User's Guide for the Journal Publishing 3.0 Preview Stylesheets

Journal Information
Journal ID: N/A
ISSN: N/A

Article/Issue Information
Publication date: 2009

User's Guide for the Journal Publishing 3.0 Preview Stylesheets

Mulberry Technologies, Inc.

Introduction

These stylesheets format documents encoded in XML using version 3.0 of the NLM/NCBI Journal Publishing tag set. Previews in both HTML and PDF format (via XSL-FO) are supported. Most generic Journal Publishing structures are supported, with assumptions and limitations as described here.

This documentation, including this User Guide, the Quick Start Guide, and Technical documentation, are all included in this package encoded in NLM/NCBI Journal Publishing XML, with HTML and PDF presentation versions generated using the Preview stylesheets.

Assuming the user already has valid Journal Publishing 3.0 data, the most important consideration regarding application of these stylesheets regards the bibliographic citation format desired. This distribution provides for semi-automated formatting of citations in two formats: NLM/Pubmed, and APA. Users who have a requirement to display citations in another format, or who do not need any citation processing, should refer to the Citations section below.

For a more precise description of the technical dependencies of these stylesheets, along with documentation of alternative ways to deploy and apply them and how to extend and modify them, users should consult the Technical documentation.

Getting started

To select which stylesheet to use, you will first need to determine the following:

1. Do you want HTML/CSS, or PDF?
2. Are your citations in NLM/Pubmed format? APA format? (These are the only formats supported by these stylesheets.)
3. To make HTML:
   - If your citations use NLM/Pubmed style, you will use `jpub3-PMCcit-html.xsl`
   - If your citations use APA style, you will use `jpub3-APAcit-html.xsl`
4. To make PDF:
   - If your citations use NLM/Pubmed style, you will use `jpub3-PMCcit-xslfo.xsl`
   - If your citations use APA style, you will use `jpub3-APAcit-xslfo.xsl`

Then:

1. Apply the stylesheet to a valid Journal Publishing 3.0 document, or a subdirectory of such documents, using a recent version of Saxon (a freely available XSLT 2.0 processor).[1]

   When you run the stylesheet, messages will be echoed to your console reporting on progress. Do not be concerned by warning messages stating that you are running an XSLT 1.0 stylesheet with a 2.0 processor, since that is what you are doing. These stylesheets will run in XSLT "backward compatibility mode", and have been designed to work properly with either 1.0 or 2.0 processors.

---

[1] The stylesheets were developed and tested with Saxon 9.1.0.3 and Saxon 9.1.0.5. The extension function the shell stylesheets rely on is available in versions of Saxon back to 8.9.

2. When the result format is HTML, copy `jpub-preview.css` to the file system next to the result file(s).
3. When the result format is XSL-FO, run the result file in an XSL-FO formatter, according to its instructions.

   (Many XSL-FO formatters allow running an XSLT transformation directly to apply a stylesheet to a document. If you wish to do this and you are using one of the shell stylesheets provided, see to it that your formatter invokes Saxon as its XSLT component.)

Invoking SaxonB 9 from the command line

Saxon is a Java application and requires Java to be installed. Java 1.4 is required; at least Java 1.5 is preferred. Saxon 9 is also available within a number of different toolkits and XML editing environments, which provide their own user interfaces to the tool. If SaxonSA is used, the command line will be different. See the Saxon documentation at http://saxonica.com/documentation/index/intro.html for more details.

From the command line, SaxonB (the free version of the product) may be invoked as follows (on one line):

```
c:\Projects\NLM3\xmlfiles> java -jar c:\path\to\saxon9.jar
        -o mydocument.html mydocument.xml jpub3-APAcit-html.xsl
```

Within the `c:\Projects\NLM3\xmlfiles` subdirectory, this will generate an output document `mydocument.html` from the input `mydocument.xml`.

MS Windows path syntax is used in this example; adjust accordingly for your system.

Advanced options

In addition to the straightforward application of formatting to Journal Publishing content using either of the supported bibliographic citation styles, this suite of stylesheets supports a number of advanced options. Some of these work directly; others are included as demonstrations of how processes may be customized to meet local requirements.

In order to learn more, see the section on Special-purpose stylesheets below, along with the Technical documentation.

