Handling schema & API versioning in ROR (Final draft)

Updated 22 Nov 2022 following public comment in initial draft Oct-Nov 2022

Overview

Proposed versioning approach

Changes that require versioning

Changes that do not require versioning

Proposing and reviewing schema changes

Deploying new versions & sunsetting previous versions

Appendix A: Tech changes needed in order to support/maintain versioning

Overview

To accommodate changing user needs, as well as evolution from the data model inherited from GRID, ROR must support making changes to its metadata schema and API functionality. In some cases, these changes may not be backward compatible with the existing schema or API functionality, and simply deploying them into the production environment would result in broken ROR integrations.

To ensure a smooth change process for integrators, it's necessary for ROR to implement versioning of its schema and API, along with clear policy and communication surrounding:

- Input on proposed schema changes
- Release of upcoming version changes
- Sunsetting previous versions

Proposed versioning approach

General principles

- The ROR metadata schema and API will be versioned in lockstep, meaning that when a new major schema version is introduced, the API version will also be incremented so that users can unambiguously request a response in a specific schema version.

- Semantic versioning will be used:
    - Minor version (ex, X.1, X.2, etc) will be incremented when non-breaking changes are made, such as adding an element.
    - Major version (ex, 1.X, 2.X, etc) will be incremented when breaking changes are made, such as removing or restructuring an element, removing an API feature, or making significant changes to behavior/functionality.
        - The API request path will change only for major versions. Minor version changes will be implemented within the existing major version request path.

Implementation details

Metadata schema

- Schema files are stored in ror-schema. When a new schema version is created, a corresponding new file will be created and named ror-schema-X.X.json.
- Previous schema files will not be deleted.

Data dump

- Data dumps will be created for all currently supported schema versions (typically 2, but possibly 3 if a new schema version has recently been released
- Data dumps will continue to be added to a single Zenodo record as versions, versioned based on the snapshot date.

REST API

- Major versions must be specified in the path portion of an API request, ex https://api.ror.org/X.X/organizations
- Requests that do not include a version in the path portion will default to the current (unversioned) schema until that schema is sunset. At that point a version will be required, and requests without a version in the path portion will return a 410 Gone error, with a detailed message.
- Minor (non-breaking) version changes will be implemented within the current major version.
- 2 major versions will be supported concurrently (the current major version and the most recent previous version). Requests using an unsupported version will return a 410 Gone error, with a detailed message.

Search UI

- The search UI will not be maintained in multiple versions. It will use the current major version.

Changes that require versioning

Minor version change

- Adding schema elements
- Changing existing API functionality, such that the response to a given request has the same structure, but different meaning/nature (ex, current ?query search behavior is changed to ?query.advanced behavior)

Major version change

- Removing or renaming schema elements
- Changing the structure or data type of a schema element (ex, changing a single value to an array)
- Removing items from controlled lists of allowed values
- Removing API functionality
- Significantly changing existing API functionality, such that the response to a given request is different in structure (ex, removing container element from ?affiliation response)

Changes that do not require versioning

- Adding items to controlled lists of allowed values
- Minor (non-breaking) changes to existing API functionality, such as bug fixes and incremental improvements to search behavior to improve performance/accuracy
- Adding new features to the API that do not impact existing features/functionality (ex, adding a new endpoint)

Proposing and reviewing schema changes

- Suggestions for schema changes will be handled through the existing ROR product process. ROR community members or staff may submit suggestions as Github issues in the ror-roadmap repository.
- Schema change suggestions will be compiled into a draft schema for community review and circulated via the ROR community advisory group and other communication channels, such as the ROR blog, Twitter, PIDForum, Tech support Google group and Githhub discussions. Review will remain open for approximately 1 month.
- Following community review, a final draft of the proposed schema will be created and shared with the community. Any major concerns will be reviewed and considered.

Deploying new versions & sunsetting previous versions

- No more than 1 new major schema version will be released each year, and ROR will not seek to release a major version each year, unless there's need for it. It's very likely that there will be several years between major version releases.
- New versions will be made available in the ROR staging environment for approximately 1 month prior to production release. Users will have an opportunity to provide feedback on the technical implementation of the new schema version via ROR communication channels, such as discussion forums.
- When a new version is deployed to production, any supported previous versions will continue to be supported until their planned sunset date. This may sometimes result in supporting 3 versions concurrently.
- Plans to sunset a previous version will be announced at least 1 year prior to the planned sunset date, via the ROR community advisory group and other communication channels, such as the ROR blog, Twitter, PIDForum, Tech support Google group and Githhub discussions. Regular reminders will continue prior to the sunset date.

Appendix A: Tech changes needed in order to support/maintain versioning

Curation

Because ROR curation activities are performed on JSON files rather than in a database, curation is inherently versioned. This presents a challenge in maintaining multiple versions concurrently. Possible approaches include:

- Shift to using a database for curation purposes, so that ROR record data can be maintained independently of a particular schema version. In this scenario, record files could be exported from the database in multiple schema versions. The data dump could also be generated in multiple schema versions from this database.
- Continue using the current curation process, but generate new and updated files in all supported schema versions.

Validation

- Validation code must be adapted to support versioning
- Additional code and tests must be written to support each new version
- Code and tests must be cleaned up when previous versions are sunsetted
- When fixing bugs or adding features, changes must work in all currently supported versions

API

- Infrastructure must be adapted to support versioning. This will likely involve running multiple ES instances indexed using different schemas and routing requests appropriately.
- API code must be adapted to support versioning (Django REST framework does have versioning features)
- Additional code and tests must be written to support each new version
- Code and tests must be cleaned up when previous versions are sunsetted

- When fixing bugs or adding features, changes must work in all currently supported versions

UI

- UI code must be updated to support changes in latest version

Documentation

- Documentation must be organized to support versioning (readme.io supports versioning)
- Documentation must be updated when new versions are released
- Documentation must be cleaned up when previous versions are sunsetted