Reading From and Writing To Wikidata/Wikibase using Software

Steve Baskauf

Digital Scholarship and Communications

Heard Libraries, Vanderbilt University

Outline

1. Review of terminology
2. Importance of the Wikibase data model
3. Reading using SPARQL queries
4. Writing using the Wikimedia API
5. How we might use Wikibase and Wikidata at Vanderbilt

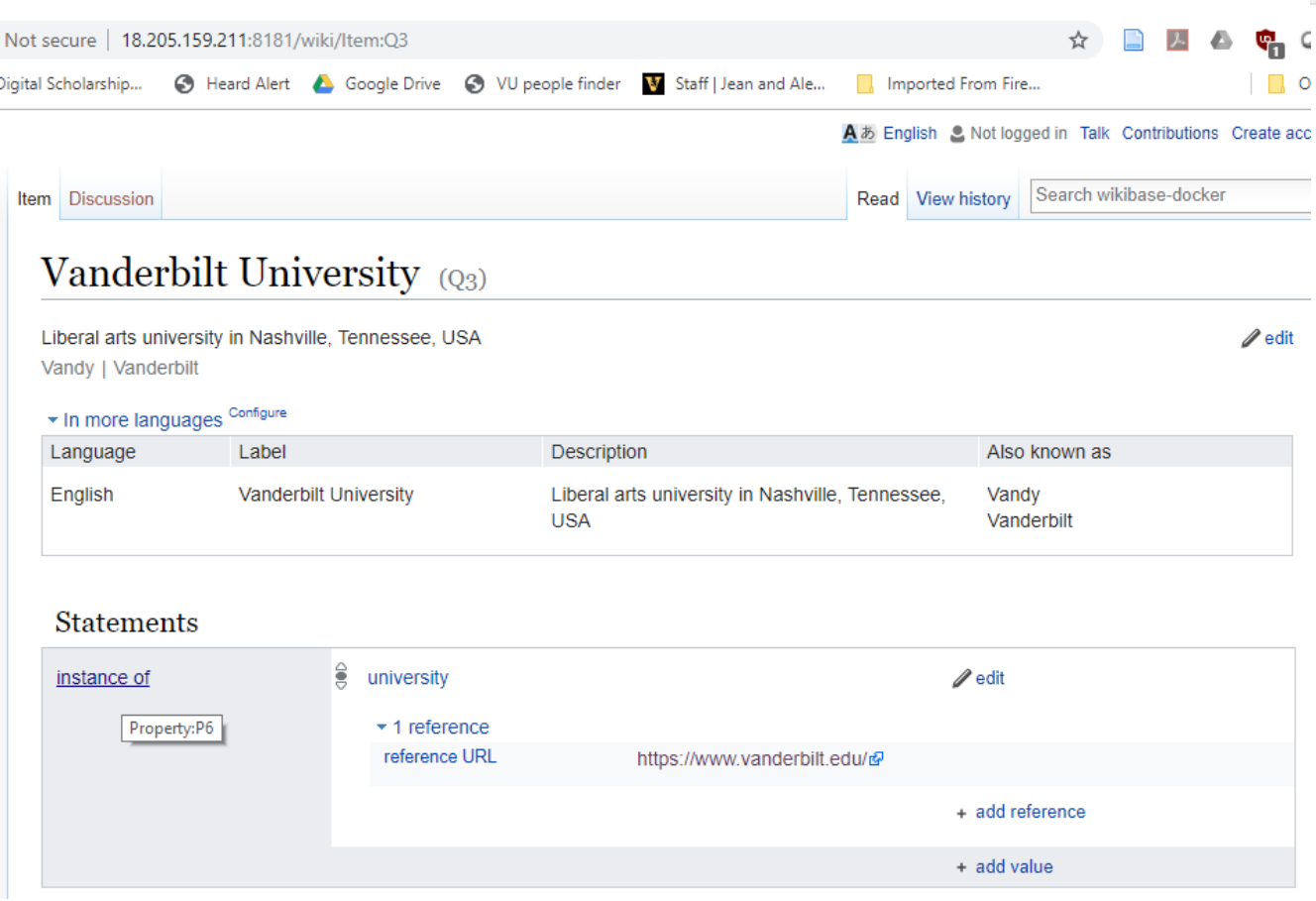
NOTE: this is an overview. For details and DIY instructions, see the blog posts at http://baskauf.blogspot.com/