Assumptions made by the preview processing

These stylesheets are designed to handle a broad range of inputs, generally requiring only that input data be XML valid to the NLM/NCBI Journal Publishing 3.0 DTD. Inevitably, however, there are limitations in their handling of structures that are valid but unlikely and/or poorly controlled by a schema, and of variant patterns of tagging.

Caution: while these stylesheets do their best to handle "normal" input, they should not be taken as an indicator of a norm. "Variant" does not necessarily mean "incorrect", and where local practice has evolved to meet particular local processing requirements, these stylesheets may and should be altered or extended to accommodate them.

Known limitations of these stylesheets

Running page headers

The PDF Preview stylesheet generates running headers using the main article title (`article-title` inside `article-meta/title-group`). If the main article title is too long, an alternative title can be provided for the page header; if available, an `alt-title` with `alt-title-type` of 'running-head' will be used.

Super- and subscripting limitations

On `sub` and `sup` elements the `arrange` attribute is not supported in either preview. All instances will be treated as `arrange='stagger'`.

Addresses

The document model allows addresses (the `address` element) both as structured elements, and within inline content such as paragraphs.

The preview stylesheets assume that addresses are formatted as blocks broken into separate lines, one for each element in the address, *unless* the address contains no `address-line` elements *and* it appears in paragraph content (that is, directly within `p`, `license-p`, `collab`, `named-content` and `styled-content` elements).

Address elements inside `aff` (affiliation) are always formatted in line.

Sub-articles

The HTML preview stylesheet includes code for handling sub-articles, including the elements `sub-article` and `response`. Because the Journal Publishing tag set allows a wide range of ways to deploy and manage such content, particularly with respect to metadata and ancillary materials such as references and footnotes, these features have not been fully tested. If you have `sub-article` or `response` content, extending one or both stylesheets in view of your particular tagging patterns and requirements will probably be necessary.

Section metadata

The same thing is true of section-level metadata (`sec-meta`), particularly with respect to the `contrib-group` element inside `sec-meta`.

Also, within `sec-meta` in the PDF preview, only simple keywords (`kwd`) within `kwd-group` will be rendered; `compound-kwd` is ignored by these stylesheets.

Test the stylesheets against your content and be prepared to amend the stylesheets, if necessary.

Special characters

These stylesheets do not support the `private-char`, `glyph-data`, or `glyph-ref` elements.

Graphics

Graphics are not resized in the HTML preview, but displayed at native resolution. In the PDF preview, they may be scaled down to fit in available space, but smaller graphics that already fit will not be adjusted.

In the HTML preview, graphics are expected to be located as given in the source data relative to the HTML *result* file. So, if source data has a graphic with `xlink:href='snapshots/babypic.jpg'`, the result file will link to 'snapshots/babypic.jpg'. If graphic files are maintained relative to the source data, they should either be copied to the same location relative to the result, or an absolute path to the graphic should be given as the value, or (most simply) the HTML result file should be placed next to the XML source file, in order to ensure that graphics will be visible in the browser.

In the PDF preview, a runtime parameter, `base-dir`, must be given to identify the directory relative to which graphics are located. If the XSL process is initiated using one of the provided XSLT 2.0 shell stylesheets, this parameter will be provided automatically as the base directory of the source file, so graphics should be located relative to the source data. If no parameter is given, either graphics must be designated with absolute path names, or they must be located relative to the *stylesheet*.

Alternative text on graphics provided as `alt-text` are not rendered in the PDF preview.

Styled content and the HTML `style` attribute

The Journal Publishing document model makes provision for styling content directly by using CSS styles, either associated with table elements (HTML table elements in this model) or as the value given the `style` attribute of the `styled-content` inline element.

When this appears, the HTML preview stylesheet will simply pass the values through. Results will depend on the level of CSS conformance both in the code and in the browser.

In the PDF preview, `width` and `vertical-align` values associated with tables are mapped to formatting in the result. On any element with the `style` attribute, values of the following properties are mapped: `color`; `background-color`; `font-size`; `font-weight`; `font-style`; `font-family`; `text-decoration`. Note that incorrect CSS values may result in errors when XSL-FO stylesheet results are formatted.

Note in particular that CSS properties related to borders are not supported in the PDF preview. For table cell borders, some bordering behavior can be specified using attributes on the `table` element.

