Paper TT03

A Primer on Converting Analysis Results Data to RDF Data Cubes using Free and Open Source Tools

Tim Williams, UCB BioSciences Inc, Raleigh, USA

ABSTRACT

There is a wealth of information describing Resource Description Framework (RDF) and Semantic Web technologies. This is contrasted by a lack of simple examples that allow beginners to experiment and learn using familiar data. Two methods are presented for converting an abridged demographics table to an RDF Data Cube. The resulting RDF file is queried using the SPARQL processor ARQ. All steps can be completed on a local machine with minimal configuration and no license fees.

Supporting documentation and files, including installation instructions, a workbook for cube construction using OpenRefine, source data, scripts, and outputs are available on the PhUSE wiki [18].

This paper is derived from the activities of the PhUSE Results Metadata working group.

INTRODUCTION

The vastness and complexity of the Semantic Web landscape is often overwhelming for beginners. The goal of this paper is to enable you to convert your familiar data to the Turtle (TTL) serialization of RDF (**Figure 1**). Exploration of the data structure and relationships enables you to progress to more advanced topics. If you already have experience in this area, you may find the basic examples lacking in many of the features that make the Linked Open Data approach so attractive. Ontologies are discussed only in passing. Inferencing, data reconciliation, use of code lists, and constructing cube slices are beyond the scope of this introductory paper.

```
ns:obs1 a qb:Observation ;
  qb:dataSet ns:dataset-demog3DimSource ;
  rdfs:label "1" ;
  prop:Treatment <code/treatment/Plc> ;
  prop:Sex <code/sex/F> ;
  prop:Statistic <code/statistic/count> ;
  prop:Result 12 ;
```

Figure 1. An observation in the TTL serialization of RDF, created by Publisci.

File:demog3Dim_p.ttl

PhUSE provides a list of resources for learning about the Semantic Web [11]. A basic knowledge of RDF is assumed for this paper but is not required. I hope to provide you with a pathway into the technology to facilitate your own discoveries. Description of the cube vocabulary is confined to the main classes and properties used in the examples. Please see the full cube description [3] once you are familiar with the basic concepts.

Two methods for converting the same source data file to RDF Cubes are described. The first approach uses a driver script where only the fundamental components are specified. The second requires additional knowledge of cube structure. Advantages and disadvantages of both approaches are listed.

Instructions and examples use the MS Windows operating system. Installation and configuration instructions for other systems are available within the resources noted in the References section.

RDF DATA CUBE

An advantage of converting your data to RDF is the ability to reconcile it against common values, thus enabling standardization and interchange within and among organizations. Other advantages include:

- Data storage in an open, non-proprietary format.
- Data access with R (with the rrdf and rrdflibs packages) or custom SAS® macros [1].
- Values, metadata, and linkages which facilitate exploration and discovery.
- Metadata is stored *with* the data instead of having to rely on the data structure or ancillary metadata sources.
- Machine interpretation and logic when used with ontologies and reasoning engines.
- A data model that is infinitely extensible.

The RDF Data Cube is a way to organize RDF data into a structure of addressable values and categorical groupings (known as *slices* of the cube). Addressable components enable data integration and comparison. Use of common terms for dimensions, units, metadata, and values means standardization becomes an integral part of the data derivation process. Integration of metadata gives embedded context for values.

Become familiar with basic cube terminology and components prior to mapping your source data to the cube structure.

DATA CUBE VOCABULARY

data.gov.uk sponsored a group of experts to develop a vocabulary for publication of statistical and related data in linked data format. The result of their work, "The RDF Data Cube Vocabulary" [3], is built upon and compatible with the cube model defined by the Statistical Data and Metadata eXchange (SDMX) [14]. The SDMX vocabulary provides an extensible platform for publishing statistical data. In this model, observations are contained within the concept of a **data set** which is comprised of three types of components: dimensions, measures, and attributes.

It is convenient to think of the **dimensions** as the pieces needed to locate result values within the cube. For example, an X,Y coordinate plot could be considered a two dimensional cube. The X and Y coordinates identify a single result value within the two-dimensional grid. A cube may have any number of dimensions and those dimensions do not have to be symmetrical.

