

OBO Tutorial

James A. Overton

Copyright 2014, Distributed under the CC-by 3.0 License

Contents

Introduction	2
Data Before	2
Outline	3
Data After	4
Names and Naming	5
All the Names We Need	5
Names for Rules for Names	6
No More than One Name	8
Open Biological and Biomedical Ontologies	8
Reference Ontologies and Application Ontologies	9
Shared Best Practises	9
Using and Reusing Ontology Terms	9
Finding Ontology Terms	10
Finding Ontologies	12
Assessing Ontologies and Terms for Reuse	13
Mapping Terms	15
Importing Terms	16
Putting it all Together	21

Processing Data with Ontologies	21
Tables to Triples	22
Using SPARQL with OWL Ontologies	22
Converting Instance Data to RDF/OWL	22
Querying Instance Data with DL Query and SPARQL	22
Best Practises for Reasoning Over Large Data Sets	22
Updating, Testing, and Releasing Ontologies	22
Tracking Changes with Ontology Diff Tools	22
Filing Effective Term Requests and Bug Reports	22
Best Practices for Committing Changes to an Ontology	22
Quality Checking for Ontologies	22
Unit Testing for Ontologies	22
Continuous Integration with Jenkins	22
Release Workflows with OWLTools	22
Publishing Ontologies	22
Managing PURLs	22

Introduction

In this tutorial we use a running example: histology assays of mice and rats. The example is small and its fictional but it's based on real datasets that I've worked with using these same tools and techniques. Biology and biomedicine are vast fields, and this example won't be a close fit to every reader's field of research. The advantage is that the running example keeps the tutorial concrete and practical. Once you learn the various techniques for working with this example data, I hope you'll see how you can apply the same techniques to your work in your own field.

Data Before

Take a look at the [data-before.csv](#) file. It's just a spreadsheet in comma-separated-values format. What information does it contain? It's hard to say without some context.

Each row of the spreadsheet describes an “observation” made by an investigator of a study subject – a mouse or a rat. The columns divide the information up

into different fields. What does each column mean? Lab computers are stuffed with spreadsheets with cryptic headers, or even lacking any headers. Those spreadsheets hang around long after the memory of what the columns mean has faded. And columns can become strange mixtures of special cases, especially when data about different sorts of things gets crammed into the same table. It's often more effective to ignore the headers, and try to generalize from the values in the column instead.

What about the values? These are often more cryptic than the headers. What does "B6C3F1" mean? Worse, familiar names can be very misleading. Does "histology" mean a general type of assay or a specific protocol for this study? I personally find many common date formats completely frustrating: do they mean "MM/DD/YY" or "DD/MM/YY"?

During the study, as it's being designed and the data is being collected, the meanings of the columns and the values will be clear to the investigators – more or less. The problem is communicating this information: to other researchers in the same lab, to researchers in other labs, and even to the original study authors at some time in the future. The solution to the communication problem is to be clear from the outset.

This tutorial is about some practical techniques that you can use to communicate your data better. Some of these techniques also improve search and analysis of your data. And most of these techniques are really very simple. There are really just two hard parts: applying them consistently yourself, and coordinating with others.

Outline

The plan for this tutorial is to build from better names, to systems of names, to using these systems with your data, and finally to updating and maintaining those systems.

1. [Introduction](#) (you are here!)
2. [Names and Naming](#): anything we want to talk about can be given a globally unique name, an IRI
3. [Open Biological and Biomedical Ontologies \(OBO\)](#): carefully constructed, standard systems of names (and statements)
4. [Using and Reusing Ontologies](#)
 - the difference between reference and application ontologies
 - finding the right ontology term
 - importing reference ontology terms into your application ontology

5. [Processing Data with Ontologies](#): using RDF, OWL, and SPARQL to work with your data
6. [Updating, Testing, and Releasing Ontologies](#)
 - contributing to reference ontologies
 - updating and maintaining your ontology

Ontologies are more than just systems of names. They express statements about the things that they name: statements about things in the world. Your data also make statements about the world. We work hard to make sure that all these statements are true. We also work hard to make sure that they're useful. By using OBO ontologies and the techniques in this tutorial, we can link our data together into a larger system of statements about the world. By communicating clearly, and sharing what we know, we make our data more valuable than ever before.

Data After

Let's return to the [data-before.csv](#) spreadsheet. Now we can spell out the meaning of each column, the current sorts of values that it contains, and how to make those values more clear.