Floating objects and the `position` attribute

Floating behavior actually entails two different functionalities: allowing a formatting application to determine (and presumably optimize) the position on the page of objects whose position is not to be anchored in the flow of text; and managing objects out of line (probably but not necessarily so they can be so placed) by gathering them into a group at the end of the document.

The first feature in Journal Publishing tagging is controlled through the use of `position` attributes on objects that may be placed in this way by the formatter. These elements include `boxed-text`, `chem-struct-wrap`, `fig`, `fig-group`, `graphic`, `media`, `preformat`, `supplementary-material`, `table-wrap`, `table-wrap-group`. The second feature is supported by means of the `floats-group` element, where such objects may be gathered.

Note that `position='float'` may be assigned to any of these objects irrespective of whether it is gathered into `floats-group`. Similarly, it is possible (although not ordinary) for an object to be placed inside `floats-group`, and not in the body of the document, without assigning it `position='float'`.

In this context, it is especially important to note that by default, on all these objects, `position` is assigned a value of 'float' by the Journal Publishing DTD. If the attribute is not set in the data, the formatting stylesheets assume the value should be 'float', both out of respect for the semantics specified by the formal document model, and in order to prevent different formatting when the DTD is used from when it is not.

Any object assigned `position='float'` will be allowed to float on the page. Exceptions to this rule include the following:

- No object, even one set to float, will float if it appears inside another object set to float. Note again that objects are set to float implicitly, unless position is set to 'anchor' or 'margin'.
- `graphic` and `media` will not float when they appear inside any of the other elements that may float. (A `graphic` inside a `fig` will not float, even if its `position` is set to float and irrespective of whether the `fig` floats.)
- `fig` will not float inside `fig-group` (though the `fig-group` may float).
- `table-wrap` will not float inside `table-wrap-group` (though the group may float).

- `preformat` will not float inside `bio`, `boxed-text`, `chem-struct`, `chem-struct-wrap`, `disp-formula`, `disp-quote`, `fig`, `glossary`, `supplementary-material`, `disp-formula`, or `table-wrap`. Note, however, that like other floatable objects, `preformat` will ordinarily float unless its position says not to!

Also, the value 'margin' is not supported; objects with `position='margin'` will act as if they have `position='anchor'`.

In the HTML preview, floating an object will not move it on the page; it will only allow text to flow around it.

In the PDF preview, objects with `position` = 'float' will be positioned on the page by the formatting engine, typically at the top of the same page where the object would appear if its `position` were 'anchor'. The exact placement will depend on the formatting engine.

Where floating is not wanted, the easiest solution is to set `position` to 'anchor' on the element.

Gathering objects in `floats-group`

Just like objects elsewhere, the placement of objects tagged inside `floats-group` is determined by their `position` attribute. The difference is that when assigned `position='float'` (and remember this is the value by default), in the PDF preview the object is formatted near its first cross-reference. Otherwise, and when no such cross-reference appears in the document, it is formatted at the end of the document.

The HTML preview stylesheet displays all objects inside `floats-group` at the end of the document. (In this case, however, cross-references are also hyperlinked.)

When articles should be formatted with a "Figures" or "Tables" section at the end, an `app` or `sec` should be used for this purpose, and the objects anchored in place. The preview stylesheets assume that `floats-group` is specifically for collecting objects that are intended to be formatted elsewhere.

Correct numbering of floating objects

As described below, the preview stylesheets do not provide automated numbering for floating objects (distinguished, as a class, from footnotes, which may also "float"). In systems where the stylesheet is extended to provide autonumbering, rules will need to be formulated and followed to govern where floating objects are encoded, whether and where they are cross-referenced, and the relation between settings of `position` and appearance (and order) inside `floats-group`, with appropriate enforcement of these constraints. Otherwise it can be expected that their numbers will not always be properly sequenced.

Setting `orientation`

A number of objects also have an `orientation` attribute, which is intended to allow specifying that they should be rotated in display, by setting its value to 'landscape'.

The HTML preview stylesheet does not support orientation. The PDF stylesheet does rotate the object; however, it assigns (somewhat arbitrarily) a width (i.e., a *vertical* spacing for an object formatted in landscape) of 6 inches for tables and 4 inches for other objects. While this will almost never be optimal, it is the best that can be accomplished without particular formatting specifications for each object, and it is enough to show that the orientation is set. Users may wish to adjust the stylesheet to meet local needs for positioning and sizing tables and figures.

