An open container for reproducible portable publications

To do:
The name Reproducible Portable Publication (.rpp) is to be discussed
This is modelled after an archive, so on disk we could simply store it in a folder, or we use a .tar.gz file to have a single file for passing around (emailing, downloading etc.)

Motivation

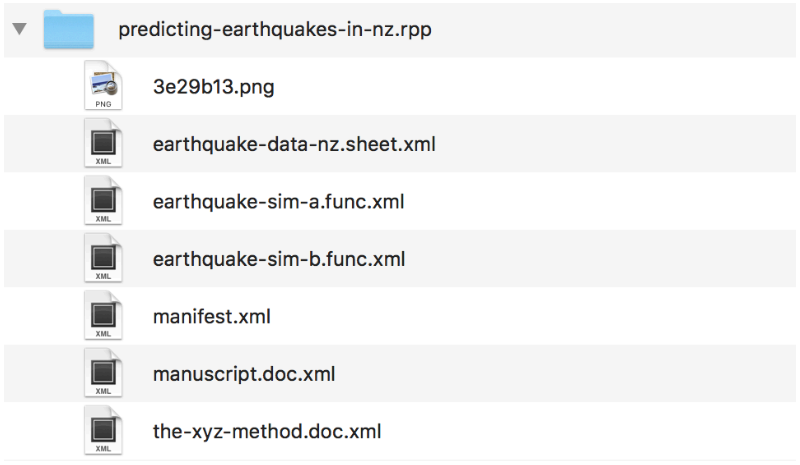
- Fragmentation of manuscript, images, source code, and data in current publications
- Often only the PDF is available as a portable asset, you may be able to download the JATS-XML file but then image files are on some servers of the publishers
- Executable aspects of research often lost upon publication (e.g. Jupyter Notebook gets turned into MS Word for submission, static PDF gets produced, Jupyter Notebook merely survives as an external attachment to the publication)

Requirements

- Zero or more documents in JATS-XML format (e.g. manuscript + additional documents)
- Zero or more sheets (e.g. strongly typed data sheet, or including computed elements via formulas)
- Zero or more static assets (images, supplements)
- Zero or more function definitions (allows scientists to share research algorithms)
- Language and tool-agnostic (e.g. run Python and/or R, open with Jupyter or Stencila)
- Self-contained archive (ideally have no external dependency for offline execution and long-term preservation)
- Compatible with existing standards (CSV, JATS, Jupyter, RMarkdown) via converters

File Layout

Here's a proposal for the Reproducible Portable Publication (.rpp) file layout:



```
predicting-earthquakes-in-nz.rpp
    3e29b13.png
    earthquake-data-nz.sheet.xml
    earthquake-sim-a.func.xml
    earthquake-sim-b.func.xml
    manifest.xml
    manuscript.doc.xml
    the-xyz-method.doc.xml
```

The above fictional publication contains a manuscript about earthquake prediction in NZ. In addition a datasheet `earthquake-data-nz.sheet.xml` is included. As part of the research the author developed two simulation functions which are included as `earthquake-sim-a.func.xml` and `earthquake-sim-b.func.xml`. In order to illustrate the usage of the developed simulation functions the author added `the-xyz-method.doc.xml`, which is a separate technical document in addition to the main manuscript.

Now let's look at the individual files in more detail:

Manifest

The manifest file keeps an index of the documents and functions of a publication.

```
<manifest>
  <name>Predicting Earthquakes in NZ</name>
  <documents>
    <document src="manuscript.doc.xml" type="document">Predicting Earthquakes in NZ</document>
    <document src="the-xyz-method.doc.xml" type="document">The XYZ Method</document>
    <document src="earthquake-data-nz.sheet.xml" type="sheet">Earthquake Data</document>
  </documents>
  <functions>
    <function src="earthquake-sim-a.func.xml" lang=”js”>earthquake_sim_a</function>
    <function src="earthquake-sim-b.func.xml" lang=”js”>earthquake_sim_b</function>
  </functions>
</manifest>
```