A single observation may include one or more **measures**. For the sake of simplicity, the instructions in this paper create cubes with a single measure per observation.

Attributes supply context and the ability to interpret a cube, its components, and its values. For example, a measure of weight may have the attribute "mg" which has a much different interpretation than a weight with attribute "lbs". Attributes may also describe the status of a data set or value as "preliminary" or "final". In order to focus on dimensions and measures, attributes receive little coverage in this paper. Attributes available from the source data can be added following steps similar to those for dimensions.

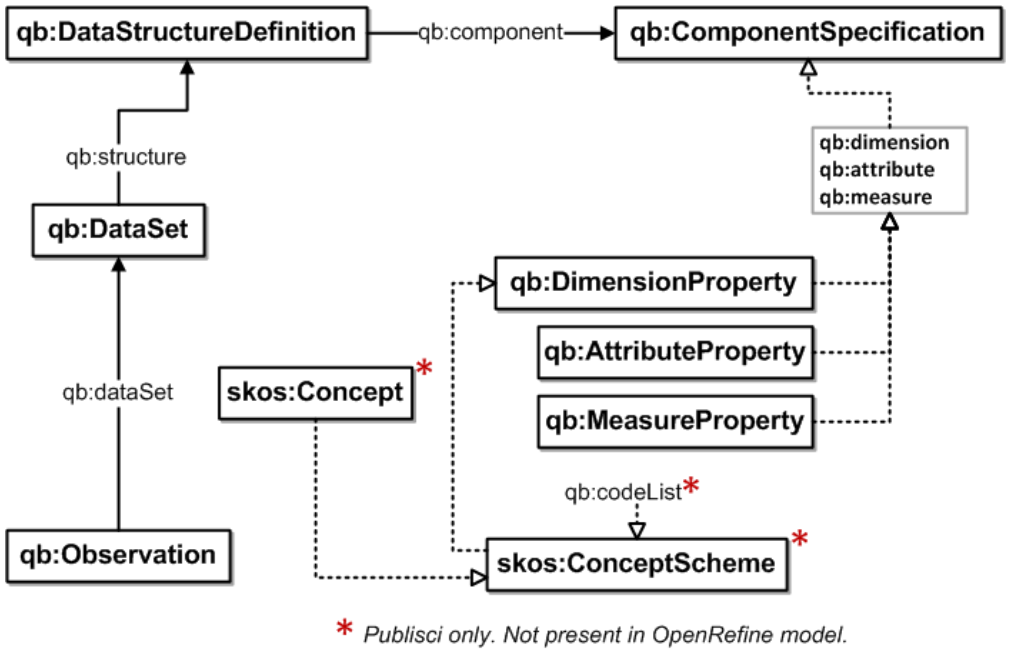

Figure 2 A reduced RDF Data Cube vocabulary for use in this paper. [1]

EXAMPLE DATA

Source data for cube construction is derived from an abridged demographics results table. The table presents the count and percentage of female and male patients in each treatment arm of a fictional study.

Baseline Characteristic	Placebo N=28	LowDose N=30	HighDose N=29
Sex			
F	12 (42.9)	14 (46.7)	16 (55.2)
M	16 (57.1)	16 (53.3)	13 (44.8)

Figure 3 An abbreviated demographics table.

Three **dimensions** are identified in **Figure 4** : 1. Treatment Arm with values Placebo, LowDose, HighDose; 2. Sex with values F, M; 3. Statistic, which displays the less obvious values of count and percentage to represent the *types* of values in the table. The observations themselves are the **measures**.

Baseline Characteristic	1. Treatment Placebo N=28	LowDose N=30	HighDose N=29
Sex			
2. F	12 (42.9)	14 (46.7)	16 (55.2)
M	16 (57.1)	16 (53.3)	13 (44.8)

3. Statistic: count / percentage

Figure 4 Dimensions identified.

Figure 5 shows a visualization of the values as a three dimensional cube. The top two slices of the cube, one for count and one for percentage, are assembled into the cube. This is one way to conceptualize how the components come together. Alternatively, the treatment or sex dimension could be shown as slices coming together to form the same cube.