- datetime: when the observation was made
- investigator: the initials of the person who made the observation
- subject: the identifier of the subject that was observed
- species: the name of the species of the subject
- strain: the name of the genetic variant that the subject belongs to
- sex: the biological sex of the subject
- group: the identifier of the group that the subject is assigned to in the study
- protocol: the name of the protocol used for the observation
- organ: the name of the subject's organ that was observed
- morphology: the name of the disease or other abnormal state of the organ
- qualifier: a keyword to further describe the morphology
- comment: unstructured notes made by the investigator about the observation

Column	Current Format	Better Format	Technique
datetime	idiosyncratic date string	ISO standard	SPARQL
investigator	investigator initials	ORCID	replace
subject	string	application ontology IRI	SPARQL

Column	Current Format	Better Format	Technique
species	English, mixed case	NCBITaxon IRI	MIREOT
strain	string	application ontology IRI	QTT
sex	English, mixed case	PATO IRI	MIREOT
group	string	application ontology IRI	SPARQL
protocol	ambiguous string	OBI IRI	MIREOT
organ	English, mixed case	UBERON IRI	extract
disease	English, mixed case	MPATH IRI	MIREOT
qualifier	English, mixed case	PATO IRI	MIREOT
comment	English unstructured text	–	–

The results of these changes are in [data-after.csv](#) and [data.owl](#). To learn more about the techniques, [read on!](#)

Names and Naming

The [data-before.csv](#) file contains terse, ambiguous, and misleading names. This isn't unusual! How can we be more explicit and communicate more clearly? The first step is to use better names.

All the Names We Need

Why do people use terse, ambiguous, cryptic names for things? Many reasons. In our day-to-day language we tend to use short names and abbreviations to save time and effort, relying on context to remove ambiguity. Unfortunately, context can fade with time and distance. There are also practical limitations: short names fit better into dense columns, and on legacy computer systems memory and storage were precious. But today storage is cheap, and better software can autocomplete values and display short names while storing longer, unambiguous names.

You're already familiar with a simple yet powerful technology that can provide us with a practically unlimited number of unambiguous names. Here's an example: <https://github.com/jamesaovertton/obo-tutorial/>. It's a [URL](#), also known as a Uniform Resource Locator or a web address, and it's one of the building blocks of the World Wide Web.

A URL has three main parts:

1. protocol, e.g. [https](#)
2. host, e.g. [github.com](#)
3. path, e.g. [jamesaoverton/obo-tutorial/](#)

The protocol tells you how to get the resource. Common protocols for web pages are [http](#) (HyperText Transfer Protocol) and [https](#) (HTTP Secure). The host is the name of the server to contact, which can be a numeric [IP address](#), but is more often a [domain name](#). The path is the name of the resource on that server.

URLs are combine three very important features: they're consistent enough to be shared across the world, almost anyone can make their own, but they're flexible enough to provide a practically unlimited number of names. The consistency comes from (1) the standardized protocols, such as HTTP. DNS names and IP addresses (2) are allocated to organizations and people, and there's a lot of them, allowing almost anyone to have their own "namespace". Between (1) and (2) we have standard ways to find the server that hosts that namespace and connect to it across the Internet. And within each namespace, the path (3) can be almost anything, allowing for each namespace to contain a vast number of names.

What can we do with all these names? We can give a globally unique name to anything we care to talk about. We can give our study a globally unique name. We can name each subject in our study. We can name each row of our table of data (each observation). We can name the relationships that our columns express. We can give a name to the values that we reuse. We don't have to assign a name to everything, but we have more than enough names for everything that we *want* to name.

Names for Rules for Names

There's actually a family of related standards for these names. Related to the URL is the [URN](#) (Universal Resource Name) which has a somewhat different structure. A [URI](#) (Universal Resource Identifier) is either a URL or a URN. URIs are limited to [ASCII](#) characters (for the English alphabet), but an [IRI](#) (Internationalized Resource Identifier) can have [Unicode](#) characters (for almost every language). Because "IRI" is the most general term, that's the one I'll use in what follows, but pretty much every example mentioned is just a humble URL.

Because IRIs can be long and redundant, there are standard ways for shortening them without losing information. A [CURIE](#) (Compact URI) consists of a prefix and a suffix, where the prefix stands in for a longer base IRI. By converting the prefix and appending the suffix we get back to the full IRI. For example, if we let the `obo` prefix stand in for the IRI <http://purl.obolibrary.org/obo/>, then the CURIE `obo:OBI_0000070` can be expanded unambiguously to http://purl.obolibrary.org/obo/OBI_0000070. Any file that contains CURIES should also define the prefixes. Here are some very common prefixes for our work:

- rdf <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
- rdfs <http://www.w3.org/2000/01/rdf-schema#>
- xsd <http://www.w3.org/2001/XMLSchema#>
- owl <http://www.w3.org/2002/07/owl#>
- obo <http://purl.obolibrary.org/obo/>

The part after the prefix is technically known as the “reference”.

This tutorial is about OBO ontologies, and we’ll have more to say about them in the next section. But OBO has its own [policy](#) for naming terms that’s worth discussing now. Every OBO ontology has its own “ID space”, which is a string of a few letters, e.g. “OBI” for the Ontology for Biomedical Investigations. Every term has a unique “Local ID”, which is a string of digits, e.g. “0000070” for OBI’s “assay” term. The combination is the term’s identifier, e.g. “OBI:0000070”. This is what we mean by the ID of an OBO term. Add the <http://purl.obolibrary.org/obo/> prefix (and use an underscore) and you get the term’s unique IRI: http://purl.obolibrary.org/obo/OBI_0000070.

