2007-05-29

Choice of DocEncodingType and encoding level for SPO publications

Hawkins, Kevin

BECAUSE SCHOLARLY PUBLISHING OFFICE WHITEPAPER[*]

Choice of `DocEncodingType` and encoding level for SPO publications

Kevin S. Hawkins

Executive Summary

SPO uses DLXS to deliver nearly all of its publications online, with Text Class items structured according to three different `DocEncodingType` values and three different encoding levels. This whitepaper explains SPO's current and possible use of all nine combinations of `DocEncodingType` and encoding level for electronic publications, giving advantages and disadvantages for each and providing considerations for intake of future publications.

Basic digitization workflows

When reformatting print copies of publications, SPO often takes advantage of the Digital Conversion Unit (DCU) workflow, in which a bound copy of a document—a book, a journal issue or a bound volume of a journal[1]—is scanned, OCRed, and pagetagged.

When publishing born-digital texts from content providers, SPO usually converts to electronic text directly. However, when page layout is important to the content providers or when the backfile is large compared to future issues, SPO will receive PDFs and use some or all of the reformatting workflow for these born-digital texts.

Background

The DLXS middleware used by SPO delivers textual content using its Text Class functionality. Every Text Class item must be of one of three encoding levels, and one `DocEncodingType` must be declared for the entire collection.

The EDITORIALDECL element in the HEADER specifies the level of encoding, with the N attribute equaling *1*, *2*, or *4*, corresponding to Level 1, Level 2, or Level 4 of *TEI Text Encoding in Libraries: Guidelines for Best Encoding Practices*.[2] Level 1 and Level 2 always use the *pageviewer* (pageturner); Level 4 may do so but need not. (Level 3 of the guidelines strongly resembles Level 4 but is not supported as such in DLXS.)

The `DocEncodingType` field in `collmgr` may have one of three values: `monograph`, `serialissue`, and `serialarticle`. As of release 13, these values constitute three *de facto* subclasses of Text Class. Each collection must declare exactly one of these `DocEncodingType` values for the items in that collection.[3] Each value assumes a different structure for the encoded text and delivers this structure in a different way, taking advantage of various parts of the Text Class code.

---

[1] Practice in the Digital Conversion Unit for scanning bound volumes has varied over time. Sometimes issues in the volume are scanned individually and sometimes the volume is scanned as a unit.

[2] http://www.diglib.org/standards/tei.htm

[3] If the markup is carefully controlled, it is possible to mix `monograph` and `serialissue` texts in one collection whose DocEncodingType is set to monograph. The middleware will treat `serialissue` texts as `monographs` (e.g., http://hdl.handle.net/2027/spo.aag0609.0002.002).

The possible combinations of encoding levels and `DocEncodingType` values are given here with examples of each:

`DocEncodingType`	Encoding level			
	Level 1	**Level 2**	**Level 4 with pageviewer**	**Level 4 without pageviewer**
`monograph`	• `machyn` 1830s MS • `hiss`	• `acls` (backlist)	• `machyn` edition	• `acls` (frontlist) • `spobooks` • `globalpublics`
`serialissue`		• `basp` • `mdiaarchive` • `mqrarchive`	(Abraham Lincoln Association is planning)	(none attested)
`serialarticle`	• `jar` • `ark` • `mjcsl` • `fs` (back issues only)	(none attested)	`phimp`	• `passages` • `mfr` • `jep` • `postid` • `mdia` • `fs` (recent issues)

Article-level metadata

Journal metadata is stored in the following locations:

	`serialissue`	**`serialarticle`**
issue-level metadata	HEADER	HEADER
article-level metadata	BIBL (in BODY)	HEADER

The use of specific elements within HEADER and BIBL for various metadata fields is determined by the middleware and loosely based on TEI practice but nevertheless various across collections. While we plan to standardize the practice, possibly leading to a modification of the DTD to allow for cleaner tagging, it is not clear whether the HEADER or BIBL is ultimately better suited for article-level metadata.

`Monograph` collections

`Monograph` collections allow each item to be presented with a `view=toc`. This view shows a list of pages (Level 1 encoding) or the DIV#-HEAD structure (Level 2 or Level 4).

Level 1 `monograph`

The standard reformatting workflow results in a Level 1 `monograph` item. The `view=toc` shows a list of pages, displayed with numbers, features, or both, according to pagetagging information. Pages can be viewed using the pageviewer, and by default the OCR text is accessible from the pageviewer as well.