Sizing tables

In HTML, tables are sized dynamically or according to attributes given on the table element.

In the PDF Preview, by default, tables are formatted at column width. If the `width` attribute on a table is set to "100%" and `orientation` is "portrait" (the default), the entire `table-group` it appears in will be formatted at page width.

Footnotes

The Journal Publishing model supports either of two general approaches to encoding footnotes in valid data. The `fn` element may be used in line where a reference to a footnote (generally the first or only reference) should appear, with the footnote content moved in rendition (typically to the bottom of the page). Or footnotes are encoded together in a structured grouping, the `fn-group` element, and footnote references in the text are encoded with `xref` elements, like any other cross-reference. The latter style of encoding is said in the Journal Publishing documentation to be appropriate for "end notes" as opposed to footnotes *per se*. `fn-group` is permitted in a number of places, including within the article back matter and in several other locations where footnotes are commonly grouped in journal articles.

These stylesheets will support either approach to encoding footnotes. In the HTML preview output, all footnotes encoded inline are grouped together in a section called "Notes" appearing at the end of the document. In the PDF, footnotes encoded inline are formatted at the foot of the page. In both cases, footnotes grouped in `fn-group` or in `author-notes` appear collected together where the `fn-group` or `author-notes` appears. This will usually be inside the back matter, but it can also occur in other places such as in the front matter, `boxed-text` or `table-wrap-foot`.

Note that the two approaches (loose `fn` elements and `fn` within `fn-group`) may be combined, but this is inadvisable unless footnotes are numbered in the data, since it may result in two distinct sets of footnote numbering. It is best to encode footnotes all one way or all the other. Generally, if you use `fn` elements within mixed content, do not also use `fn-group`. If you use `fn-group`, use it for all footnotes and use `xref` to provide references to them.

Additionally, in the PDF preview, errors will result if inline footnotes appear inside floating objects (such as boxed text or figures). This is because the XSL-FO formatter cannot position both a footnote and a floating object together (since when a footnote is moved to appear on the same page as a float, the page layout determining where the float can be placed may also be thrown off). This restriction means, in effect, that if you have footnotes in your floating objects, you must locate the footnotes inside an `fn-group`, and reference them with `xref`, rather than using `fn` elements inline.

With regard to their numbering, footnotes are described further below.

Autonumbering, labels, and cross-references

Except for two element types, `ref` (reference) and `fn` (footnote), these preview stylesheets do not generate numbered labels automatically. If figures, tables and similar structures such as statements, supplementary material, etc. are to be labelled, they must be labelled in the data. Labels provided in the data (`label` elements) are displayed with the elements they label; the same label content also appears in cross-references that point to those elements when a cross-reference element (`xref` elements) is given as empty. (When `xref` elements have text content, it is used as the text of the cross-reference, irrespective of whether a label appears with its target.)

The approach to labelling taken by these stylesheets is meant to be as transparent and easy to understand as possible: it assumes that when labels are wanted on content, they will be provided explicitly in the source data (and therefore should not be generated automatically), and that when a cross-reference is made to an object, it will either have content of its own (which can serve as anchoring text), or its target will have an explicit label that can be used unaltered as the text in the cross-reference.[2]

There are two exceptions to this principle: numbering is be provided for footnotes (`fn` elements) and for references (`ref` elements) when they do not have explicit labels, as these elements' primary purpose is their systematic cross-referencing. In both cases, explicit labels (`label` elements or, in the case of footnotes, `symbol` attributes) will override autogenerated labels, allowing for projects to control these values in their data when necessary. In order to protect the consistency of labeling, warning messages may be generated for unlabelled elements where labels were expected. In certain places, such as within the document metadata (`author-notes` or `fn-group` inside `title-group`), unlabelled footnotes are permitted in the PDF preview, and will not generate warning labels even if other footnotes have labels.