Using numeric IDs for terms has advantages and disadvantages. The biggest and most obvious disadvantage is that the term ID is *opaque*: it carries no meaning for a human reader. There’s nothing about the ID that tells you that it’s the ID for “assay” – at this point I’ve just memorized that connection. But the opacity of the ID becomes an advantage if we ever have to replace the OBI “assay” term with a better term. The new “assay” term will get a new ID and the old term can keep its ID to support legacy data – we try our best not to break legacy systems! If the term ID were not opaque but used a word, something like “OBI_assay”, what would we call the new term? “OBI_assay_new”? And what if we had to replace it, in turn? “OBI_assay_new_2”? My bet is that people would keep using the old “OBI_assay” ID just because it looks right, and never update to the improved terms.

Opaque IDs make versioning easier, and versioning is crucial. We can overcome the disadvantages of opacity with better tools. It’s trivial for software to go from a unique ID to a human-readable label. But trying to go from an ambiguous, human-readable label to a unique ID is a recipe for disaster. Better to use globally unique IRIs and improve our tools.

One other acronym you might come across is “[PURL](#)” (Persistent URL). A PURL is just a URL, but with additional expectations that it will always point to the same resource. PURLs usually work by transparently redirecting a request to another address. If a resource is moved to a new server, for example, the PURL doesn’t change, but it will redirect you to the new address instead of the old one. (In general, [cool URIs don’t change](#), and every IRI should be persistent, but people use PURLs to make their intentions clear.)

It’s important to know that there are some differences, but for pretty much all our purposes we can treat these as the same: URL, URI, IRI, and PURL for full names; CURIE or OBO term ID for abbreviations. I’ll try to stick to “IRI”.

No More than One Name

IRIs let us assign as many names as we want. But it's important to step back and ask: how many names *do* we want?

Names are most useful when everything we want to talk about has only one name. When you and I both use the same name for the same thing, then we're communicating. If we use different names for the same thing, then it takes extra work to figure out that we both mean the same thing.

Coordinating names can be a challenge. The challenge only increases when the community of communicators is large, and our knowledge of the things we're talking about is changing quickly. I'm sure you'll recognize this situation in your field of research: new terminology is introduced, revised and revised again to fit new facts and usage, and then perhaps settles into some stable textbook meaning or disappears all together.

[Ontologies](#) are part of the solution.

Open Biological and Biomedical Ontologies

One of the things that ontologies do is provide systems of names. The Open Biological and Biomedical Ontologies (OBO) is a community of ontology projects for science with shared principles and practises. Most OBO ontologies^[1] provide standardized names for the things that are general within a scientific domain. The term `obo:OBI_0000070` is an example: it names the class of all assays in general. The observation described in the first row of [data-before.csv](#) is a *particular* assay that happened at a certain place and time. The class of all assays is *general* (or *universal*) – it does not exist in space and time, but we can give it a name and talk about it.

Much of the work in science involves general classes. For example, the [Gene Ontology \(GO\)](#) provides terms for universals such as [apoptotic process](#), which is a kind of [biological process](#), and for [host cell membrane](#), which is a kind of [cellular component](#). The [Uberon](#) cross-species anatomy ontology provides terms for universals such as [lung](#), which is a kind of [anatomical entity](#). The [NCBI Taxonomy](#) is not an ontology (strictly speaking), but an OBO-compatible resource is available that provides terms for biological taxa such as [Mus musculus](#).

We can give a name (an IRI) to the subject of the first row of `data-before.csv`, a particular mouse, and assign it to the class of all mice. We can name its particular lung that was used in the experiment and assign it to the class of all lungs (across species), even if we don't know whether the data refers to the right or the left lung.

In addition to names, ontologies also encode information about relationships between classes and facts that hold for all members of a class. This lets us

draw additional inferences about our particular mouse and its particular lung, multiplying the value of our data.

Reference Ontologies and Application Ontologies

We should distinguish between *reference ontologies* and *application ontologies*. A reference ontology is meant to serve a general purpose, providing terms for some domain of science. We can compare it to a textbook. An application ontology is meant to serve a specific purpose, limited to a particular project or institution, but often crossing several domains. An application ontology is likely to include terms from several reference ontologies. Reference ontologies focus almost exclusively on universals, while applications ontologies are more likely to include terms for particulars.

OBO ontologies are reference ontologies – they focus on terms for the universals within a scientific domain. In our running example we will be using terms from several reference ontologies, such as Uberon and OBI. In this tutorial we will be building an application ontology to support our running example, importing these reference ontology terms and adding other terms that don't exist in (or belong in) any reference ontology.

Shared Best Practises

OBO projects all share a commitment to a set of principles and best practises.

TODO: Write a summary of the shared principles and practises. In the meantime, you can peruse these links.