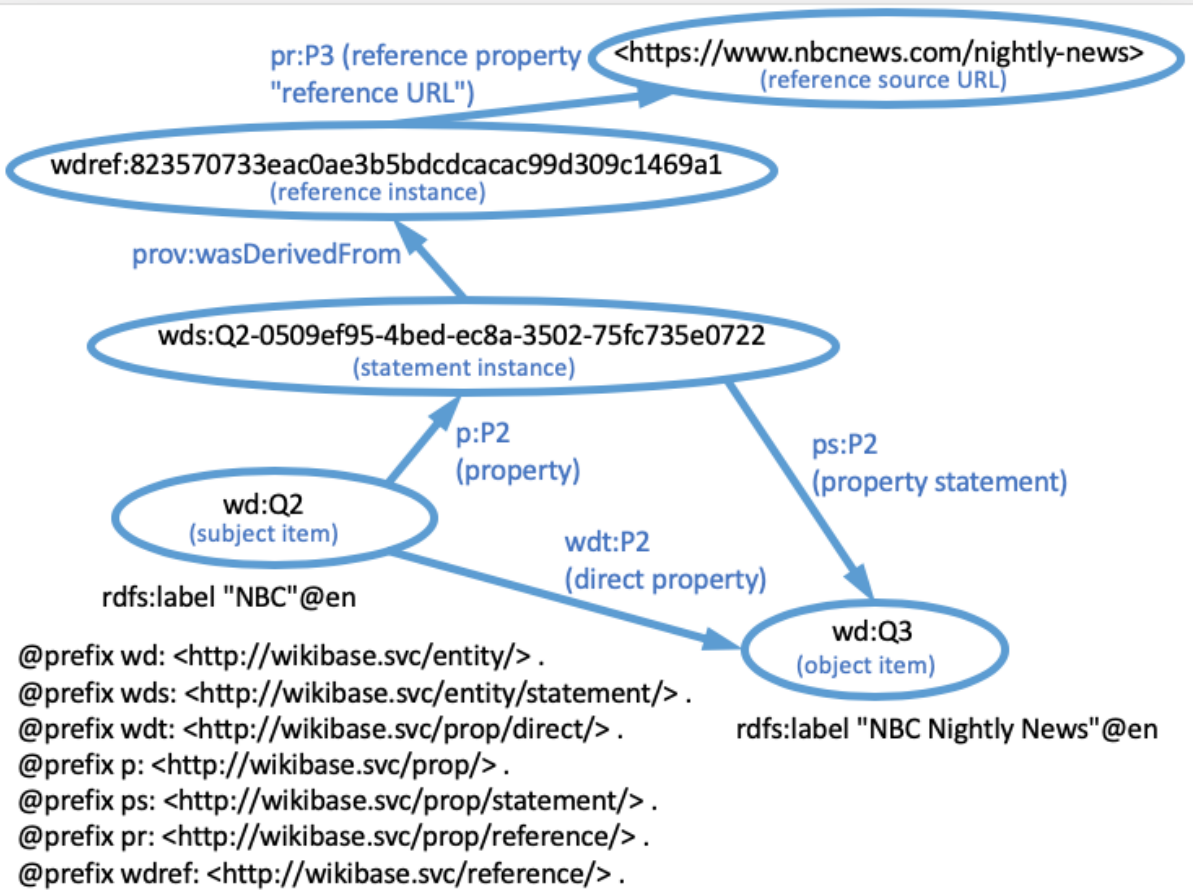
Terminology review: Wikidata vs. Wikibase