Automated numbers are assigned based on the order of footnotes in the document, not of first references to them. Similarly, in the PDF output, since footnotes are placed based on the location of the `fn` element in the document, a cross-reference can be made to a footnote that will appear on a subsequent page. This also means that if footnote A appears before footnote B in the document, a reference to footnote B will appear out of sequence if it appears before footnote A does. The easiest ways to avoid this are (a) to use footnotes as endnotes (group them in `fn-group` in the back matter), or (b) make sure that if inline footnotes are used (resulting in actual footnotes in the PDF), they are not placed before cross-references to the same footnote.

These stylesheets do not accommodate a practice of redundant tagging, whereby a footnote encoded inline is also provided with an `xref` (cross-reference) element to that footnote. If an `fn` element is used in line, a reference will appear for it. Cross-references to footnotes should be used only for end notes (`fn` elements in `fn-group`) or in cases where more than one reference is made to a single footnote.

Summary of best practice respecting cross-references and labels

For correct operation with these stylesheets, the simplest approach to using labels and assuring that cross-references are clear is to follow an "all or nothing" rule:

- As described above, either all footnotes (`fn` elements) should be enclosed in `fn-group` elements (effectively, to present as endnotes), or all should be placed inline at their point of first reference.

  Avoid using (empty) `xref` elements to refer to footnotes that have not yet appeared, unless the footnote is inside `fn-group`, as this will result in numbering out of sequence. Footnote numbers are assigned in the order of the footnotes, not of first reference to them. (This will not matter if footnotes are labelled in the source data or if `xref` elements have data content, in which case maintaining correct referencing is an editorial task.)
- Either label all footnotes explicitly in the source data (using `label` elements or `symbol` attributes), or let the system number all of them.
- Similarly, no reference elements (`ref`), whether they are cross-referenced or not, should be given `label` elements, or all of them should.
- Any object type (such as figures or tables) that is cross-referenced should be given `label` elements consistently and throughout, *or* cross-references (`xref` elements) should never be empty.

As described in the Technical documentation, any of the stylesheet logic regarding generating labels and/or automatic numbering can be modified in the stylesheets.

---

[2] For projects that wish to exploit the potentials of automated processes for systematic numbering, the stylesheet does contain code that can be configured to generate labels, including autonumbering. See the Technical documentation for more information.

Warning messages for missing labels

While (apart from the complications of footnotes and references) this logic is fairly simple, an error condition may still arise if an object has no `label` element, and a cross-reference (`xref`) pointing to it has no content of its own. That is, while either its own content or its target's label can provide text for a cross-reference, a particular cross-reference might have neither (even in a valid document). When this happens, a warning message will be generated in place of a label, both with the object and wherever it is cross-referenced. A summary view of all such warning messages is also provided in a special section entitled **Process Warnings** at the end of the document.

This error is corrected either by providing the targeted object with a `label` element (or alternatively, for a footnote, a `symbol` attribute), by providing its cross-reference(s) with content, or both.

When there are no such errors, the **Process Warnings** section does not appear.

Other limitations on `xref` elements

The current implementation of the preview stylesheets does not support cross-references to more than one target using a single `xref` element. Although according to the DTD, an `IDREFS` value is allowed on `xref/@rid`, only a single `IDREF` will work.

In other words,

```
<xref rid="fig1">See Figure 1</xref>
```

will generate a cross-reference in the output, but

```
<xref rid="fig1 fig2">See Figures 1 and 2</xref>
```

will not work correctly (it will attempt to link to an object identified as "fig1 fig2").

Citations

Journal Publishing XML may tag bibliographic citations in any of several ways, including the `element-citation` element, which assumes that rendering of citations, including punctuation, is provided by a stylesheet. Stylesheets in this package are capable of rendering of `element-citation` into two different bibliographic formats.[3]

- **NLM/Pubmed format**. The stylesheet provided here is a modified version of a transformation used internally at NCBI.
- **APA-like format**. A stylesheet for generating common citation styles conforming to APA guidelines is also provided. We call this format "APA-like" in order to stress that results may not correspond precisely either to local rules regarding details of APA format or in some cases to APA's own guidelines.

If your citations follow neither practice, or when your citations do not format properly due to special cases, you have two alternatives:

- Instead of `element-citation` in bibliographies or reference lists, use `mixed-citation` with explicit formatting. This element allows citations to be formatted and punctuated directly in the XML data. Neither preprocessing stylesheet will override any punctuation within `mixed-citation` elements.