- [The OBO Foundry](#)
- [OBO Foundry Principles](#)
- [OBO Foundry Operations Committee](#)
 - [Editorial Working Group](#)
 - [Technical Working Group](#)
 - [Outreach Working Group](#)

Now let's see how to [use and reuse ontology terms](#).

Using and Reusing Ontology Terms

IRIs give us all the globally unique names we need. Ontologies give us systems of standardized names (and more). Now let's see how to use them.

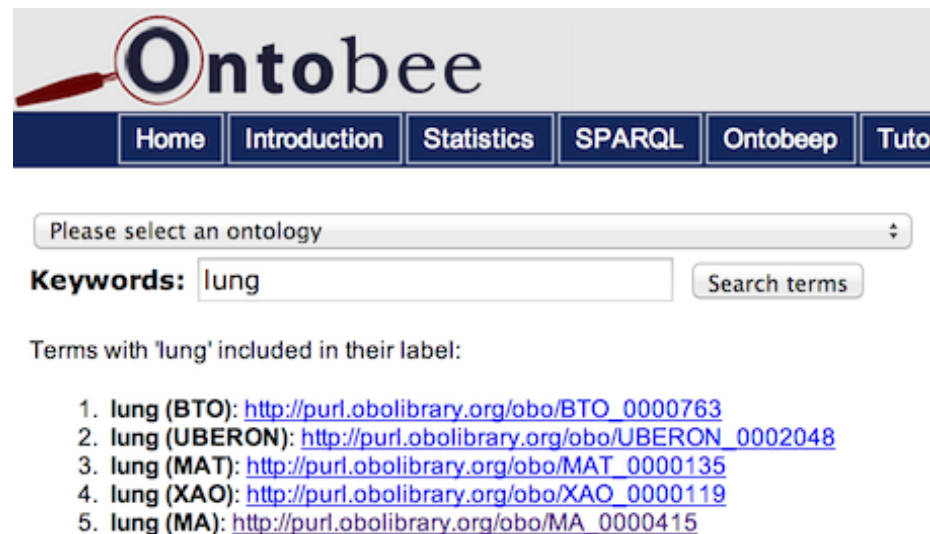
“Using” an OBO reference ontology term means using its IRI in your data. Every term should have just one IRI – don’t make up your own name for it, use the official one! In this section you’ll learn how to find the terms you want, how to assess whether they’re high-quality, and how to import them into the application ontology for your project. We’ll illustrate these techniques by building an application ontology to support our running example.

Finding Ontology Terms

OBO ontologies are freely distributed, and there’s a number of different web sites you can use to find them. We usually begin by searching for an appropriate label, and then check to make sure that the textual and logical definitions fit what we need.

Ontobee

[Ontobee](#) lists all the OBO ontologies and provides useful search and analysis tools. Many OBO projects use it as the endpoint for their PURLS – if you click http://purl.obolibrary.org/obo/OBI_0000070 you’ll be taken to the Ontobee page for OBI’s “assay” term. Ontobee also provides information on how each term is used in other OBO ontologies. Ontobee term pages provide RDF/OWL data that can be used by software. I use Ontobee all the time for finding OBO terms, either within an ontology that I know or across all the OBO projects.



The screenshot shows the Ontobee web interface. At the top is the Ontobee logo, which consists of a magnifying glass icon over the word "Ontobee". Below the logo is a navigation bar with buttons for "Home", "Introduction", "Statistics", "SPARQL", "Ontobee", and "Tuto". Below the navigation bar is a search form. It includes a dropdown menu labeled "Please select an ontology" with a downward arrow. Below that is a "Keywords:" label followed by a text input field containing the word "lung". To the right of the input field is a "Search terms" button. Below the search form, the text "Terms with 'lung' included in their label:" is displayed. Below this text is a list of five results, each numbered and showing the term "lung" followed by the ontology name and its PURL:


1. lung (BTO): http://purl.obolibrary.org/obo/BTO_0000763
2. lung (UBERON): http://purl.obolibrary.org/obo/UBERON_0002048
3. lung (MAT): http://purl.obolibrary.org/obo/MAT_0000135
4. lung (XAO): http://purl.obolibrary.org/obo/XAO_0000119
5. lung (MA): http://purl.obolibrary.org/obo/MA_0000415

BioPortal

[BioPortal](#) provides access to OBO ontologies and many more ontologies and terminologies for biology and medicine. If you can't find the term you're looking for using Ontobee, maybe you can find it using BioPortal. If the term you want belongs to an ontology that is not part of OBO, you might still be able to use it. See the section below on assessing ontologies for reuse.



Welcome to BioPortal, the world's m

For help using BioPortal, click on this icon: 

Search all ontologies

Search

[Advanced Search](#)

OntoMaton

[OntoMaton](#) is a useful plugin for [Google Spreadsheets](#). It uses BioPortal to find and autocomplete terms in selected columns of your sheet. You can also configure it to use only the ontologies you prefer, such as only OBO ontologies. Restrict to the smallest set of ontologies you can to improve search speed. Use the term links to check their definitions! When I have a list of terms I want to map, it saves to setup a new Google spreadsheet with OntoMaton configured to the ontologies I want to use.

