Connected Open Heritage report: Keeping data up-to-date

Wikimedia Sverige

Background

Within the framework of the Connected Open Heritage project Wikimedia Sverige has made cultural heritage data from all over the world available through the Wikimedia platforms. This includes both structured data about e.g. a listed building as well as multimedia files illustrating these objects.

Our focus in the project has been on the making available of collections from authoritative actors working in the cultural heritage sector. It however lies in the nature of these institutions that they are continuously refining, extending and developing the data which they maintain on their collections. E.g. new archeological finds get discovered every year, researchers uncover new information about what is depicted in an image and unfortunately sometimes an listing building is destroyed and thus gets de-listed.

Through this report we want to explore and propose solutions to the challenges of keeping the information held on the Wikimedia platforms in sync with the information held by the cultural heritage institutions. Throughout this report we will refer to GLAMs (Galleries, Archives, Libraries and Museums) but the same conclusions apply to government agencies or other agencies providing data or images.

To start investigating the problem we recognize that the underlying problem is three-fold:

- How do we detect that information has been added or removed.
- How do we detect which information it replaces.
- How do we ensure that user-added contributions are not undone/overwritten when we attempt to add the updated information.

Detecting the need for an update

The first challenge is detecting when the source information has been updated. The complexity of this problem varies depending on whether data has been added (e.g. a new item in the database), removed (e.g. a building having been de-listed) or replaced (e.g. an erroneous fact has been corrected).

The straightforward solution is to regularly query the source and compare the result to the result from the previous query. This scales badly, however: one has to maintain a copy of the original dataset and then fetch and compare potentially millions of records only to find that there have been no updates. If the data contains a “last updated” timestamp, the task is made easier as one only needs to compare the entries that have been updated. However, one still needs to fetch all the records and maintain a local copy of all the data. This solution also works badly for deleted records as these can be hard to distinguish from records that are temporarily unavailable.

The solution is to work together with the GLAM to determine when there are updates. How to practically do this can differ depending on how hands-on the GLAM wishes to be and the technical infrastructure the GLAM has.

- In the image case study (LSH) below the updates were isolated and compiled by the GLAM and delivered to us as a single dataset requesting an update.
- In discussions with one data agency (RAÄ) we raised the possibility of an update feed containing both the updated item and the actual changes. Bots could then subscribe to this feed to automatically implement the change on Wikidata.
- By providing a querying functionality and a “last updated” timestamp updated records can be retrieved and re-imported (see comments above).
- By providing a known update schedule (such as when new annual data is released) updating tools can be manually triggered when it is known that there have been substantial changes.

Types of changes

Different types of updates require different types of action on our side and are differently complicated to implement. The following main types of changes and corresponding actions were identified:

- **Data**
  - Error discovered in imported data (e.g. research showed that a construction date was incorrect): Update the affected values and their sources.
  - A new entry is added (e.g. a new building is listed): Create a new item.
  - An entry is removed (e.g. a building is delisted): Set an end date to previously imported values.
  - Changed data (e.g. the protection level for a building is raised): Add the new value and set an end date to the old value.
  - Updated data (e.g. there is a new measurement of the water quality level of a lake): If value is the same then no change is necessary, otherwise add the new value and, if needed, set an end date to the old value.
- **Images**
  - A better image has been created (e.g. new images were taken of an object): Upload as a new image.
  - A better version of the image is available (e.g. a higher resolution version was gotten from the original): Replace the image.
  - New or corrected info (e.g. new research on an object): Regenerate the image description page and replace the old one.

To some extent **data** is easier to deal with than **images** since Wikidata allows for conflicting values (where one is ranked as preferred). Data is also modular, and the structure of

Wikidata makes it possible to just update the needed fact rather than the full page content. At the same time it might be non-trivial to determine if the previous value implicitly needs updating (e.g by adding an end time qualifier). It is also much harder to identify user changes to a previously imported value (e.g. if a value was deleted), leading to the risk of undesired values being re-imported. In most cases this is resolved by using ranking rather than deleting statements.

It is also worth noting that there are some updates (in either images or data) which do not necessitate an update on our side (e.g. because we are not making use of that bit of information). When possible, these should be excluded before any updating is attempted. This is facilitated by maintaining a list of fields in a particular dataset that are of interest to us.

How to perform the updates

Case study: LSH (images)

For several years we have been performing batch uploads of images together with the Royal Armoury, Skokloster Castle and the Hallwyl Museum (LSH). In 2016 we were approached with a request to perform two types of updates to previously uploaded images.

- Replace some image files where the photographer had gone back to the original RAW files to perform better colour balancing.
- Replace some image descriptions for objects where a research project had revealed a substantial amount of new information (sometimes invalidating the information held during the original upload).

The first request was quite straightforward to fulfill. The only possible user changes to an uploaded image are if a new version has been uploaded, if the image has been renamed or if the image has been deleted. When performing batch uploads we always ensure that the unique ID of the image is included which allowed us to find the image to replace even when they had been renamed. Similarly deleted images will not return any matches thereby informing us that they should be skipped. Compared to other changes, the upload of a new version of an image (over the old image) are fairly rare, nevertheless they do happen. We therefore ensured, for every image we wished to replace, that the latest file in the file history was uploaded by the account used by us for the batch uploads. Where this was not the case the update was reviewed manually.

The second request, updating object descriptions, required a more complicated solution. Part of the problem is that the image description page is a single wikitext blob so it is not trivial to determine which parts an update affected. At the same time, most of the user changes are related to categorisation which can fairly easily be separated from the remaining content.