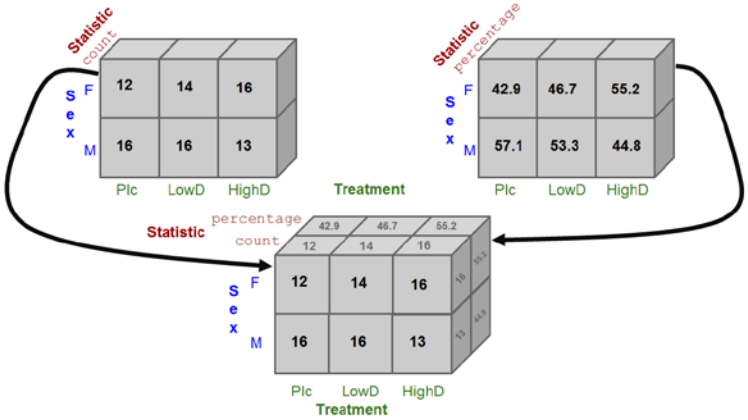

Figure 5 Demographics results visualized as a cube.

The demographics results from the display in **Figure 3** are translated into the comma separated value (CSV) file `demo3DimSource.csv`[2] shown in **Figure 6**. Dimensions and measure are represented as columns in the data file that serves as the data source for both cube creation methods illustrated in this paper.

```
Treatment,Sex,Statistic,Result
Plc,F,count,12
Plc,F,percentage,42.9
Plc,M,count,16
Plc,M,percentage,57.1
LowD,F,count,14,
LowD,F,percentage,46.7
LowD,M,count,16
LowD,M,percentage,53.3
etc.
```

Figure 6 Data excerpt in CSV format.
File: demog3DimSource.csv

CREATE A CUBE WITH PUBLISCI

One of the easiest ways to create your first RDF Data Cube is to use the Ruby gem Publisci. Do not be deterred if you lack Ruby programming experience. Installation of Ruby[13] and Publisci [15] is straightforward even for novices. Knowledge of Ruby is only needed if you wish to change or extend Publisci's functionality, which is not necessary in this example.

INSTALLATION

Administrator access on your local machine and access to the internet are required for installation. Firewall and security restrictions may prevent access to required components, so it is best to perform the installation when not behind a company firewall.

Download the Ruby Installer [13] and execute the installation file, accepting the default choices when prompted. Ensure that you check the boxes "Add Ruby executables to your PATH" and "Associate .rb and .rbw file with this Ruby Installation". Upon completion, confirm Ruby is available at the command line by opening a new command window and executing:

```
ruby –help
```

Installation of Ruby is now complete. You are ready to install the Publisci gem.

There are over thirty dependencies for the gem, so ensure you are not behind a firewall to make the installation automatic and painless. In theory it is possible to download and install each interdependent gem in sequence, but that approach is a sure recipe for madness.

From the command line, execute:

```
gem install publisci
```

Documentation and modules [15] required by Publisci takes a few minutes to install.

USAGE

You must now create a short Ruby program to drive the creation of the cube. The gem's author supplies examples on the Publisci website [15] and the code in **Figure 7** is available for download from the PhUSE Wiki [18].

The example program first includes the required Ruby components followed by specifying the input file (1). The three dimensions are defined (2) along with the single measure (3). The Base URL is specified as example.org to build the Universal Resource Identifiers (URI's) for the various cube components (4).

Metadata is added to the cube (5) before specifying the output file (6). Specify full paths for (1) and (6) when source data, programs, and outputs are not in the same folder.

```
require 'publisci'
include PubliSci::DSL

data do
(1) source 'demog3DimSource.csv'
    dimension 'Treatment', 'Sex', 'Statistic'(2)
(3) measure 'Result'
    option :base_url, 'http://example.org'(4)
    option 'label_column', 'Statistic'
end
(5) metadata do
    dataset 'Demographics Analysis Results'
    title 'Demographics'
    creator 'Your-Name-Here'
    description 'Publisci 3-Dim Cube'
    date '2014-07-07T00:00:00'
end
(6) open('demog3Dim_p.ttl','w'){|file| file.write generate_n3}
```

Figure 7 Ruby Program for three dimensional cube
File: demog3Dim.rb

Save the program as `demog3Dim.rb` and execute it from the command line[3]:

```
ruby demog3dim.rb
```

RESULT

Open the resulting `demog3Dim_p.ttl` file in any text editor to view and explore your creation. The cube lacks some of the elements found in well-formed cubes [3] like ranges on the dimensions. In contrast to the OpenRefine method, Publisci generates code lists automatically which is both efficient and instructive for the new user.