The screenshot shows a Google Sheet titled "Term Mappings" with three columns: "Local Term Label", "Ontology Term Label", and "Ontology Term IRI". The data is as follows:

	A	B	C
	Local Term Label	Ontology Term Label	Ontology Term IRI
2	RAT	Rattus norvegicus	http://purl.obolibrary.org/obo/NCBITaxon_10116
3	MOUSE	Mus musculus	http://purl.obolibrary.org/obo/NCBITaxon_10090
4	F 344/N		
5	B6C3F1		
6	FEMALE	female	http://purl.obolibrary.org/obo/PATO_0000383
7	MALE	male	http://purl.obolibrary.org/obo/PATO_0000384
8	HISTOLOGY		
9	LUNG	lung	http://purl.obolibrary.org/obo/UBERON_0002048
10	NOSE	external nose	http://purl.obolibrary.org/obo/UBERON_0007827
11	ADRENAL CORTEX		
12	ADENOCARCINOMA		
13	INFLAMMATION		
14	NECROSIS		
15	SEVERE		
16	MILD		
17	MODERATE		

The "OntoMaton" sidebar on the right shows a search for "adrenal cortex" in the "UBERON - Uber Anatomy Ontology". The search results show "adrenal cortex", "adrenal gland cortex zone", and "cortex" with "Insert Term" buttons next to each.

Finding Ontologies

Ontobee lists all the OBO ontologies and a few more, but the official list is at the OBO home page: <http://obofoundry.org>. That list distinguishes between “OBO Foundry” ontologies that have been reviewed for their adherence to the OBO Foundry principles, and “OBO Library” ontologies that have made a commitment to the principles but have not been reviewed. For each entry there’s a brief description of the domain, plus links to information pages and ontology files.

Several of the OBO principles and best practises are based on open source software development best practises. These include keeping files in publicly accessible version control systems that track and store every change to the ontology. Most OBO projects use one of these websites for hosting project files in version control:

- [GitHub](#)
- [Google Code](#)
- [SourceForge](#)

GitHub is built around the [Git](#) distributed version control system. Google Code and Sourceforge also support Git, but more often you’ll see project using [Subversion \(SVN\)](#). Version control systems can be intimidating at first, but there are tutorials and applications to make life easier for beginners. If you’re an absolute beginner, don’t worry! You can probably do everything you’ll need to do with just your web browser.

- Subversion’s official documentation is a very good [book](#), free online and available in dead tree copies

- GitHub has a great [interactive Git tutorial](#)
- If you're comfortable with the command line, every platform has official tools for SVN and Git
- If you already use an Integrated Development Environment (IDE) of some sort, it's very likely to have Git and SVN support built in
- On Windows, most people seem to use [TortoiseSVN](#) as a graphical SVN client
- For Mac it's more difficult to recommend a graphical SVN client: [Versions](#) is pretty but not free; [SvnX](#) is free but less pretty
- [SourceTree](#) is a powerful, free, application for working with Git, available for Windows and Mac
- GitHub also has free graphical applications for Git on [Windows](#) and [Mac](#)

OBO projects use the [Web Ontology Language \(OWL\)](#) and are distributed in OWL files. You can use the Ontobee and BioPortal websites to browse ontologies, but if you want to download the OWL file and work with it on your computer then you probably want to use [Protégé](#). Protégé is free, open source, and cross-platform.

Assessing Ontologies and Terms for Reuse

Once you find a term that might be suitable, there are some things that you should check before you start using it. The OBO principles are intended to cover all these bases, so when using a term from an OBO ontology it should be quick to check all these boxes. With projects outside the OBO community you will have to check more carefully.

License

The first question is: can I use this term in my application ontology? This raises a series of questions about copyright.

OBO ontologies are intended to be shared and required to be [open](#), in the sense of “open source”. The [Creative Commons CC-by 3.0](#) license is recommended for OBO projects, although there's some variation. We expect people who use OBO ontology terms to use the official IRI for that term. We also expect users to respect common standards in the scientific community for giving credit to ontology developers for their work, such as citing the ontology in your publications. (Most ontologies have a “release paper” in a journal that you can use for citations. Check the ontology's website.)

PURLs

Every term should have a single, official IRI that identifies it, often called a “PURL”. When you put that IRI into a web browser you should be directed to a useful webpage or file. Even IRIs for obsolete terms should continue to be directed to something useful. If the term you want to use doesn’t have a reliable, permanent identifier, than how can you be sure that you and the rest of the community are using the same term?

There is an OBO principle for [unique identifiers](#) and a [identifier policy](#) with the technical details.

Labels

Every term should have a single, official label. It can have many synonyms, translations, and alternative terms, but there should be a just one official label. It should probably be in American English. Ideally, it will be unambiguous across all of biology and biomedicine, and avoid ambiguities with common words in English. It shouldn’t contain jargon or acronyms. Taking all of this into account, many ontology labels are long and ugly, but at least they’re clear. You can always add a shorter synonym or alternative term.