- **Wikibase** is a generic platform that can be used by anyone.
  - Wikibase comes "empty", without any items or properties.
  - Wikibase assumes a particular data model based on statements and evidence.
  - You can do anything you want with Wikibase.
- **Wikidata** is a specific implementation of Wikibase.
  - Wikidata properties are developed by community consensus.
  - Wikidata items can be created and described by anyone (subject to notability requirements)
- **Bots** (computer programs) can be used to write to Wikibase instances, including Wikidata
  - You must conform to community standards when using a bot with Wikidata

The Wikibase data model

- The **core data model** is based on items, main statements about items, references about the statements, and statements about the references. It also includes qualifiers and ranks for statements.
- Multilingual **labeling** is assumed in the model.
- Wikibase has **no "built-in" properties** for making the statements, so properties (and their IDs) vary among installations.
- There are key **philosophical differences** between the Wikibase model and traditional models based on RDFS/OWL (particularly no built-in notion of classes).
- The Wikibase model is generic, but can be expressed as an **RDF graph** (therefore allowing SPARQL queries).





Wikibase model as expressed in the GUI

Wikibase model as an RDF graph
(from https://heardlibrary.github.io/digital-scholarship/lod/wikibase/)

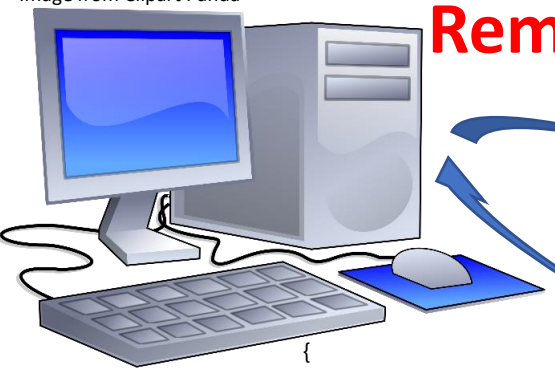
Understanding the Wikibase data model is critical for:

- constructing the queries required to read
- building labels, statements and references when writing

Key aspects of reading from Wikidata (or Wikibase)

- The **Wikidata Query Service** supports **SPARQL** via HTTP. (SPARQL is not the only way to read data, but it's the most versatile.)
- The SPARQL **protocol** uses generic **HTTP**, so can be used in **any** programming **language**.
- Use SPARQL **graph patterns** to screen for **any data** you want (but you must understand the Wikibase graph model).
- SPARQL **Select** queries can return selected **data** as JSON.
- SPARQL **Construct** queries can return **triples** in any serialization.
- **Authentication** is **not required** to use the Wikidata Query Service.
- There is no practical difference between reading from Wikidata and any other Wikibase installation (but properties will differ).

Image from Clipart Panda



Remote client software

querying process (via HTTP)

SPARQL query:

```
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT DISTINCT ?label
WHERE {
 wd:Q29052 rdfs:label ?label.
 }
```

HTTP Request:

GET
https://query.wikidata.org/sparql?query=PREFIX%20wd%3A%20%3Chttp%3A%2F%2Fwww.wikidata.org%2Fentity%2F%3E%0APREFIX%20rdfs%3A%20%3Chttp%3A%2F%2Fwww.w3.org%2F2000%2F01%2Frdf-schema%23%3E%0ASELECT%20DISTINCT%20%3Flabel%0AWHERE%20%7B%0A%20%20wd%3AQ29052%20rdfs%3Alabel%20%3Flabel.%0A%20%20%7D

ACCEPT: application/sparql-results+json

HTTP Response (JSON)


```
{
  "head": {
    "vars": [
      "label"
    ]
  },
  "results": {
    "bindings": [
      {
        "label": {
          "xml:lang": "en",
          "type": "literal",
          "value": "Vanderbilt University"
        }
      },
      {
        "label": {
          "xml:lang": "zh",
          "type": "literal",
          "value": "范德堡大学"
        }
      }
    ]
  }
}
```


"API"