Advantages

- One of the fastest and easiest open-source ways to generate an RDF cube from a CSV file.
- Requires less knowledge of cube structure and relations.
- Easier to use than OpenRefine.
- Automatic code list generation.

Disadvantages

- No recent development on Publisci.[4]
- Not as flexible as OpenRefine for adding attributes and customizing components.
- Extension of capabilities, like adding cube slices, requires knowledge of Ruby.

CREATE A CUBE WITH OPENREFINE

INTRODUCTION

OpenRefine, formerly GoogleRefine, is a powerful tool for exploring, cleaning, fixing, reconciling, and transforming data. OpenRefine also offers the "next level" of cube construction customization using the RDF Refine extension. With increased customization comes increased complexity. A good familiarity with the cube model is required. You will define the structure (skeleton) from scratch and then attach values to the skeleton. Re-use of previously defined cube skeletons supplies starting points for similar cubes, thus avoiding the time consuming steps of defining the entire structure each time.

INSTALLATION

Installation requires a recent version of Java [6] with compatible versions of OpenRefine (2.5) and RDF Refine extension (0.8). Pay close attention to notes on the RDF Refine web page for compatibility of future versions.

Installation instructions and source files for both OpenRefine [10] and the RDF Extension [12] are available online. I present a concise installation guide for the Windows OS on the PhUSE wiki (*TT03_CompanionDoc-OpenRefine-InstallInstructions.docx*) as a companion to this paper [18].

OpenRefine has inherent versioning with its Undo/Redo project history. I recommend additional version control of the workspace to snapshot and identify key development time points like the completion of a cube skeleton.

CUBE CREATION PROCESS (**Figure 8**)

The detailed step-by-step instruction necessary to define a cube skeleton is too lengthy for the confines of this paper. Please see the companion guide on the PhUSE Wiki [18] or contact the author for details beyond the overview provided in this section.

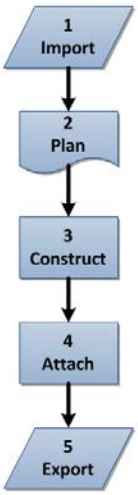

Figure 8

1. Import

OpenRefine requires data prior to designing the cube structure. Start the application, create a new project, and import your source data file. OpenRefine excels in data reconciliation so take time to review the source data.

Each observation must have a unique URI. The URI is typically created in one of two ways. The first approach uses General Refine Expression language (GREL) to form the observation URI by combining the values of the treatment, sex and statistics dimensions:

```
'http://www.example.org/dc/demog/dataset/'+cells["treatment"]+'-
'+cells["sex"]+'-'+cells["statistic"].value.urlify()
```

The resulting RDF for the observation that holds the count of females in the placebo treatment arm is:

```
ds:Plc-F-count a qb:Observation ;
```

Combining dimensions to form the URI is descriptive and concise when the cube has a small number of dimensions and the values of the dimensions are short. In real-world applications there will be many dimensions with longer values, leading to RDF that is difficult to read. I therefore chose to create a unique value column named Index that is then used in a GREL formula for the observation URI.

GREL code using a unique index to identify observations:

```
'http://www.example.org/dc/demog/dataset/obs'+cells["Index"].value.urlify()
```

Resulting RDF:

```
  ds:obs1 a qb:Observation ;
```

2. Plan

Source data is reviewed in the planning stage to identify the columns that become the cube components. **Dimensions** were identified in **Figure 4** as *treatment*, *sex,* and *statistic* and translated into a CSV file as columns of values along with a column named *result* for the **measure** values (**Figure 6**). Cube creation is easier and more consistent when a workbook is used to guide the cube creation process [18].