---

[3] Note that even in the best of cases, formatting of bibliographic citations according to truly generalized and systematic rules is not entirely possible to specify and therefore to automate. Testing shows that while these stylesheets will handle the majority of common cases properly, adjustments always have to be made. When this occurs with the provided stylesheets, users have the same options as if they were posed with a format that was not implemented: either extend the stylesheet or fall back on explicit formatting using `mixed-citation`.

Even if all citations are given as punctuated `mixed-citation` elements, however, it is still necessary to run one or another citations preprocessor. This is because they may still contain elements with formatting consequences, which the preview stylesheets have no rules to handle. The only citations that are safe to run through the preview stylesheets without any preprocessing whatsoever conform to a profile of `mixed-citation` that includes only elements whose formatting consequences are inherent in their semantics,[4] and will not vary from one citation formatting style to another. See the Technical documentation on the "Simple citation" element profile for more details.

- Extend or modify one of the provided citation processors, or develop your own citation preprocessor implementing your own rules for formatting citations.

If you have no citations, you can run the stylesheets for either format.

Special-purpose stylesheets

There are three types of stylesheets included in this package:

- The main Preview stylesheets (found in the `main` subdirectory);
- Various ancillary stylesheets for pre- or post-processing data in support of special operations (in the `prep`, `citations-prep` and `post` subdirectories);
- "Shell" or "wrapper" stylesheets (here called "top-level stylesheets") to be used to invoke the others in combination. These are the stylesheets that will typically be applied directly to documents, as described above.

Users wanting a simple approach can acquire a recent version of the Saxon XSLT engine, identify the top-level stylesheet that performs the necessary tasks, and apply it to their data.[1]

The top-level stylesheets depend on the Saxon processor because they use XSLT 2.0 features and Saxon extension functions to perform "pipelining", efficiently executing a sequence of transformations without writing interim results to the file system. All of the stylesheets can also be run on their own, and some of them work also in XSLT 1.0 engines. Due to limitations of the main Preview stylesheets, especially limitations in their handling of citations (which cannot be handled generically), this is recommended only in special cases within controlled environments where citation formatting is otherwise accounted for.

Nevertheless, publishers and content creators are free to experiment. This model also serves to provide for extensions, in the form either of new processing steps (such as a preprocessor for a new citation format) or modifications to existing steps. Or projects may wish to "unbundle" the processes and apply the stylesheets by different means altogether. The Technical documentation describes these alternatives in more detail.

Top-level stylesheets included in this package are as follows:

`jpub3-PMCcit-html.xsl`
: Generates HTML results with formatting of NLM/Pubmed citations supported.

`jpub3-APAcit-html.xsl`
: Generates HTML results with formatting of APA citations supported.

`jpub3-PMCcit-xslfo.xsl`
: Generates XSL-FO results for PDF production, with formatting of NLM/Pubmed citations supported.

`jpub3-APAcit-xslfo.xsl`
: GeneratesXSL-FO results for PDF production, with formatting of APA citations supported.

---

[4] I.e., elements such as `italic` and `sup` (superscript).

`jpub3-PMCcit-web-html.xsl`
: This works like `jpub3-PMCcit-html.xsl` except it includes a preprocess that removes all elements marked with a `specific-use` attribute whose value is "print-only". This stylesheet works by invoking an additional preprocessing step, `prep/jpub3-webfilter.xsl`. It may be useful mainly as a demonstration, illustrating how processes can be customized to meet local requirements as described in the Technical documentation.

`jpub3-PMCcit-xhtml.xsl`
: This also works like `jpub3-PMCcit-html.xsl`, except it includes a postprocess that converts the HTML results into XHTML, resulting in XHTML results that can be correctly rendered by MathML-capable browsers. The postprocessor called by this stylesheet, `xhtml-ns.xsl`, can be applied as the final step in any pipeline that must generate XHTML, as also described in the Technical docs.

`jpub3-PMCcit-print-fo.xsl`
: This shell is the complement of `jpub3-PMCcit-web-html.xsl`. It formats citations in NLM/Pubmed citation format, removes all elements marked with a `specific-use` attribute whose value is "web-only", and then transforms the document to XSL-FO results, ready for formatting as PDF by an XSL formatter.