Level 2 `monograph`

If we use the standard reformatting workflow yet we have the resources to provide direct access to sections of the text (such as chapters), we can insert markup by hand that is read by the `view=toc` to generate something that resembles a table of contents for the monograph.

Like Level 2 `serialissue` collections, DIV1-BIBLs are inserted in the OCRed text. However, HEAD tags are also inserted, containing the text to be displayed in the `view=toc`, and the BIBL tag contains only a BIBLSCOPE for that section's starting page number. This BIBLSCOPE lacks other BIBL tags that are used in Level 2 `serialissue` collections to allow for direct access to articles by author and title fabregions.

Level 4 `monograph` with pageviewer

This results when a monograph is scanned front print but also keyboarded to produce Level 4 markup, either because OCR doesn't work well or because the content is especially valuable. The `view=toc` displays a full table of contents based on the DIV#-HEAD hierarchy, and search regions may be set up to take advantage of granular encoding.

This model was also used for the `machyn` edition, where the pageviewer was used for the manuscript on which the edition is based.[4]

Level 4 `monograph` without pageviewer

When a monograph is received in electronic format (such as an author manuscript in Word format), or in theory in cases where the original print object is keyboarded but never scanned, a Level 4 text is produced. The `view=toc` displays a full table of contents based on the DIV#-HEAD hierarchy, and search regions may be set up to take advantage of granular encoding.

`Serialissue` collections

In a serialissue collection, the middleware looks for DIV1-BIBL structures ("subdivs" or "article divs") in the text. When only a `c` and `idno` parameters are given, it displays a sort of table of contents for the digitized item, listing all the DIV1-BIBLs.

Level 1 `serialissue`

This is impossible since in a `serialissue` collection, the middleware searches for DIV1-BIBLs, which are absent in Level 1 texts.

Level 2 `serialissue`

If we use the standard reformatting workflow yet wish to provide direct access to sections of the text (such as articles in the journal issue or bound journal volume), we insert DIV1-BIBLs into the text to mark pages where the text divisions begin. SPO used to do this by hand, but now `makebackfile.sh` can do this based on a FileMaker database.[5]

This resembles a Level 2 `monograph` but has no HEAD tag as the first child of each DIV1 and has more BIBLSCOPE tags for various components of the article metadata.

---

[4] See http://hdl.handle.net/2027/spo.5076866.0001.001.

[5] Chris Powell uses XSLT to accomplish the same thing.The `makebackfile.sh` and XSLT methods currently handle articles that begin in the middle of a page differently, leading to slightly different result sets when using the two methods.

Level 4 `serialissue`

No Level 4 serialissue collection is known to have been attempted; however, Chris Powell reports that the Abraham Lincoln Association is planning to create such texts. These would presumably look like Level 4 `monograph` texts, except that the DIV#-HEAD hierarchy would include articles rather than chapters.

While such a collection would allow for a hierarchical organization of a journal issue's contents (with a clear representation of subarticles, if they existed), it seems that the current DLXS code would have trouble providing access to individual articles without the use of BIBLs. A Level 4 `serialarticle` collection would give all the desired benefits of a Level 4 `serialissue` collection without the complications.

`Serialarticle` collections

The `serialarticle` `DocEncodingType` was developed for SPO to publish born-digital journal articles outside the reformatting workflow, thereby allowing SPO to move away from the print convention of the issue as the basic unit of a journal. By making each article its own Text Class items, SPO was able to provide:

- stable URLs for individual articles
- article-level usage statistics
- bookbag functionality at the item level

Level 1 `serialarticle`

This model has been used for publications in which features of the print page layout must be retained in the online version. PDFs of individual articles are provided by the content provider, and parts of the reformatting workflow are used to give a pageviewer collection of individual articles. In some cases (specifically `jar` and `phimpz`) content providers have specifically requested that only page images and PDF files be presented to the user—in this case, the XML is used only for searching.

Since we have so far used this model for publications where electronic source files are provided, we use the electronic text extracted from the source files, rather than text generated by OCR, for the collection index in order to give higher search accuracy and better structure. While some markup (primarily P tags) can be automatically inserted into this extracted text, there is insufficient markup to create automatically a Level 4 text.[6]

Were it not for the existing preservation workflow that favors the issue or bound volume as the basic unit, we could use this model for print backfiles as well, resulting in a publication like jar except that the page images would be created by reformatting from print. This would simply require reworking of prep scripts.