Labels are important, but the definition is even more important, so don’t stop at reading the label!

OBO ontology terms usually have an `rdfs:label` and an “editor preferred term” (annotation using `obo:IAO_0000111`) with the same string. There are also [naming conventions](#) for OBO terms, and an [IAO ontology metadata](#) page.

Textual Definitions

Does the term have a textual definition?

There’s a surprising number of “ontologies” out there with terms that have IRIs and labels but no textual definitions at all. There’s a simple reason for that: good textual definitions are hard to write! And when you’ve written a clear and concise textual definition, more often than not you’ll find that your collaborators say “That’s not how we use ‘X’”. It’s not until you have a definition that you can really start to discuss the meaning of a term, and build some consensus. Without a good definition it’s hard to be sure that everyone understands the term in the same way.

An “annotation” is a bit of data attached to a term that doesn’t have logical structure – we’ll talk about logic in just a second. [Labels](#) and [textual definitions](#) are the most important annotations, but terms are even more valuable when they have examples of usage, editor names, editor notes, synonyms and alternative terms.

Logical Definitions

Does the term have a logical definition?

With very few exceptions, every class should have a parent class. OBO ontologies usually extend the [Basic Formal Ontology \(BFO\)](#) which has just one term without a parent: “[entity](#)”. Knowing a term’s parent (i.e. supertype) tells you a lot about that term, because the child term is defined in part by the parent term.

Most ontologies use the Web Ontology Language, which provides a range of powerful tools for make logical statements about terms and their relationships. Well-defined logical relations allow for automated reasoning over the ontology and data that uses it. The richer the logical definitions, the more work the reasoner can do, although there’s a balance to strike between power and performance.

If the term you want to use fits into the upper ontology and other relations that you’re already using, then you’re enriching the network of data in your application ontology. If the term doesn’t fit, then you might not be adding much by using it. OBO ontologies strive to form a rich, interconnected network.

Community

Who made this ontology? When was it made? Has it been updated since its initial release? Is it ever going to be updated again?

Several related OBO principles require that an ontology have a community of [users](#) who [collaborate](#) on the ontology, [some person or group or organization in charge](#), and a commitment to ongoing [maintenance](#). Too often I’ve found an ontology that passes all the other criteria, but it was developed by a few authors to support a single project five years ago, and it hasn’t been updated since. Building an ontology and maintaining it is hard. Building a community and maintaining it is even harder. But this is another thing that OBO projects strive to do.

Mapping Terms

We’ve just seen how to find OBO ontologies and terms, and how to assess their quality. Now let’s put that into practise.

Starting with data like our [data-before.csv](#) spreadsheet, my first step is usually to create a list of all the “local” terms I want to “map” to reference ontology terms. I put this list in a spreadsheet. Then I use Ontobee and OntoMaton to find one ontology term for each local term. Usually I can find one, with a little work, but sometimes I can’t. My third step is to decide how to get each term into my application ontology. There are four main techniques that I use, each described in the next section:

1. if there's just a few terms from the reference ontology, I'll MIREOT them using OntoFox
2. if I want to use many terms from a small reference ontology, I'll import the whole thing using Protégé
3. if there are many terms from the reference ontology or the reference ontology has complex logical axioms, but I don't want to import the whole thing, then I'll extract the terms I need using OWLAPI
4. if there's no reference ontology term, then I'll define the term for my application ontology using QTT or in Protégé

So those are my three steps:

1. make a list of local terms
2. search for reference ontology terms
3. decide how to import each term into my application ontology

This [Google spreadsheet](#) shows the list of terms from [data-before.csv](#), the reference ontology terms I want to map them too, and the import technique I will use. It also shows how you can use the OntoMaton plugin to make searching easier.

Importing Terms

Once you've found the perfect term, how do you use get it in to your application ontology? What if you can't find the perfect term? In either case, there's a range of technique that you can use.

Importing Single Terms with MIREOT

Using an ontology term means using its IRI, but you will also want to keep its label, definition, and other annotations. [MIREOT: The minimum information to reference an external ontology term](#) is a guideline for what information to include when importing a term. We even use "MIREOT" as a verb: "You should MIREOT that term into your ontology."

When you MIREOT a term, you don't have to include its parent terms or logical axioms, but if you don't then you have to be careful. As an example, the term [Homo sapiens](#) is about 29 nodes deep in the NCBI Taxonomy. If you want to use that term, you probably don't want to import all its ancestors. But you do have to ensure that everything you say about that term is still true! You could place *Homo sapiens* under [Mammalia](#) in your application ontology, which would be true, even if it doesn't include all the information that's in the NCBI Taxonomy. Don't place it under [Mus musculus](#)!

Importing Sets of Terms Using OntoFox [OntoFox](#) is a convenient tool for MIREOTing terms from OBO ontologies. You can use the form on the web site, or you can create an input file that specifies the configuration and upload it. The configuration file is useful when you want to run the same import repeatedly, to get the latest versions of the terms, for instance.