Decisions on cube structure, patterns, and prefixes must now be made. While the official cube structure specification [3] explicitly defines components and their relationships, the cube author has considerable latitude when defining paths and component names. The workbook becomes essential to document decisions and ensure consistency across multiple cube definitions. Choices were made in this presentation to reduce complexity and are subject to change in a real-world application.

External vocabularies are a necessary and beneficial part of cube creation. Beyond the basic cube definition (qb: http://purl.org/linked-data/cube#), there are several vocabularies that should be imported. Your choices depend on the type of data in the cube and the desired level of annotation and metadata.

3. Construct

Constructing the cube skeleton is time-consuming. External vocabularies are imported and the Base URI is specified. The DataSet, DataStructureDefinition, components, and properties must be described and added to the skeleton piece by piece. Cube authors should check their progress during construction by using the convenient "RDF Preview" tab.

Cube metadata is added. Metadata should include cube author (dct:creator), cube title (dct:title), description (dct:description), issue date (dct:issued), and other data the author feels is relevant.

Capture the skeleton as JSON code to allow reuse in future cubes. This is a substantial saving because skeleton creation is labour-intensive, time consuming, and error-prone.

4. Attach

Values from the source data are attached to the cube components. It is in this step where you will either use a unique index value to define observations or a combination of the values of each dimension in the cube (as described in Step 1. Import). The qb:Observation is attached to the DataSet to complete construction. The result is checked for the first ten observations using the "RDF Preview" tab or by exporting the cube to see all observations.

5. Export

The Export button in OpenRefine offers a variety of formats including **RDF as Turtle** and **RDF/XML**. "Export to Turtle" is used to create the TTL file in this paper.

RESULT

The simpler and more readable TTL file created by OpenRefine provides a more obvious connection between the resulting cube and the source data. However, this representation does not facilitate external linkages or code lists for data standardization and validation. As demonstrated in the companion workbook, URI's can be constructed in OpenRefine using the Open Refine Expression Language GREL. A note of caution: the GREL function `urlify()`converts case and spaces in the data, thus violating data integrity unless code lists are used. Consider the approach in this example as a building block toward a more advanced cube.

Advantages

- Ease of installation.
- Flexibility in cube structure definition.
- Incremental development with the ability to check the RDF along the way.
- Ability to capture and re-use RDF skeletons.
- Cube components available within the interface (after import of the qb prefix).
- An excellent tool for data exploration and reconciliation, even when not designing and creating data cubes [10] [17] .

Disadvantages

- Requires additional knowledge of the cube specification.
- Steep learning curve.
- Many steps needed to create the cube (somewhat mitigated by skeleton reuse).
- Dimensions specified as values and not URIs.
- Compatibility issues between versions of OpenRefine and the RDF plugin are well documented [12] but easy to miss.

SPARQL QUERIES WITH ARQ

Now that your data is in an RDF Cube, the next logical step is to upload it to a SPARQL endpoint to validate the cube structure and make it available for querying. This presents another daunting technology hurdle if you merely want to learn more about RDF and how to use SPARQL. However, ARQ [2] can query the TTL file locally, thus avoiding upload to a public endpoint or the need to install and configure an endpoint on your machine. Bob DuCharme's book "Learning SPARQL"[4] is an excellent resource that includes instructions on downloading and using ARQ for querying local RDF files.

ARQ is a Java-based part of the Apache Jena framework. Once you cover the basics with ARQ, you may wish to investigate Jena Fuseki [7] to expose your triples over HTTP on a local or remote endpoint.

INSTALL

Download ARQ from the Jena website [2] and unzip the binary .zip file into a convenient folder. For users of the Windows OS, find the `arq.bat` file in the /bat subfolder. Add the path to the /bat folder to your system path. Create a new environment variable called `JENA_HOME` that contains the path to the folder above /bat. You are now ready to start creating your first queries.