Level 2 `serialarticle`

No such publication is known. It would, in theory, consist of a pageviewer collection of individual articles, each of which would have DIV1-BIBLs marking individual sections of the article. When no parameters are given in the URL aside from `idno`, an outline of the document would be given, appearing similar to the list of articles in a serialissue item.

---

[6] For jar we strip the P tags out during indexing, but in the future, especially for publications that wish to provide access to the text view in the pageviewer, we might modify `makepub.sh` to output more usable extracted text that can be indexed without additional modification.

The process for creating such a text would be similar to `acls` backlist tagging, involving insertion of markup at appropriate places.

Level 4 `serialarticle` with pageviewer

This model is used for `phimp`, where the source files allow easy creation of Level 4 markup even though electronic text is not desired by the content provider. SPO will use this model instead of creating Level 1 text according to the `jar` conversion method to avoid discarding structured information.

Level 4 `serialarticle` without pageviewer

This is the most common model for SPO publications and is used for publications consisting of individual born-digital articles. They are most often converted from various source file formats into RTF and then processed using `makepub.sh`, but sometimes files are instead turned into Text Class markup by way of another markup language (as in the Simoni monographs).

SPO has also used this model for backfiles by upgrading markup generated through the reformatting workflow (as in the `mfr` backfile), keyboarding (as in the `passages` backfile), or upgrading of markup, such as from HTML (as in the `jep` backfile).

Comparison of features available in the publication models

The following features of SPO publications will work regardless of publication model chosen:

- **Author, title, and subject browse**: The current dynamic browse scripts can build author and title browse lists based on any of the possible models.
- **Browsing by issue and article**: A static HTML page can be used to list volumes and issues. Alternatively, the `picklist` code under development can be used with any publication model. For `serialissue` collections, it can be used in conjunction with the `toc` view to produce a list of volumes or issues (depending on the size of the collection); for `serialarticle` collections, it can be used to produce a list of volumes or issues (depending on the size of the collection), plus lists of articles in an issue.
- **Other metadata searching**: We have taken advantage of the flexibility of the Text Class header to encode collection-specific information in item headers, such as keywords or article type (as in `jii` and `gefame`); however, it should be possible to store this information in BIBLs as well.
- **Metadata harvesting by OAI**: OAIProvider shares article-level metadata for all publication models.
- **Handles**: Handles are created for items in `serialarticle` and `monograph` collections as part of the release procedure and at the article level for `serialissue` collections.
- **Search engine crawling**: Full-text indexing by major web crawlers is available for all public collections. As far as we know, only Google follows links to OCR text in Level 1 and Level 2.

- **Usage statistics**: Article-level usage data for both `serialissue` and `serialarticle` collections is handled by the new stats system.[7]
- **Web feeds**: Web feeds can be implemented for any Text Class collection.
- **High-resolution images**: Linking from Text Class items to Image Class items can be achieved for all publication models.

However, other factors remain in choosing a model.

Size of backfile and frontlist

Economic considerations often trump other factors in deciding on a publication model. If a publication has a large backfile for which no reliable electronic source files exist, we will almost certainly use the preservation workflow (used for Level 1 and Level 2 items) for the backlist because of the prohibitive cost of upgrading the markup through keyboarding. Exceptions include:

- publications with small backlists, for which investment in upgrading is minimal (`mfr`)
- publications whose backfile is printed in newspaper format, with jumps across pages and multiple columns on a page, for which a pageviewer mechanism would be difficult to use (`passages`)

Monograph collections should be done using Level 2 instead of Level 1 when there is sufficient funding to facilitate this kind of direct access to sections of the text.

It's also possible to mix encoding levels within a `serialarticle` collection. For *Feminist Studies*, we will likely have the backfile scanned with each article would be scanned under a different IDNO (Level 1) and will put new issues online as encoded text (Level 4).[8]

Format of source files from content provider

Frontlist content comes to SPO in various file formats, which could be broadly divided into those which generally guarantee uniformity of page layout and typography across platforms (like PDF and LaTeX) and those that do not (like word processor, page layout, and desktop publishing file formats). If SPO has only the former, it is more straightforward to generate Level 1 or Level 2 content; if it has only the latter, it is more straightforward to generate Level 4 content. While it is possible to take PDF or LaTeX files and generate Level 4 markup, it is nearly impossible to take formats like Microsoft Word or Adobe InDesign and generate Level 1 or Level 2 content that will match the pagination and page layout that the content provider intended.