Reproducible JATS

Below is an excerpt from a JATS document, showcasing how reproducible (=dynamic) figures are represented. Note that this is still valid static JATS, and can be processed with existing toolchains. Tools such as Stencila can read the reproducible elements and run them interactively.

```
<fig id="f1">
  <caption>
    <title>Figure 1</title>
    <p>Biodiversity on Mars</p>
  </caption>
  <alternatives>
    <code executable="yes" specific-use="input" language="mini">
      bars(counts_by_species)
    </code>
    <code specific-use="output" language="json">
      {
        "execution_time": 322,
        "value_type": "plot-ly",
        "value": {...}
      }
    </code>
    <!-- static version for existing JATS toolchains -->
    <graphic specific-use=”output” xlink:href="89f8b53e361f.svg"/>
  </alternatives>
</fig>
```

Functions

Here’s how a custom sum function (`sum.fun.xml`) is stored in the container:

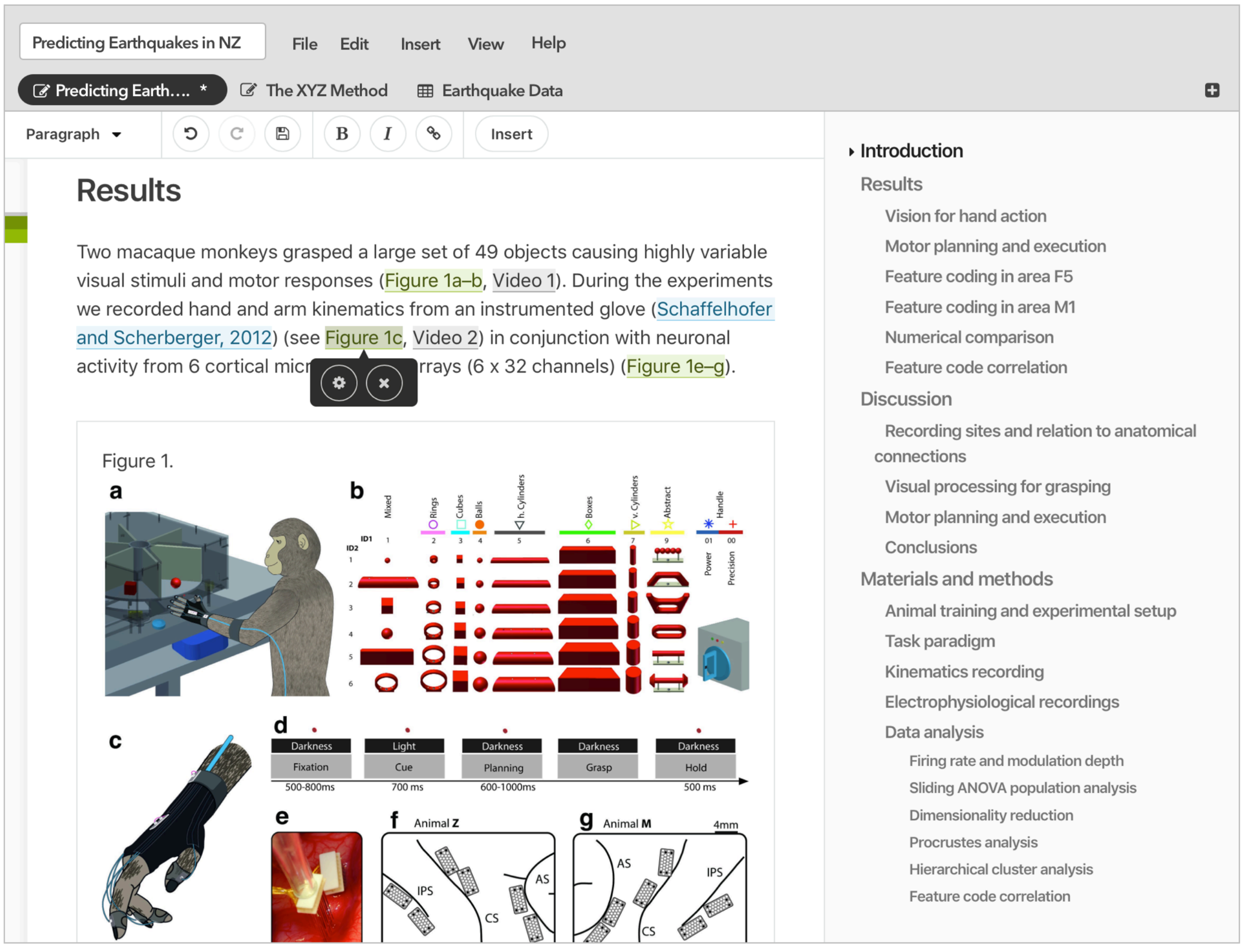
```
<function language="javascript">
  <name>sum</name>
  <description>Adds up some numbers</descriptions>
  <params>
    <param>
      <name>values</name>
      <type>number[]</type>
      <description>The numbers to add up</description>
    </param>
  </params>
  <return>
    <type>number</type>
    <description></description>
  </return>
  <source>function sum(values) { return stdlib.sum(values) }</source>
</function>
```

