabulary <http://xmlns.com/foaf/0.1/> .
}
```

Returns all dataset that use the FOAF vocabulary. See also example query 4 online.

5.4.5 By URI (Regex) Pattern

If we have a URI `http://dbpedia.org/resource/Amsterdam`, and we want to find datasets that might contain triples using that URI, we can do the following query:

```
SELECT ?dataset
WHERE {
  ?dataset a void:Dataset .
  ?dataset void:uriRegexPattern ?pattern .
  FILTER(REGEX("http://dbpedia.org/resource/Amsterdam", ?pattern))
```

```
}
```

Returns all datasets with a URI Regex Pattern that matches `"http://dbpedia.org/resource/Amsterdam"`. See also example query 5 online.

5.4.6 By the Identifier of a Dataset

If multiple voiD descriptions about a dataset are published by different parties, this dataset might be given different URIs in each voiD description. Thus, we will use the `foaf:homepage` property, which uniquely identifies the dataset, to gather these descriptions, smushing them under a single URI.

```
CONSTRUCT {
 :DBpedia ?p ?o
}
WHERE {
  ?dataset a void:Dataset; foaf:homepage <http://dbpedia.org/> ; ?p ?o .
}
```

Smushes all the properties of any dataset description with a `foaf:homepage` property of http://dpbedia.org/, into a single RDF description. See also example query 6 online.

Cheat Sheet

Note that classes start with a capital letter, properties with lower case. The following table is intended to give a quick overview; it is not normative. The only normative place is at the voiD vocabulary location at http://rdfs.org/ns/void/.

The entities defined in the voiD vocabulary (in alphabetical order, classes first) are:

voiD entity	description	example
`void:Dataset`	A dataset is a collection of data, published and maintained by a single provider, available as RDF on the Web, where at least some of the resources in the dataset are identified by dereferencable URIs.	see section 1. Describing Datasets
`void:Linkset`	A linkset is a subset of a dataset, consisting of triples where the subject is a resource hosted in one dataset, and the object is a resource hosted in another.	see section 2. Describing Dataset Interlinking

voiD cheat sheet.

void:TechnicalFeature	To express a certain technical feature of a dataset, such as formats (available in RDF/XML, etc.).	see section 1.5 Technical Description (Features)
void:dataDump	If an RDF dump of the dataset is available, then its location can be announced using this property.	see section 1.6 SPARQL endpoints, URI lookup, and RDF data dumps
void:exampleResource	Gives an example of a representative resource of a dataset.	see section 1.1 General Dataset Metadata
void:feature	Defines which technical features a dataset supports.	see section 1.5 Technical Description (Features)
void:linkPredicate	States that the interlinking is about a certain RDF predicate.	see section 2. Describing Dataset Interlinking
void:objectsTarget	For RDF properties (such as foaf:based_near) where the direction shall be stated explicitly this property is used to indicate the sink of the interlinking.	see section 2.2 Non-directed vs. Directed
void:sparqlEndpoint	Announcement of a SPARQL endpoint.	see section 1.6 SPARQL endpoints, URI lookup, and RDF data dumps
void:statItem	States a certain quantitative measure.	see section 3.1 Basic dataset statistics
void:subjectsTarget	For RDF properties (such as foaf:based_near) where the direction shall be stated explicitly	see section 2.2 Non-directed

	this property is used to indicate the source of the interlinking.	vs. Directed
void:subset	Defines a logical subset of a dataset (a linkset or a dataset containing only certain resources of a domain, for example).	see section 2. Describing Dataset Interlinking
void:target	States that a linkset has a certain dataset as its target.	see section 2. Describing Dataset Interlinking
void:uriLookupEndpoint	Defines a simple URI look-up protocol for accessing a dataset based on an URI end-point.	see section 1.6 SPARQL endpoints, URI lookup, and RDF data dumps
void:uriRegexPattern	A regular expression pattern that matches one or more URI in the dataset.	see section 1.1 General Dataset Metadata
void:vocabulary	Links a dataset to an RDFS vocabulary or OWL ontology whose classes and properties are used in the dataset.	see section 1.7 Vocabularies used in a dataset

References

[DCMI type voc]
DCMI Type Vocabulary, January 2008. URI: http://dublincore.org/documents/2008/01/14/dcmi-type-vocabulary/

[RFC2119]
Key words for use in RFCs to indicate requirement levels, RFC 2119, S. Bradner, March 1997. URI: http://www.rfc-editor.org/rfc/rfc2119.txt

[RDFa]
RDFa Primer, B. Adida, M. Birbeck,W3C Working Group Note 14 October 2008. URI: http://www.w3.org/TR/2008/NOTE-xhtml-rdfa-primer-20081014/

[RDF/XML]
RDF/XML Syntax Specification (Revised), D. Beckett, W3C Recommendation 10 February 2004. URI: http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/

[Turtle]
Turtle - Terse RDF Triple Language, D. Beckett, T. Berners-Lee, W3C Team Submission 14 January 2008. URI: http://www.w3.org/TeamSubmission/2008/SUBM-turtle-20080114/

[voiD vocabulary]
voiD Vocabulary, K. Alexander, R. Cyganiak, M. Hausenblas, J. Zhao, January 2009. URI: http://rdfs.org/ns/void/

[WOD Discovery]
Discovery and Usage of Linked Datasets on the Web of Data, M. Hausenblas, NodMag #4, 2008. URI: http://www.talis.com/nodalities/pdf/nodalities_issue4.pdf

Acknowledgements

Our thanks go out to some chaps who influenced the design of voiD, provided use cases and ensured that we would never get bored too quickly. These people were (alphabetically): Orri Erling, Hugh Glaser, Olaf Hartig, Tom Heath, Ian Millard, Marc-Alexandre Nolin, Yves Raimond, Yrjänä Rankka, Francois Scharffe, Giovanni Tummarello.

The work has partly been supported by the following projects: EC FP7 project ICT-2007.1.2 ROMULUS (http://www.ict-romulus.eu/), etc.