SCHOLARLY PUBLISHING OFFICE WHITEPAPER[*]

Specifications for implementing web feeds in DLXS

Kevin S. Hawkins

Executive Summary

SPO uses DLXS to deliver nearly all of its publications online, but this software does not offer any functionality for *web feeds* (commonly called *RSS feeds*). The components of web feeds are reviewed, and recommendations are made for implementing Atom 1.0 web feeds in SPO publications using a script run during releasing which determines which content to push to the feed.

Motivations

Members of the editorial board of the *Journal of Electronic Publishing* (collid `jep`) and SPO staff have requested that web feed functionality be provided for SPO publications. This would allow end users to subscribe to feeds of not only serial content (the obvious use) but in theory any other collection to which content is added, especially on a regular basis. A feed could provide a single notice that a collection has been updated, perhaps with a table of contents (henceforth a *small feed*), or instead contain separate notices for each item added to the collection (a *large feed*).

Components of a web feed

A web feed operates with three necessary components:

- a website that *syndicates* (or *publishes*) a feed
- a feed *standard* being used by the website
- the end user's *news aggregator* (*feed reader* or *news reader*), which retrieves the feed periodically.

A feed is an XML document containing individual *entries* (also *items*, *stories*, or *articles*).

Syndicating a feed

The content provider's website often has buttons (icons called *chicklets*) to be used by the end user to add a feed to his or her news aggregator. This feed may be provided in any of a number of feed standards, and often more than one is available. There are two types of chicklets:

- those that provide a feed's URL, which can be manually given to a news aggregator for inclusion
- those that, when clicked, add a feed to a news aggregator automatically

There is no standard protocol for adding feeds to a news reader, leading to a proliferation of chicklets of the second type. Content providers can avoid needing to keep up with the idiosyncrasies of specific aggregators and avoid using valuable webpage real estate by using a service such as FeedBurner.[1] Given a URL of the first type, FeedBurner provides a special page

[1] <http://www.feedburner.com/>

for the content provider's website with chicklets for various news aggregators. FeedBurner also provides additional marketing and publicity services.

Feed standards

The two most common web feed formats are RSS 2.0[2] and Atom 1.0 (the *Atom syndication format* or simply *Atom*).[3] Both use simple XML schemes, and nearly all aggregators today support both RSS 2.0 and Atom 1.0.[4] These standards are compared in detail in Wikipedia[5] and by Tim Bray.[6]

News aggregators

News aggregators are of two types:

- client software aggregators
- web-based aggregators (hosted on websites)

The choice of news aggregator belongs entirely to the end user. Which aggregator is used by an end user is of no concern to the content provider since nearly all aggregators support both RSS 2.0 and Atom 1.0.

Relationship to DLXS

Since there are many existing services and software for web feeds, adding web feed functionality to DLXS would seem relatively straightforward. The web feed could be linked to from a chicklet on a static HTML page, meaning no changes to DLXS templates would be required. However, if we want chicklets provided on templates, changes to the DLXS templates would be required.

Since the feed would be updated at the point of release, a script to update the feed would operate similar to `mkcrawl`: in conjunction with releasing yet independent of DLXS mechanisms.

However, the difficulties for a DLXS implementation lie in determining what information to provide in the feed. For example, if the feed will consist not simply of a single entry announcing a new issue of a journal (small feed) but instead will consist of entries for each article in the latest issue (large feed), then a reliable method for extracting metadata of only the newly released content needs to be developed. The indexed content (`/l1/obj/c/collid/collid.xml`), with appropriate *fabregions* defined in files at `/l1/idx/c/collid/`, can serve this purpose for Text Class (including all three `DocEncodingType` values), Bib Class, and Findaid Class items. Image Class items, on the other hand, have their metadata stored only in the Media Table of the MySQL database, so this would need to be consulted to generate the metadata needed for the large feed.

Feeds usually contain a set number of the most recently updated entries on a particular website, such as the last ten posts to a blog. At any given time, the small feed should contain only one entry (the latest one), and the large feed should contain entries for *all* items added to the collection in the most recent batch only.

---

[2] See < http://www.rssboard.org/rss-specification>.
[3] See < http://www.atomenabled.org/developers/syndication/>.
[4] See "XML Formats" at <http://www.aggcompare.com/>.
[5] See <http://en.wikipedia.org/wiki/Atom_%28standard%29#Atom_Compared_to_RSS_2.0>.
[6] See <http://www.intertwingly.net/wiki/pie/Rss20AndAtom10Compared>.

Recommendations

Scope

Web feeds should be offered for all collections, public and restricted. For restricted collections, authentication would happen only if the user attempts to access the full content. Consent of content providers to set up large feeds may be needed for these restricted collections.

Since SPO's greatest need is for Text Class items, web feed functionality should be implemented for this class first. Two web feeds should be offered for each collection: one containing one feed entry per collection update, containing a simple notice that content has been added with a table of contents (*small feed*), and the other containing one feed entry per item added to the collection (*large feed*). The small feed might not be worth providing on a collection basis since users can find out when a collection is updated by simply subscribing to the large feed. However, the small feed functionality could be used for a "SPO news items" feed on the new SPO website instead.

