- Source: github.com/ubc-library-rc/dryad2dataverse
- Documentation: ubc-library-rc.github.io/dryad2dataverse/

Speaker notes

- For those who wish to have a look at the source material during the presentation

What is it?

- A stand-alone application
- A Python library

Speaker notes

- Tool(s) to facilitate translation of data from Dryad to Dataverse, from start to finish
- Both a downloadable piece of software and (hopefully) a development tool
- [Presumably] easy to use

Rationale

- Collection consolidation
- Findability

Speaker notes

- UBC's "primary" research data repository is a Dataverse repository at Scholars Portal.
- Other UBC users have deposited their data sets into Dryad; there is a large UBC contingent of data there — over 500 studies
- Collection is split between two (or more, really) repositories: 50% Dryad, 50% Dataverse. That's not ideal.
- Scholars Portal Dataverse is already aligned with Geodisy for national-level geospatial searching
- SP Dataverse is connected to UBC's Summon instance

Problems that can arise

- Except maybe the citation count that researchers now have to check their citations in more than one place, but could be resolved with GraphQL [Eugene]

Why not transfer everything manually?

Speaker notes

- Manual transfer of metadata would be painful and slow
- A software based solution *should* result in less data loss and human error. Or possibly more. However, it's easier to automate. So, software.
- As all material in Dryad is public domain data (CC0) by the terms of their license agreeement, so there's no impediment to adding these data sets to UBC's collection at Scholars Portal.

Desired goals

- Simple
- Modular
- No overhead

Speaker notes

- Simple enough to be used by users with little or ideally no programming experience
- Modular - not all components shold be required
- System neutral
- No requirement of server overhead
- Ideally, a piece of software that would run from the command line with basic information supplied by an end user
- Should be able to be daemonized or scheduled

Technical overview

- API to API
- A database for persistence

Speaker notes

- Dryad and Dataverse both have relatively well documented Application Programming Interfaces (APIs), so it would make sense to use a programmatic approach to transfer the data.
- The software sits between the two APIs and transfers data from one to the other.
- A tiny database monitors changes

The steps

Speaker notes

1. Create a metadata crosswalk.

   UBC Research Commons has experience with Dataverse mapping from both a migration from moving research data from UBC to Scholars Portal and from an old Dataverse installation to a new one. This is arguably the most important step.

2. Analyze the Dryad collection

   Using the Dryad API, analyse the collection to see if file transfers to Dataverse make sense (ie, are there too many large files).

3. User the native Dataverse JSON as import. More complex, but complete control over what goes where, which is not possible if using DDI or schema.org JSON

4. Start programming

End result

- Pure Python
- Limited dependencies
- A complete script for those who hate programming

Speaker notes

- Pure Python 3 (v3.6 or higher)
- Limited dependency on external libraries
- Three modules
- Serialize -> Transfer -> Monitor
- Script: dryadd.py

The components

- serializer
- transfer
- monitor

Speaker notes

The Python library has three primary components, working in a sequence. In essence, a translator module, an upload module and a monitor.

dryad2dataverse.Serializer

```
>>> import dryad2dataverse.serializer
>>> i_heart_dryad = dryad2dataverse.serializer.Serializer('doi:10.5061/dryad.2rbnzs7jp')
>>> i_heard_dryad.dvJson


['datasetVersion']['metadataBlocks']['citation']['fields'][x]['value']
```

Speaker notes

- Serializer module connects to a Dryad instance and converts the Dryad JSON output to Dataverse JSON output
- Also includes the file JSON
- Technically hardest part; Dataverse JSON is "complex".