OntoFox is a free service, but it has limited resources. If you aren't careful, you can request too many terms. If this happens then the server can hang or crash, and not only won't you get the result you wanted but other people won't be able to use the service. I recommend using OntoFox only for small sets of terms, say less than a hundred, including intermediate terms discussed below.

To use OntoFox you need to select a source ontology, then provide two sets of terms that you want to import: low level terms and top level terms. You will also have to choose whether you want to include *intermediate* terms. You have three options:

- **includeNoIntermediates**: only the terms you specified will be included
- **includeAllIntermediates**: all the terms between low level and top level will be included
- **includeComputedIntermediates**: a minimal set of terms connecting the low level to the top level will be included

Another option is to include all the children of a given low level term. These options can be a little confusing at first, and it's worthwhile to experiment with a few different combinations. For large ontologies such as the NCBI Taxonomy, **includeNoIntermediates** is probably the best choice. Otherwise **includeComputedIntermediates** strikes a good balance.

Beware that including too many intermediate terms can cause problems for the OntoFox server. When you're experimenting with including intermediates, make sure that there's only a few levels of hierarchy between "low" and "top", and start with just a few of the terms you want to import.

OntoFox also provide a range of options for selecting which annotations you want to include, and for mapping between sets of annotations. Although the form on the web site only allows extracting terms from one ontology at a time, in a configuration file you can specify several ontologies each with a number of terms and other settings. This is another reason to learn how to use the [input configuration files](#).

The [ontofox.txt](#) file shows what an OntoFox input file looks like. It has four sections because we're importing from four ontologies: OBI, NCBI Taxonomy, MPATH, and PATO. It lists the low level and top level terms we want to import from each reference ontology. Each section has a different rule for selecting intermediates, but all of them use **includeAllAxiomsRecursively** to get the full set of annotations. It also shows how to use **subClassOf** to place imported terms exactly where you want them.

On the OntoFox website you can “Upload input file” and with `ontofox.txt` then click “Get OWL (RDF/XML) Output File”. The resulting OWL file is [ontofox.owl](#).

Importing Lists of Terms with Quick Term Templates

The MIREOT technique and OntoFox tool are the right choice for importing a limited number terms from other ontologies. For creating your own terms, Protégé is a good tool. But if you find yourself creating more than a few terms with very similar structure, you might wish that you had a spreadsheet. In that case, consider using the [Quick Term Template \(QTT\)](#) method.

QTT is a technique for transforming tables of data about terms into OWL format. In the table you have a row for each term with columns for the IRI, the label, the definition, and other annotations. You can also have columns with IRIs for other terms that have a well-defined relationship to the term in that row, such as the parent term. This is the input to the QTT tool, and the output is an OWL representation of the same data ready to be used in your ontology.

The `qtt.txt` file is a tab-separated spreadsheet with the two local terms we want to define for our example: strains of inbred rats and mice. These terms should probably be defined in some OBO ontology but I haven’t managed to find them. It suits our current purposes to use them as an example of application ontology terms. As you can see, I’ve given the term ID in the first column, then annotations such as the label, definition, and editor. I’ve also given the IRI of the parent term.

Older versions of Protégé supported the [MappingMaster](#) plugin for QTT, but it is no longer maintained. If you know how to write Java code and use OWLAPI then it’s straightforward to write your own QTT code. But the easiest solution is to use Ontorat.

Importing Lists of Terms Using Ontorat [Ontorat](#) is another tool in the same family as Ontobee and OntoFox. As with OntoFox, you can use the form on the web site or upload a configuration file. And as with OntoFox, the configuration file has the advantage of being repeatable and providing more powerful options, at the cost of a slightly steeper learning curve.

The `ontorat.txt` file shows what an Ontorat input file looks like. We’re only using sections for annotations and superclasses, but there are other options. Read the [tutorial](#) to get all the details, but the basic operation is to create a new class for each row in the QTT table, then create annotations and axioms by replacing every occurrence of `{columnX}` with the contents of column X.

- single quotes are human-readable names for terms, and they should be defined with an IRI for that term

- double quotes are for annotation strings
- angle brackets are for IRIs

On the Ontorat website you can “Load Settings File” with this file to fill in the settings, then “Specify input data file” with a “File upload”, and use `qtt.txt`. Then click “Get OWL (RDF/XML) Output File”. The resulting OWL file is [ontorat.owl](#). I changed the “Ontology IRI” for this file using Protégé to something more sensible than the default.

Importing Ontologies with Protégé

OBO ontologies are part of the web of linked data. Using terms from other ontologies means making links to resources across the web. Another form of linking is *import*: you can import another complete ontology or OWL file into your own ontology.