Usage

ARQ is run from the command line and uses the format:

```
arq –-data <rdf filename>.ttl --query <query filename>.rq
```

The following examples illustrate queries of the cube created with OpenRefine and exported as a Turtle TTL file. The queries could be adjusted for the Publisci cube's use of code lists and `skos:prefLabel`. The examples assume the `.ttl` and `.rq` files are in the same folder. Add complete path specifications to the calls as needed.

Example 1
Number of females in the placebo group.
The answer is found at the intersection of the three dimensions: `treatment`='Plc', `sex`='F' and `statistic`='count'.

```
PREFIX prop:  <http://www.example.org/dc/demog/prop/>
SELECT ?value
WHERE
{
  ?obs prop:treatment "Plc";
       prop:sex "F";
       prop:statistic "count";
       prop:result ?value.
}
```

Figure 9 SPARQL Query: Number of Females in the Placebo group.
File: demog3DimPlcFemale.rq

To execute the query: `arq --data demog3Dim.ttl --query demog3DimPlcFemale.rq`

Result:
```
-------------------------------------------------
| value                                         |
=================================================
| "12"^^<http://www.w3.org/2001/XMLSchema#double> |
-------------------------------------------------
```

Example 2
Number of females in all treatments.
The answer is found by summing the `result` values for counts (`statistic= "count"`) of females (`sex='F'`) across all treatment arms.

```
PREFIX prop:  <http://www.example.org/dc/demog/prop/>
SELECT (SUM(?value) as ?FPlcSum)
WHERE
{
  ?obs prop:sex "F";
       prop:statistic "count";
       prop:result ?value.
}
```

Figure 10 SPARQL Query: Number of Females in all treatments.
File: demog3DimAllFemale.rq

To execute the query: `arq --data Demog3Dim.ttl --query demog3DimAllFemale.rq`

Result:
```
-----------
| FPlcSum |
===========
| 42.0e0  |
-----------
```

Both results require post-query formatting before presentation in a report. This is often accomplished with custom API because SPARQL provides little in the way of labelling and formatting options. SPARQL results are more functional than attractive.

CONCLUSION

Publisci is a quick and easy way to start working with RDF Data Cubes. Once you become familiar with the vocabulary and components, the benefits of OpenRefine's flexibility become evident. Re-use of JSON skeletons for similar datasets delivers some efficiency for an otherwise laborious task. Both approaches provide a path to learning about RDF data cubes but are unlikely candidates for conversion of clinical trials data to RDF in a production environment. Structures developed with OpenRefine may guide cube creation using R or SAS as part of a validated data conversion process.

Successful creation of the demographics cube leads to the desire to create cubes for other analysis results. Multiple cubes could be brought together as nested cubes, as demonstrated by Lefort and Leroux [8]. A main index cube would reference the specialized cubes.

The cubes created in this paper are not well-formed according to "The RDF Cube Vocabulary" [3], though the *Publisci* version comes the closest. Compromises were made in order to reduce complexity for users new to the concept. Noticeably absent are ranges for dimensions, slices, and code lists (for the OpenRefine cube). The OpenRefine version uses values for dimensions instead of URI's, thus allowing for simplified SPARQL queries. Cubes that are closer to meeting the recommended

structure can be validated directly using SPARQL queries [3] or tools that offer graphical conformance reports like the LOD2 Technology Stack [9] .

The RDF Cube model is not without its challenges. New users face a steep learning curve and a dearth of standard tools hampers adoption. The challenge of increased storage requirements for RDF is no longer a big factor thanks to recent advances in storage cost and speed.

Interest in Semantic Web technology within the Pharmaceutical Industry is on the rise. Core CDISC standards are now available as RDF and several companies are piloting Semantic Web approaches. PhUSE working groups like the "Protocol Representation Model (PRM) in RDF" and the "Results and Metadata in RDF" are actively pursuing solutions in this area. Please consider joining these or other initiatives in this exciting and growing field.

REFERENCES

1. Andersen, M. "SAS-RDF-writer", "SAS-SPARQLwrapper". <https://github.com/MarcJAndersen> (Retrieved: 09-Aug-2014).

2. "ARQ – A SPARQL Processor for Jena," *Installation source and documentation*. <http://jena.apache.org/documentation/query/> (Retrieved:: 09-Aug-2014).