Eg: `dv['datasetVersion']['metadataBlocks']['citation']['fields'][x]['value']`, where x is an integer

Complex hierarchical struture based on the underlying database rather than a human-parseable JSON object

dryad2dataverse.Transfer

```
>>> import dryad2dataverse.transfer
>>> dv = dryad2dataverse.transfer.Transfer(i_heart_dryad)
>>> dv.download_files()
>>> dv.upload_study(targetDv='dryad')
>>> dv.upload_files()
```

Speaker notes

Transfers to Dataverse are not necessarily straightforward; it's not possible to update a single metadata field, etc. Dryad2dataverse uses the Transfer object to hopefully get around many of the issues involved

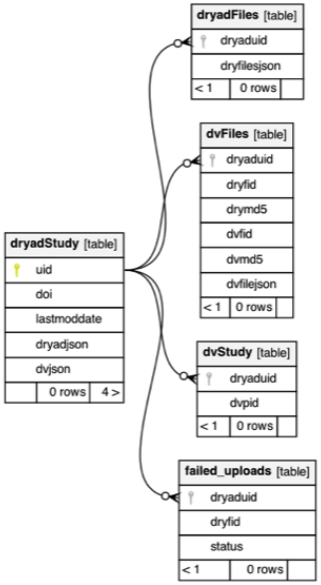
- Transfers require some sort of Dataverse privileges
- Transfer is dependent on the Serializer
- Takes a Serializer instance (ie, a Dryad study) as an input
- Copies the entire Dryad study to Dataverse, using the mappings provided by Serializer
- Puts a copy of the Dryad JSON in the dataverse record so that comparisons can be made if required
- Assigns the "correct" date to a study
- Moves files over *as-is* as much as possible; they are not unzipped as Dataverse does by default
- Tries to mirror the Dryad structure as much as possible
- Creation of .tab files is dependent on settings of the target Dataverse installation

dryad2dataverse.Monitor

```
>>> monitor = dryad2dataverse.monitor.Monitor()
>>> monitor.status(i_heart_dryad)
{'status': 'new', 'dvpid': None}
>>> monitor.status(i_still_heart_dryad)
{'status': 'unchanged', 'dvpid': 'doi:99.99999/FK2/FAKER'}
>>> monitor.diff_files(i_still_heart_dryad)
{}
>>> monitor.update(transfer)
>>> monitor.set_timestamp()
```

Speaker notes

- The first two modules are fine for one-off transfers
- Dryad studies, like Dataverse, are not necessarily constant
- To track changes over time, the previous state must be kept so that there is a method of comparison
- Uses a portable sqlite3 database which stores Dryad and Dataverse metadata
- Monitors both file and metadata changes
- Produces a report of *what* has changed
- In conjunction with the Transfer object, can work to update material that has already been transferred.



scripts/dryadd.py

- The no-programming solution
- Standalone programs available for Intel Mac and Windows

Speaker notes

- A command line program to convert, upload and monitor Dryad studies
- Takes institutional ROR as input
- Requires zero knowledge of Python (with the exception of installation)
- Completely self-contained
- Auto database copy on each run

Features

Speaker notes

- Email notification of new file changes and updates to multiple people
- Options to skip problematic studies
- Can be run at any interval desired by the user

Limitations

Speaker notes

- File spoofing is not perfect by any means. The best solution is to be a super-user and turn of file processing for problematic file types
- Files that are larger than the maximum upload size are ignored. This means that some files may not be transferred. This affected less than 2% of UBC's studies.
- **By design** studies are not published. This allows a manual curation step.
- Does not track and merge any metadata changes made on the Dataverse side
- Date issues: Although publication dates can be changed in the *citation*, they can't be changed as *publication* dates, which are exported to summon. "Publication" dates are Dataverse ingest dates.
- The citation uses Dataverse's **distributionDate** which matches the most recent Dryad's **publicationDate**.

Specific limitations in Dryadd.py

- Auto-mailer setup can be tricky and annoying if using Google for email
- Problems with Dryad API (ie, bad information out) may necessitate manually skipping problem studies

But wait! There's more

- Bulk release utility dryadd.py now available as standalone programs
- Doesn't even require dryad2dataverse

See it in action

https://dataverse.scholarsportal.info/dataverse/UBC_DRYAD

ubc-library-rc.github.io/dryad2dataverse/

Paul Lesack
University of British Columbia Library Research Commons
paul.lesack@ubc.ca

UBC is located on the traditional, ancestral, and unceded territory of the xʷməθkʷəy̓əm (Musqueam), sḷilwətaʔɬ təməxʷ (Tsleil-Waututh), Stz'uminus, S'ólh Téméxw (Stó:lō), Skwxwú7mesh-ulh Temíx̱w (Squamish), and Coast Salish peoples.