How to plan data ingest for VIVO

Error rendering macro 'toc'

null

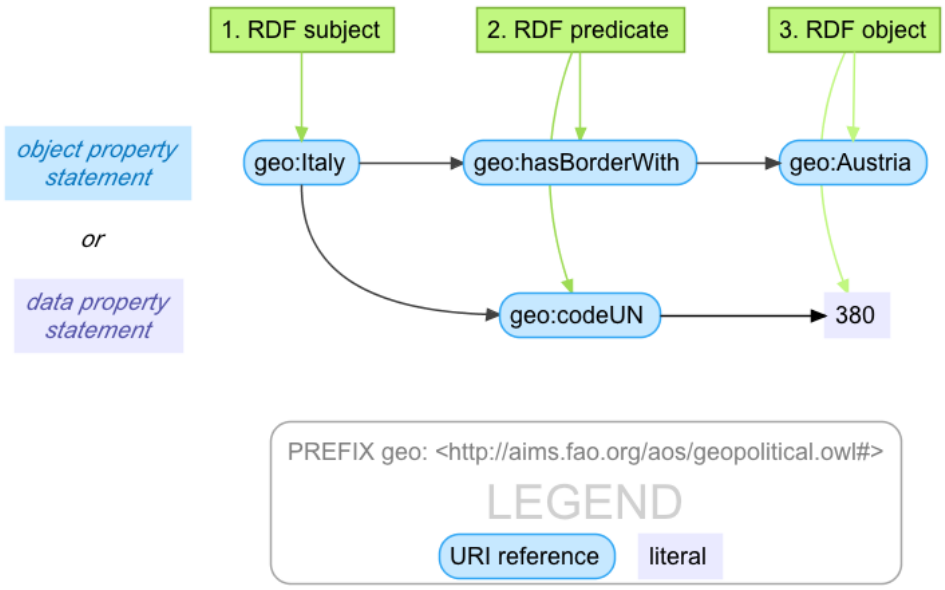
How VIVO differs from a spreadsheet

VIVO stores data as RDF individuals – entities that are instances of OWL classes and that relate to each other through OWL object properties and have attributes represented as OWL datatype property statements. Put very simply,

- Classes are types – Person, Event, Book, Organization
- Individuals are instances of types – Joe DiMaggio, the 2014 AAAS Conference, Origin of Species, or the National Science Foundation
- Object properties express relationships between to individual entities, whether of the same or different types – a book has a chapter, a person attends an event
- Datatype properties (data properties for short) express simple attribute relationships for one individual – a time, date, short string, or full page of text

Every class, property, and individual has a URI that serves as an identifier but is also resolvable on the Web as Linked Data.

A triple in RDF has a subject, a predicate, and an object – think back to sentence diagramming in junior high school if you go back that far.



So far all this would fit in a spreadsheet – one row per statement, but never more than 3 columns.

This may not be the most useful analogy, however, since you can't say very much in a single, 3-part statement and your data will be much more complex than that. A person has a first name, middle name, and lastname; and a title, a position linking them to a department; many research interests; sometimes hundreds of publications, and so on. In a spreadsheet world you can keep adding columns to represent more attributes, but that soon breaks down.

But let's stay simple and say you only want to load basic directory information in VIVO – name, title, department, email address, and phone number.

name	title	department	email	phone
Sally Jones	Department Chair	Entomology	sj24@university.edu	888 777-6666
Ruth Ginsley	Professor	Classics	rbg12@university.edu	888 772-1357
Sam Snead	Therapist	Health Services	ss429@university.edu	888 772-7831

Piece of cake – until you have a person with 2 (or 6) positions (it happens). Or two offices and hence two work phone numbers.

VIVO breaks data apart in chunks of information that belong together in much the same way that relational databases store information about different types of things in different tables. There's no right or wrong way to do it, but VIVO stores the person independently of the position and the department – the position has information a person's title and their beginning and ending date, while the department will be connected to multiple people through their positions but also to grants, courses, and other information.

VIVO even stores a person's detailed name and contact information as a vCard, a W3C standard ontology that itself contains multiple chunks of information. More on this later.

Storing information in small units removes the need to specify how many 'slots' to allow in the data model while also allowing information to be assembled in different ways for different purposes – a familiar concept from the relational database world, but accomplished through an even more granular structure of building blocks – the RDF triple. There are other important differences as well – if you want to learn more, we recommend The Semantic Web for the Working Ontologist, by Dean Allemang and Jim Hendler.

Where VIVO data typically comes from

It's perfectly possible, if laborious, to add all data to VIVO through interactive editing. For a small research institution this may be the preferred method, and many VIVO institutions employ students or staff to add and update information for which no reliable system of record exists. If VIVO has been hooked up to the institutional single sign-on, self-editing by faculty members or researchers has been used effectively, especially if basic information has been populated and the focus of self editing is on populating research interests, teaching statements, professional service, or other more straightforward information.

This approach does not scale well to larger institutions, and full reliance on researchers do editing brings its own problems of training, consistency in data entry, and motivating people to keep content up to date. Many VIVOs are supported through libraries that are more comfortable providing carrots than sticks and want the VIVO outreach message to focus on positive benefits vs. threats about stale content or mandates to enter content for annual reporting purposes.

VIVO is all about sharing local data both locally and globally. Much of the local data typically resides in "systems of record" – formerly entirely locally hosted and often homegrown, but more recently starting to migrate to open source software (e.g, Kuali) or to cloud solutions.

These systems of record are often silos used for a defined set of business purposes such as personnel and payroll, grants administration, course registration and management, an institutional repository, news and communications, event calendar(s), or extension. Even when the same software platform is used, local metadata requirements and functional customizations may make any data source unique.