3. Cygniak, R., Reynolds D. and Tennison J. "The RDF Data Cube Vocabulary," *W3C Recommendation 16 January 2014*. <http://www.w3.org/TR/vocab-data-cube/> (Retrieved: 09-Aug-2014).

4. DuCharme, B. (2013), "Learning SPARQL. Querying and Updating with SPARQL 1.1," *2nd Edition*, Sebastopol, CA: O'Reilly Media Inc.

5. Goto, N., Prins, P., Nakao M., et al. "BioRuby: bioinformatics software for the Ruby programming language," <http://bioinformatics.oxfordjournals.org/content/26/20/2617> (Retrieved: 09-Aug-2014).

6. Java. *Installation source and documentation*. <http://www.java.com/download> (Retrieved: 09-Aug-2014).

7. Jena Fuseki. *Installation source and documentation*.<http://jena.apache.org/documentation/serving_data/> (Retrieved: 09-Aug-2014).

8. Lefort, L. and Leroux, H. "Design and generation of Linked Clinical Data Cubes," <http://www.ict.csiro.au/staff/laurent.lefort/SemStats2013-Lefort.pdf> (Retrieved: 09-Aug-2014).

9. LOD2 Technology Stack. *Cube validation and other tools*. <http://stack.lod2.eu/> (Retrieved: 09-Aug-2014).

10. OpenRefine. *Installation source and documentation*. <http://openrefine.org> (Retrieved: 09-Aug-2014).

11. PhUSE Semantic Technology Curriculum. <http://www.phusewiki.org/wiki/index.php?title=Semantic_Technology_Curriculum> (Retrieved: 09-Aug2014).

12. "RDF Refine – a Google Refine extension for exporting RDF," *Installation source and documentation*. <http://refine.deri.ie/> (Retrieved: 09-Aug-2014).

13. Ruby Installer for Windows. *Installation source*. <http://rubyinstaller.org/> (Retrieved: 09-Aug-2014).

14. Statistical Data and Metadata eXchange (SDMX). <http://sdmx.org/> (Retrieved: 09-Aug-2014).

15. Strinz, W. Ruby Gem Publisci. *Installation source and documentation*. <https://github.com/wstrinz/publisci> (Retrieved: 09-Aug-2014).

16. The Ruby Science Foundation (2013), "SciRuby: Tools for scientific computing in Ruby," <http://sciruby.com> (Retrieved: 09-Aug-2014).

17. Verborgh, R., De Wilde, M. (2013), "Using Open Refine," Birmingham: PACKT Publishing.

18. Williams, T. (2014). *Supporting documents for paper TT03*. PhUSE Wiki [search by author name, paper number TT03, or title]: <http://www.phusewiki.org> (Retrieved: 09-Aug-2014).

ACKNOWLEDGEMENTS

The author is indebted to the following people and organizations:

- Marc Anderson for introducing me to the RDF Data Cube.
- Ian Fleming and the team members of the PhUSE Results Metadata working group.
- Will Strinz for developing and discussing Publisci.
- Bob DuCharme for his excellent book on SPARQL [4].
- Numerous contributors to the Semantic Web, RDF, Linked Open Data communities.

This paper is based on free, open-source software and the efforts of volunteers in PhUSE working groups. Please support those who donate their time and expertise through your own collaboration, participation, and promotion of these activities.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Tim Williams
UCB BioSciences, Inc
P.O. Box 110167
Raleigh, NC 27613
Work Phone: +1-919-767-5997

Email: tim.williams@ucb.com
www.linkedin.com/in/timpwilliams

ENDNOTES

[1] Absent from the Cube model for the sake of simplicity: attributes, slices, code lists, and linkages to SKOS and SDMX concepts. See [3] for the full representation.
[2] Source data and other files referenced in this paper are available on the PhUSE Wiki [18].
[3] Other execution methods are possible; command line execution is illustrated because I am an "old school" programmer.
[4] Publisci's author Will Strinz has indicated a willingness to discuss modifications if there is enough interest.