"endpoint"
https://query.wikidata.org/sparql

Wikikdata triple store (graph database)

http://www.wikidata.org/entity/Q29052

rdfs:label → "范德堡大学"@zh-Hans

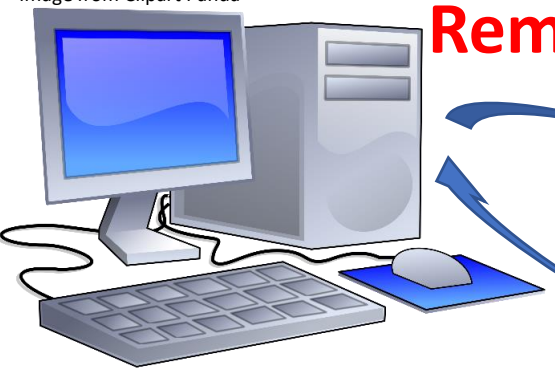
rdfs:label → "Vanderbilt University"@en

Server application (Blazegraph)

Key aspects of writing to Wikibase

- The generic **MediaWiki API** can be used to write to any Wikibase instance (using special Wikibase-specific actions). (Pywikibot an alternative for Python only.)
- API **write** (POST) commands require credentials and **authentication**.
- The API **sandbox** (e.g. https://test.wikidata.org/wiki/Special:ApiSandbox) is a key tool for learning about new API actions.
- Syntax of POST body is idiosyncratic; will **vary** somewhat depending on:
  - whether an item or statement is being created
  - whether properties are assigned to a main statement or reference
  - the kind of value (string literal, item, geocoordinates, etc.)
- Data must be written in an order consistent with the Wikibase model

Image from Clipart Panda



Remote client software writing process (via HTTP)

HTTP Request:

```
POST https://test.wikidata.org
BODY: {
"action":"wbcreateclaim",
"format":"json",
"entity":"Q188427",
"snaktype":"value",
"token": "c9e1a72c00b914f6743d1739ff25d5d65d2de71d+\\",
"property": "P82",
"value":"{\"entity-type\":\"item\",\"numeric-id\":1917}"
}
```

HTTP Response (JSON)


```
{
  "pageinfo": {
    "lastrevid": 517229
  },
  "success": 1,
  "claim": {
    "mainsnak": {
      "snaktype": "value",
      "property": "P82",
      "hash":
"5a4802b3850ccb7ab0c54d1da8c270f75e610
994",
      "datavalue": {
        "value": {
          "entity-type": "item",
          "numeric-id": 1917,
          "id": "Q1917"
        },
        "type": "wikibase-entityid"
      },
      "datatype": "wikibase-item"
    },
    "type": "statement",
    "id": "Q188427$F414A7DB-2D5B-4741-
BF53-CE7FA99F6EF3",
    "rank": "normal"
  }
}
```