For our current purposes, there are two main uses for OWL imports. The first is when MIREOT just isn’t enough: you don’t want to import just a selection of terms from another ontology, you want to import the whole thing. The second is using the results of OntoFox and Ontorat. These tools generate OWL files. Using a configuration file, you can run these tools again and again to update terms, generating new OWL files each time. You should keep those results in separate OWL files, then import those files into your application ontology. This clean separation makes it easy to update your MIREOT and QTT imports.

Keep in mind that imports can be somewhat unpredictable. Tools such as Protégé will keep a local copy of imported files, which might not always be up-to-date. If you and your collaborators have different local copies of the imports, you won’t see exactly the same terms and axioms, which can lead to all sorts of confusion!

OBO ontologies have versions, usually specified by date. I recommend importing a specific, dated version of any ontology. To update it, just change the import link to the latest dated version. This allows you to control when updates are made. And if the other ontology releases a new version that breaks something in your ontology, you’ll be able to stick to the older version until the problem is fixed.

For OntoFox or Ontorat OWL files, just use the latest OWL files. You control when you update those files, so there shouldn’t be any problem with mismatching versions.

Another thing to beware of is the `catalog.xml` (aka `catalog-v001.xml`) files used by Protégé and OWLAPI. These are configuration files that Protégé automatically generates, and that contain information about imports. By editing these files you can change what files Protégé uses for imports, performing all sorts of redirections and substitutions. This can be very convenient and *very* confusing.

While sometimes necessary, be very cautious when using `catalog.xml` files. This page explains the [details of OWL imports in Protégé](#).

To import an ontology, first open your application ontology with Protégé. On the “Active Ontology” tab there should be a pane called “Ontology imports”. Look for “Direct Imports” and click the plus sign next to it. This opens the “Import ontology wizard” which will give you a range of choices. You can either import from a local file or from a URI on the web. For reference ontologies I suggest using a URI. Protégé will periodically fetch a new copy of the file, so remember to use dated version whenever possible. For OntoFox and Ontorat results I tell the import wizard to use a local file.

Here’s a screenshot from the latest Protégé 5 beta release importing the 2014-08-18 version of OBI.



Extracting Ontology Modules with OWLAPI

Sometimes you want to import too many terms for OntoFox, but the target ontology is too large to import all of it. There’s a tool you can use in this middle case: the OWLAPI’s [SyntacticLocalityModuleExtractor](#). Given an ontology and a set of terms to extract, it will create a “module” with just those terms and the terms they depend on. These dependencies are recursive, so the tool will take your list of terms, add all the terms they depend on, then add all the terms those dependencies depend on, and so on until it finds a closed set of terms. The resulting ontology can be saved to an OWL file and imported into your application ontology.

Using this technique requires using OWLAPI, which means writing some Java code. There’s no web service to do it for you. I’ve provided an example:

- short official example (PDF) [The Rough Guide to the OWL API](#)
- full example: [Extractor.java](#)
- list of terms to extract: [uberon-terms.txt](#)
- sample command – make sure you download [Uberon](#) first (it’s about 60MB):

```
java -jar bin/obo-tutorial.jar extract \  
  examples/uberon.owl \  
  examples/uberon-terms.txt \  
  examples/uberon-module.owl \  
  "https://github.com/jamesaoverton/obo-tutorial/raw/  
    master/examples/uberon-module.owl"
```

This command will extract the terms listed in `uberon-terms.txt` and all their dependencies into the `uberon-module.owl` file. Because Uberon has many logical axioms linking terms, there are many dependencies, and the result is more than 2MB.

Putting it all Together

First we saw how to find reference ontology terms and assess them. Then we saw how to create a “mapping” table for all our terms. And we just saw four techniques for importing reference ontology terms into our application ontology. The last step is to put all of this together and complete our application ontology.

1. Open a new ontology
2. Change its “Ontology IRI” to <https://github.com/jamesaoverton/obo-tutorial/raw/master/example/application.owl>
3. Import three OWL files:
 - `ontofox.owl`
 - `ontorat.owl`
 - `uberon-module.owl`
4. Add any other annotations or terms you like
5. Save the ontology to `application.owl`

Or skip those steps and just look at the resulting `application.owl` file I’ve created. The result includes all the terms listed in our `terms.csv` file, plus the dependencies that we want.

One application ontology can support many similar projects. In the [next section](#) we’ll see how to connect the data in the running example to the application ontology to take full advantage of it.

Processing Data with Ontologies

TODO: Write this section.

Tables to Triples

Converting Relational Data to RDF

Using SPARQL with OWL Ontologies

Converting Instance Data to RDF/OWL

Querying Instance Data with DL Query and SPARQL

Best Practises for Reasoning Over Large Data Sets

Updating, Testing, and Releasing Ontologies

TODO: Write this section.

Tracking Changes with Ontology Diff Tools

Filing Effective Term Requests and Bug Reports

Issue Trackers

TermGenie

Best Practices for Committing Changes to an Ontology

Quality Checking for Ontologies

Unit Testing for Ontologies

Continuous Integration with Jenkins

Release Workflows with OWLTools

Publishing Ontologies

Managing PURLs