Concern of content providers for page layout

Some content providers feel that maintaining print-like page layout in an electronic version is of utmost importance (as with `jar` and `phimp`) and therefore have no interest in Level 4 text. An additional benefit to page-image content is that it prints faithfully, unlike with

[7] Usage of an individual article is recorded as a section-level hit for `serialissue` collections and an item-level hit for `serialarticle` collections.

[8] This is possible in DLXS collections as long as each index file contains at least one text of the deepest level of markup found in the collection. Doing this for *Feminist Studies* will give Jeremy an opportunity to rewrite the code for Level 1 `serialarticle` collections so that whether or not the pageturner is displayed is determined not at the collection level but at the item level, based on the `encodinglevel` of each item.

electronic text. These publications should use Level 1 or Level 2 encoding, optionally disabling user access to the text view in the pageviewer.

Note that DLXS expects that divisions in Level 2 texts begin only at page breaks, meaning that there will always be difficulty in accurately encoding articles that begin mid-page and pages with more than one article on them.

"Textualness" of content

While Level 4 Text Class markup can handle the most common content objects, there are many textual features that it is incapable of satisfactorily encoding and rendering:

- complicated tables
- textual diagrams
- mathematical and chemical formulae
- meaningful text layout, such as the "rat's tail" in *Alice in Wonderland*

These elements are sometimes captured and rendered as inline images in Level 4 items, but any textual content within these images is not searchable using XPAT.

For consistent rendering of these content objects, an entire collection—even its frontlist—may be kept in Level 2 markup (as was done with `basp`) rather than choosing Level 4.

On the other hand, Level 4 collections are much better suited for the following:

- links to high-resolution images
- links to multimedia content
- hyperlinks for footnotes, cross-references, and external sites
- customized searching of regions of text (taking advantage of structured XML)

Item-level access settings

DLXS allows for item-level access controls, meaning that differential access is available down to the level of an article in a `serialarticle` collection but only to an issue in a `serialissue` collection. Article-level access for `serialissue` collections would need to be engineered.

If pay-per-view functionality is ever added to DLXS, this would most likely rely on the item-level access controls, yielding one more disadvantage to the `serialissue` model.

Conclusion

For journals it seems preferable to choose the `serialarticle` model over the `serialissue` model since having the article as the basic unit rather than the issue would be better suited to allowing content to be reused in a way that no longer reflects the issue structure of the journal.

Recommendations for choice of markup level

The choice of markup level depends on file formats available from the content provider and on whether the content provider insists on preservation of page layout. The following table shows which file formats can be converted into which markup level:

Format from content provider	Available levels of markup
"plain text"	Level 4

word processor formats (Microsoft Word, RTF, WordPerfect, etc.)	Level 4
page layout or desktop publishing formats (Adobe InDesign, Adobe PageMaker, QuarkXPress, Microsoft Publisher, etc.)	Level 4
PDF, PostScript, LaTeX	Level 1, Level 2, or possibly Level 4*
paper copies	Level 1, Level 2, or Level 4 (using OCR Proofing or vendor)

*Level 4 text can be created from PDF files, or from OCRd text (as with the `mfr` backfile), if the flow of text in the document is simple, but the quality of the output varies greatly depending on how the PDFs were generated. Extracting from PDF files and processing with `makepub.sh` can in many cases lead to:

- words broken due to hyphenation
- words running together because a space character was missing from the end of a line
- a new `P` element beginning at each column break and sometimes even at each line break

All of these can interfere with phrase searching in the data.

The following shows which file formats are required for pageviewer versus encoded text delivery:

Delivery type	**File format options**
pageviewer	• paper • PDF • PostScript • LaTeX
encoded text	• “plain text” • word processor formats • page layout or desktop publishing formats • paper (using OCR Proofing or vendor) • PDF (possibly) • PostScript (possibly) • LaTeX (possibly)

Recommendations for `DocEncodingType` for serial publications

The default `DocEncodingType` for serial publications should be `serialarticle` to allow for maximum manipulation of content at the item, rather than issue, level. `Serialarticle` items are easiest to create for frontlist content but can also be created for reformatted material, especially in cases where one article ends and another begins on the same page.

In cases where the volume of scanning is too high to identify individual articles, `serialissue` should be used.