Wikimedia API

"endpoint"
https://test.wikidata.org

Wikidata relational database

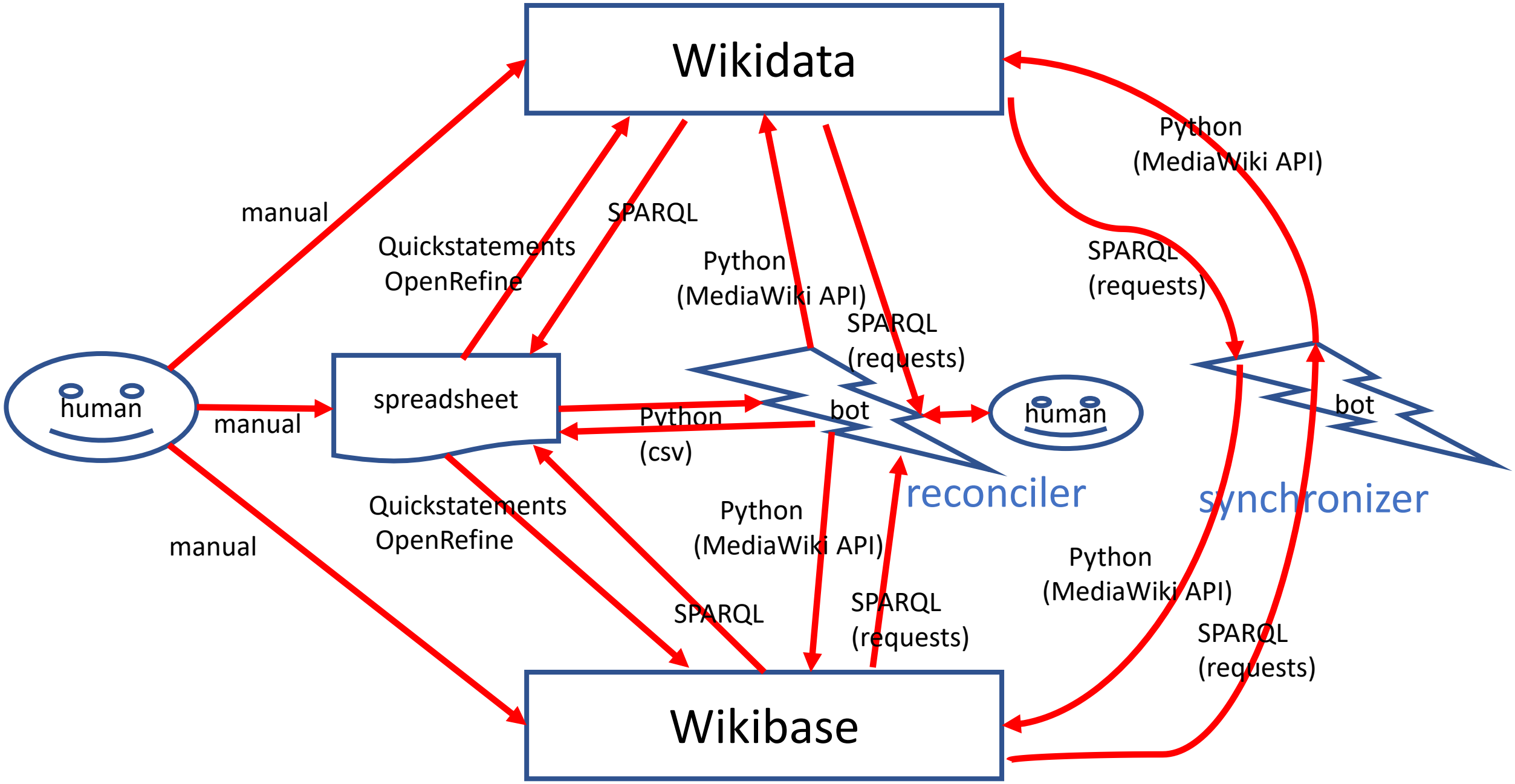
item	property	value
Q188427	P82	Q355
Q188427	P82	Q781
Q188427	P82	Q1917

Key aspects of writing to Wikidata (beyond Wikibase requirements)

- Bots should have a dedicated account.
- Bots should have a defined purpose, vetted by the community.
- Bots should be throttled and process the Maxlag parameter.
- Use https://test.wikidata.org/, NOT the real Wikidata API for testing.
- You should demonstrate that the bot works before production.

Future (possible) projects at Vanderbilt

- This fall, Linked Data Working Group themed broadly on Wikidata.
- Use a local Wikibase installation to create records of faculty and their publications, then synch with Wikidata so the data are accessible via Scholia.
- Use a local Wikibase installation to manage (and possibly crowdsource) metadata related to the Vanderbilt Television News Archive.


Wikidata
manual
Quickstatements
OpenRefine
SPARQL
Python
(MediaWiki API)
SPARQL
(requests)
Python
(MediaWiki API)
SPARQL
(requests)
human
manual
spreadsheet
Python
(csv)
bot
human
bot
reconciler
synchronizer
manual
Quickstatements
OpenRefine
Python
(MediaWiki API)
Python
(MediaWiki API)
SPARQL
SPARQL
(requests)
SPARQL
(requests)
Wikibase

Some questions

- Do we try to duplicate all of Wikidata's properties, or make our own analogs and track the mappings?
- What fraction of faculty publications (from machine-readable sources) could be linked on Wikidata without human interaction?
- Do we automatically accept (and synch) any faculty publication listed in Wikidata or should there be vetting in our local Wikibase?
- Is it even worth having the local installation of Wikibase rather than a relational database or conventional triplestore?