Sheets

TODO: illustrate sheet data format

Authoring Interface

Each document (sheet or narrative) goes into a tab, the whole thing makes up a research project/publication.



Results

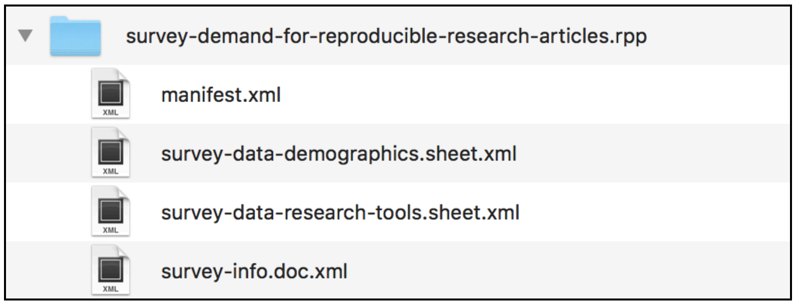
Two macaque monkeys grasped a large set of 49 objects causing highly variable visual stimuli and motor responses (Figure 1a–b, Video 1). During the experiments we recorded hand and arm kinematics from an instrumented glove (Schaffelhofer and Scherberger, 2012) (see Figure 1c, Video 2) in conjunction with neuronal activity from 6 cortical mic rrays (6 x 32 channels) (Figure 1e–g).

Figure 1.

Use cases

Data sharing

A simple way of sharing research could be data-only publication. For instance eLife could share their reproducible research survey data in a publication like so:


survey-demand-for-reproducible-research-articles.rpp
manifest.xml
survey-data-demographics.sheet.xml
survey-data-research-tools.sheet.xml
survey-info.doc.xml

This would be better than sharing a CSV file, since we can have typed columns, as well as an info document included. We could also attach a spreadsheet that does aggregations of the data and plots some charts.

TODO: describe more use-cases

Questions

Wouldn't it be better to store data in a CSV file rather than XML?
CSV does not have enough expressiveness for our needs. E.g. we want to store the column types (e.g. number) as well as the formatting (e.g. currency $). For that reason we will model after an explicit XML format where we can capture all information (types, formulas, errors). This may not be the most efficient serialization format at first, however this does not have priority as it can later be optimized (e.g. using a combination of comma separated values and XML metadata) to represent the same information.

How do I get my CSV data into a sheet and do edits?
You can import from CSV or connect to other data interfaces (to be developed), and then edit the data (e.g. in the Stencila Sheet editor). Alternatively, you can just copy and paste your data over from Excel or Google Sheets. Then do you could either export to CSV or a Data Package.

I prefer Jupyter/RMarkdown for writing reproducible articles, how can I integrate?

We are designing the reproducible JATS spec to be expressive enough for representing Jupyter/RMarkdown notebooks. You'll be able to import a Jupyter Notebook in the Stencila application and edit and execute it. It is then represented as an `.rpp` archive until you export it to a Jupyter Notebook again. Journals are more likely to accept .rpp as a submission format (since the manuscript is modelled in JATS-XML, a format already used). We hope that other communities (Jupyter, RMarkdown) are adopting .rpp as a standardised compile format to allow submission from Jupyter directly without using the Stencila app for conversion.