Changes to DLXS templates, as well as scripts used in developing feeds, should be added to the DLXS code base since it might prove useful to DLPS, DLXS subscribers, or both.

Web feeds should be implemented first in `jep`, whose content providers have requested the functionality, and in `phimpz`, which release content irregularly and so most needs this feature.

Chicklets

Initially chicklets on static HTML pages will link to the feeds, but later chicklets will be added to the navbars or footers of collections. Placement of the chicklets will depend on ease of coding in DLXS, as determined by the SPO programmers, and on interface design considerations, in consultation with the DLPS user testing and interface specialist, but will likely appear in the template navbars or footers. We should use a standard, easily identifiable icon for chicklets in all collections. The best choice for this is the icon from Mozilla Firefox, distributed by the Mozilla Foundation under an open-source license and adopted by Microsoft and Opera for their software.[7] However, the prevalence of the term *RSS* for web feeds should be taken into account in the choice of icon and `alt` text.

Feed format

Atom 1.0 is the clear choice for feed format. While the RSS formats have been developed by the user and developer community somewhat haphazardly, Atom 1.0 adheres to various Internet standards and is itself an IETF standard (RFC 4287[8]). Since it is nearly as widely supported as RSS, SPO should implement Atom 1.0 and not an RSS version to adhere to its goal of compliance with open standards.

In addition, SPO could use FeedBurner for each of its feeds, providing an additional button from the static HTML to a FeedBurner page for the feed, which would allow users to add the feed to their aggregator with one click. Using FeedBurner gives SPO additional statistics for marketing.

---

[7] See < http://en.wikipedia.org/wiki/Image:Feed-icon.svg>.
[8] < http://tools.ietf.org/html/rfc4287>

Mechanics

The updating of feeds should be done at the point of releasing content. As with `mkcrawl`, it will be most effective to invoke a feed script (`mkfeed`) from within a collection's `rdist` script. The two feeds would be updated as below.

Small feed

This plan assumes that small feeds will be generated for each collection, but it could easily be appropriate for a "SPO news items" feed.

The small feed can be generated by consulting a database table with records of when each collection was last updated:

`ID`	primary key
`collid`	text
`updated`	timestamp (when the collection was last updated)

For a given collid, `mkfeed` would check whether the current time is more than a certain number of hours after the timestamp for this collid. If so, it would leave the same `atom:id` on the feed entry element; otherwise, it would create a new value for `atom:id`, causing aggregators to retrieve the feed entry. In either case, it should update the timestamp in the database table with the current time and replace the existing single feed entry in the feed file with a new feed entry.

This entry should at least give a generic message saying something like "The Scholarly Publishing Office announces new content available in [collection name]." At most, this one entry should give the volume, issue number, and date, plus a little table of contents listing each article's title and author and, most importantly, a link to the collection's homepage. Retrieval of title and author for each article would be accomplished using XPAT fabregion searches, as described below for the large feed.

This feed entry would be wrapped in `feed` element with appropriate child elements. Then this XML file would be uploaded to the production servers at `/l1/web/c/collid/smallfeed.xml` (or another filename).

Large feed

The large feed can be generated by consulting a database table with records of when each item in a collection was first put online:

`ID`	primary key
`IDNO`	text
`collid`	text (which collid each IDNO belongs to, in order to disambiguate IDNOs in more than one collid)
`online`	timestamp (when the items was first put online)

For a given collid, `mkfeed` would retrieve a list of IDNOs from `/l1/prep/c/collid/idnosOnline.txt`. For each IDNO, it would attempt to retrieve a record from the database table: if the record does not exist, it would create a record with the current time as the value of the timestamp.[9] For each IDNO of a newly created record, it would create a feed entry for the item associated with this IDNO based on XPAT fabregion searches on the indexed content (`collid.xml`).[10] These fabregions are defined in files in `/l1/idx/c/collid/`, where differences in encoding structure among the `DocEncodingType` values are not visible.

The feed entry should contain all useful metadata which can be extracted from the indexed content. For Level 4 items, this would give us reason to begin tagging abstracts consistently so that a fabregion of abstracts could be constructed for each collection and these could be included in the feed entries. In addition, the search template in DLXS could make use of this fabregion as well. For items without abstracts, the first body paragraph might be used instead, either in the abstract fabregion or in a separate one that would be excluded from DLXS searches but used to supplement the metadata in the feed entries.

In the end, all new feed entries would be concatenated and wrapped in a feed element with appropriate child elements. Then this XML file would be uploaded to the production servers at `/l1/web/c/collid/largefeed.xml` (or another filename).

Once the Unified Bibliographic Database is developed and put into use, it could be used instead of the database table described above.

---

[9] This allows us to determine which items have been added to the collection for the first time. Another option is to use the `idnosOnline.txt` file, as in the `llmc` update process. In either case, we have chosen not to attempt to track the time of last update of a file instead of when it was first put online because doing so would rely on timestamps on files in `/l1/prep/`, which are unreliable as a source of data. Besides, items are only updated when we discover errors in them, so this is not something we want to publicize.

[10] This method is chosen because there is no more reliable location for metadata about individual texts across the three types of `DocEncodingType` in Text Class.