The first step was to regenerate the image description page based on the updated information. This text was then compared to the earliest version of the image description page. If it was the same, this meant that the updated information was not used in our image description and so no update was needed.

Then, the earliest and the latest versions of the image description page were compared. If these were the same then there had been no (un-reverted) changes to the page and we were free to update it as we wished. If they differed by more than just categories (or HTML comments) the update was instead performed manually.

When the earliest and the latest version only differed in categories, then the rest of the page was replaced by the newly generated text. As we replaced all initial categories with the final set of categories, we ensured not to re-add any categories removed since the first version, and we kept any categories added since the first version.

The logic above allowed us to automatically update 96% of the images. For the remaining ones, we produced a diff between the first and the latest version which allowed the changes to be easily added to the newly generated information.

One known issue is that the above logic can only be used once per image since a later update would encounter all of the changes introduced here. The update procedure assumes that the 1st version of the image description is the baseline that we compare to new information. If we only do one update project throughout the life of the file, that's not a problem. But if we want to introduce even more updates at some later point, the question is, which version should be the baseline? A solution could be to assume that the last version saved by our bot account is the baseline, which may or may not be equal to the 1st version, depending on whether this is our first attempt to update. The bot account would here be the account responsible for the initial upload and/or subsequent updates.

For more details see:
https://se.wikimedia.org/wiki/Projekt:LSH-tillg%C3%A4ngligg%C3%B6rande_2016/Updating_descriptions

Case study: RAÄ (data)

As part of Wiki Loves Monuments the data from Swedish National Heritage Board (*Riksantikvarieämbetet*, RAÄ) on listed buildings and archeological monuments has been available on Wikipedia for many years. Keeping the information up to date, however, has always been a struggle. As part of the Connected Open Heritage project, this data was migrated to Wikidata where keeping it updated should be easier.

The main barrier for keeping the data updated is the discoverability of relevant changes. Both datasets are in active use and may see multiple changes per day. Today, however, the only way of discovering these is by investigating each item through the API, checking when it was last updated and if needed comparing every statement to the values stored on Wikidata. Another facet of this problem is that RAÄ's database contains some information that is not compatible with Wikidata, such as free-text descriptions of the objects and their history, or not relevant for Wikidata. Updates to these fields should be ignored and consequently most updates will prove to not require any change in the Wikidata information.

A suggestion which was brought up to address this was the introduction of an RSS feed (or similar) maintained by RAÄ, which updates every time there is a change to one of the public fields in the database. The update would contain information about when the change occurred, which item was changed, which field(s) were changed, the type of change (addition, deletion or change) and, if relevant, the new value of the changed/added field. The field would also include updates corresponding to the complete deletion/unpublishing of an item as well as when a new item gets added.

Once the feed has been activated, a bot maintained by the Wikimedia community could subscribe to it and parse each update to detect if it affects any properties marked as being in use on Wikidata for the dataset. If so, it would load/create the corresponding Wikidata item and:

- for an addition: add the value (with a timestamp),
- for a deletion: add an end date to the value (and lower its rank),
- for a change: do both of the above.

This assumes that all (non-new) items mentioned in the feed already exist on Wikidata and are tagged with the corresponding ID. A less automated mechanism could instead propose these changes for a human editor to review and quickly add. By not deleting data (only modifying its rank) we ensure that user-generated contributions do not get removed by our updates.

To avoid accidental changes or multiple sequential changes to the same item either the RSS feed or the bot could employ a cooldown period whereby all change to an item are combined over a given time period and a final update is only done if there is a net change to the item at the end of this time period.

A necessary prerequisite to a successful implementation of this idea is close cooperation with the GLAM; ideally, with staff who have a basic understanding of Wikidata and the migration project. This is especially important if the maintenance of the updater bot is left to the Wikimedia community, as the easiness (or lack of it) of consumption of the RSS feed could prove to be a bottleneck.

Case study: WFD (data)

Together with the Kalmar County Administrative Board we had an earlier project aimed at importing water quality data to Wikidata for all bodies of water in Europe. This data is collected by the EU under the Water Framework Directive (WFD) and a new report is released every 6 years at which point it becomes relevant to amend the data on Wikidata.

Detecting the need for an update in this case becomes trivial in so much as a completely newly published dataset always being a reason for updating, even if just to add the new source to state that a value is still true several years later.

This update is primarily of the “updated data” type. For instance, if the chemical status of a lake was previously “bad” it might now be “good” whereas the ecological status of the same lake remains “bad”.

The workflow which was designed relies on manually re-running the import scripts with the new data files each time new reports are released. Added to the original import logic the following logic for handling updates would be applied:

- If the status is the same: Ensure that a start date (not just a timepoint) is used as a qualifier for the original value. Add the new source.
- If the status has changed: Add an end date to the old status (and ensure it already had a start date). Add the new status as a new value with a start date and the new source.

For values consisting of lists (e.g. pollutants), the same is done as above, with the following additional logic:

- If the value was not present before – add it as a new value with a start date and a source.
- If a pre-existing value is not present in the new dataset – add an end date to the previous value.
- If either the new or the old lists were completely empty, handle this by adding a “no value” statement (or setting an end date to such a pre-existing statement).

Open for debate is still whether the new source reference should also be added to values which get an end date qualifier so as to source that part of the statement. An advantage of this approach is that it signals to the consumer that the data is being actively maintained, therefore raising the reliability of the dataset.