For this reason, coupled with variations in local technology skills and support environments, the VIVO community has not developed cookie cutter, one-size-fits-all solutions to ingest.

Here's a rough outline of the approach we recommend for people new to VIVO when they start thinking about data:

1. Look at other VIVOs to see what data other people have loaded and how it appears
2. Learn the basics about RDF and the Semantic Web (there's an excellent book) – write 10 lines of RDF
3. Download and install VIVO on a PC or Mac, most easily through the latest VIVO Vagrant instance
4. Add one person, a position, a department, and some keywords using the VIVO interactive editor
5. Export the data as RDF and study it – what did you think you were entering, and how different is the structure that VIVO creates?
6. Add three more people, their positions, and a couple more departments. Try exporting again to get used to the RDF VIVO is creating.
7. Try typing a small RDF file with a few new records, using the same URI as VIVO created if you are repeating the same entity. Load that RDF into VIVO through the Add/Remove RDF command on the Site Admin menu – does it look right? If not, double check your typing.
8. Repeat this with a publication to learn about the Authorship structure – enter by hand, study, hand edit a couple of new records, ingest, test
9. Don't be afraid to start over with a clean database – you are learning, not going directly to production.
10. When you start to feel comfortable, think about what data you want to start with – perhaps people and their positions and titles. Don't start with the most challenging and complex data first.
11. Work with a tool like Karma to learn semantic modeling and produce RDF, however simple, without writing your own code
12. Load the data from Karma into your VIVO – does it look right? If not, look at the differences in the RDF – it may be subtle
13. Then start looking more deeply at the different ingest methods described later in this section of the wiki and elsewhere

Cleaning data prior to loading

If you have experience with data you'll no doubt be familiar with having to clean your data before importing it into software designed to show that data publicly. There will be duplicates, uncontrolled name variants, missing data, mis-spellings, capitalization differences, oddball characters, international differences in names, broken links, and very likely other problems.

VIVO is not smart enough to fix these problems since it in almost all cases has no means of distinguishing an important subtle variation from an error. With the Semantic Web, 'inference' does not refer to the ability to second-guess intent, nor 'reasoning' to invoking artificial intelligence.

Furthermore, it's almost always easier to fix dirty data before it goes into VIVO than to find and fix it after loading.

There's another important consideration – the benefit of fixing data at its source. Putting data into a VIVO makes that data much more discoverable through both searching and browsing. People will find errors that are most likely hidden in the source system of record – but if you fix them in VIVO, they won't be correct in the source system, and the next ingest runs the risk of overwriting changes.

There are many ways to clean data, but in the past few years Open Refine has emerged offering quite remarkable capabilities including the ability to develop changes interactively that can then be saved as scripts to run repeatedly on larger batches.

Matching against data already in VIVO

The first data ingest from a clean slate is the easiest one. After the first time there's a high likelihood that updates will refer to many of the same people, organizations, events, courses, publications, etc. You don't want to introduce duplicates, so you need to check new information against what's already there.

Sometimes this is straightforward because you have a unique identifier to definitively distinguish new data from existing – your institutional identifier for people, but also potentially an ORCID iD, a DOI on a publication, or a FundRef registry identifier.

Often the matching is more challenging – is 'Smith, D' a match with Delores Smith or David Smith? Names will be misspelled, abbreviated, changed over time – and while a lot can be done with relatively simple matching algorithms, this remains an area of research, and of homegrown, ad-hoc solutions embedded in local workflows.

One way to check each new entry is to query VIVO, directly or through a derivative SPARQL endpoint using Fuseki or other tool, to see the identifier or name matches data already in VIVO. As VIVO scales, there can be performance reasons for extracting a list of names of people, organizations, and other common data types for ready reference during an ingest process. It's also sometimes possible to develop an algorithm for URI construction that is reliably repeatable and hence avoids the need to query – for example, if you embed the unique identifier for a person in their VIVO URI through some simple transformation, you should be able to predict a person's URI without querying VIVO. If that person has not yet been added to VIVO, adding them through a later ingest is not a problem as long as the same URI would have been generated. This requires that the identifier embedded in the URI is not private or sensitive (e.g., not the U.S. social security number) and will not change.

One caution here – it's important to think carefully about the default namespace you use for URIs in VIVO if you want linked data requests to work – please see A simple installation and look for the basic settings of the Vitro.defaultNamespace.

Doing further cleanup once in VIVO

VIVO's interactive editing can be helpful in fixing problems in relatively small datasets but it's important to remember that if data originate outside of VIVO and are not corrected at the source, subsequent updates will likely re-introduce the error.

It's also common to have discrepancies in the source data – for example, the naming conventions and identifiers used for departments in a personnel database vs. those used in a grants administration system. There is a command to merge two individuals in VIVO, specifying which URI to retain, but that will combine the statements associated with both, leading to duplicate labels and potentially other duplicates.

VIVO has only limited support for owl:sameAs reasoning due to the performance implications of having to query for all statements about more than one URI whenever rendering information about any one URI declared to be sameAs another.

Many VIVO installations have developed workflows for checking VIVO data, ranging from broken link checkers to nightly SPARQL queries to detect malformed data such as publications without authors, people without identifiers, 'orphaned' dates no longer referenced by any property statements, and so forth. These tools have been discussed on previous Apps and Tools Interest Group calls that have been recorded and uploaded to YouTube.

Ingest tools: home brew or off the shelf?

Major options including the Harvester, semantic ingest tools such as Karma, and XSLT

Typical ingest processes

Alternative approaches to ingest and making ingest repeatable

Challenges for data ingest

Challenges in the data, in workflow, in working incrementally, in modeling, and in migration

Monitoring for quality