Users' Guide for the NISO JATS 1.0 Preview Stylesheets

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

Article/Issue Information
Publication date: 2012

Users' Guide for the NISO JATS 1.0 Preview Stylesheets

Mulberry Technologies, Inc.

Introduction

The NISO JATS 1.0 Preview Stylesheets package includes stylesheets that produce formatted previews of documents encoded in XML using the NISO JATS (Journal Article Tag Suite) vocabulary. Previews in both HTML and PDF format (via XSL-FO) are supported. Most generic Journal Publishing structures are supported, subject to the assumptions and limitations described in this document.

The primary purpose of the stylesheets is to support version 1.0 of the Publishing ("blue") tag set of NISO JATS, but they will also work on data valid to the related tag sets JATS 1.0 Authoring ("orange") and NLM Journal Publishing ("blue") or Authoring ("orange") versions 2.3 and 3.0.

The documentation included in this package, namely this *Users' Guide*, the *Quick Start Guide*, and the *Technical Documentation*, can serve as examples of the tagging style supported; they are all encoded in NISO JATS Publishing XML, with HTML and PDF presentation versions generated using the preview stylesheets.

Assuming the user already has valid JATS (Publishing) 1.0 data, the most important consideration regarding application of these stylesheets regards the format of bibliographic citation information in the input and the bibliographic format desired for those citations in the output. Citations tagged using `mixed-citation` and provided with appropriate punctuation in the input (called here "formatted citations") are handled correctly by the stylesheets in this package. Citations tagged without punctuation, using `element-citation` ("unformatted citations") can also be handled: this distribution includes stylesheets which format such citations in the user's choice of two formats: NLM/Pubmed, and APA.

In addition, this package also demonstrates how pre- and post-processing steps connected together in pipelines may achieve other processing goals, both preview and production. These include processing OASIS (CALS) tables and handling tagging intended to optimize outputs for different media (such as web and print).

For a more precise description of the technical dependencies of these stylesheets, along with documentation of alternative ways to deploy and apply them and of how to extend and modify them, users should consult the *Technical Documentation*. That document also includes a fuller specification of the formatted and unformatted citation formats supported by this package.

Getting started

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

1. If you have unformatted citations in your input, do you want the output to be in NLM/Pubmed format or APA format? (These are the only formats these stylesheets can produce from unformatted citations in the input.)

   If all the citations in the input are formatted, you can use either the NLM/Pubmed or the APA stylesheets and pipelines mentioned below.

   If you have no citations or if your citations contain only simple inline tagging with punctuation,[1] you can forego citation formatting altogether.

2. Do you want HTML (with CSS), or PDF?
3. Which of the following methods do you prefer to use for processing the input?
   - An XProc (XML Pipelining) processor such as XML Calabash?

     If using XProc, you will select a pipeline specification from the `shells/xproc` subdirectory.
   - A copy of Saxon that supports the necessary extension functions?[2]

     If using Saxon, you will select a shell stylesheet from the `shells/saxon` subdirectory.
   - A different XSLT processor?

     Only the Saxon pipelines require Saxon; the stylesheets themselves will work in any conformant XSLT processor. Many of the stylesheets are in XSLT 1.0 and will work in a 1.0 processor (or 2.0) processor; some require XSLT 2.0.[3] You can run pipelines either "by hand" (run several stylesheets each on the results of the previous one) or using a custom-built pipelining method (shell or batch script, Java Ant, `make`, etc.). If you don't need to run a pipeline, you can also run XSLT 1.0 stylesheets (such as the HTML Preview stylesheet) in a web browser that supports XSLT 1.0.
4. To make HTML:
   - If NLM/Pubmed style should be used to render unformatted citations in the input, you will use `shells/xproc/jats-PMCcit-html.xpl` or `shells/saxon/jats-PMCcit-html.xsl`.

     If you are running without Saxon or XProc, you will first apply `xslt/citations-prep/jats-PMCcit.xsl` to your input, and then apply `xslt/main/jats-html.xsl` to the output of the first transformation.
   - If APA style should be used for unformatted citations in the input, you will use `shells/xproc/jats-APAcit-html.xpl` or `shells/saxon/jats-APAcit-html.xsl`. Since the APA citation style transformation requires XSLT 2.0, you will need an XSLT 2.0 processor to run it without Saxon or XProc.
   - If all citations in the input are formatted, you can use either the NLM/Pubmed or the APA pipeline; if they use only simple inline tagging, you can use `xslt/main/jats-html.xsl` without pipelining.[4]
5. To make PDF:
   - To generate NLM/Pubmed style citations from unformatted citations in the input, XProc users will use `shells/xproc/jats-PMCcit-xslfo.xpl`.

     Saxon users will use `shells/saxon/jats-PMCcit-xslfo.xsl`.

     XSLT 1.0 users will use `xslt/citations-prep/jats-PMCcit.xsl` followed by `xslt/main/jats-xslfo.xsl`.
   - To generate APA-style citations from unformatted citations in the input, XProc users will use `shells/xproc/jats-APAcit-xslfo.xpl`.

---

[1] As described in the *Technical Documentation* under "Simple citation" format.

[2] Since version 9.2, the open-source version of Saxon (Saxon HE) does not support the needed extension function, so you will need Saxon PE or EE. Before version 9.2, the extension functions required by the Saxon shell stylesheets was available in versions of Saxon back to 8.9.

[3] Consult the *Technical Documentation* for details on which stylesheets require XSLT 2.0. (Apart from the Saxon shell stylesheets, most do not.)

[4] See the *Technical Documentation* on these stylesheets' specification of a "Simple Citation" format for more information on what tagging in citations will be handled by default, without customization.

Saxon users will use `shells/saxon/jats-APAcit-xslfo.xsl`.

XSLT 2.0 users will use `xslt/citations-prep/jats-APAcit.xsl` followed by `xslt/main/jats-xslfo.xsl`.

XSLT 1.0 users cannot produce APA-style citations automatically, since the first of these stylesheets relies on XSLT 2.0.

- If all citations in the input are formatted, either of the the NLM or the APA pipelines can be used, or `xslt/main/jats-xslfo.xsl` alone can be used if they are formatted with simple inline tagging.[4]

Then:

1. Apply the stylesheet(s) or pipeline to a valid Journal Publishing 3.0 document, or a subdirectory of such documents, using the tool of your choice.

   When you run your pipeline, messages may 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. These stylesheets will run in XSLT "backward compatibility mode", and have been designed to work properly with either 1.0 or 2.0 processors.
2. When the result format is HTML, copy `jats-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 the correct processor (such as Saxon or an XProc engine) as its XSLT component.

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. While these can be used "out of the box", they are also included as demonstrations of how processes may be further tailored and 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 of variant patterns of tagging and of structures that are valid but unlikely and/or poorly controlled by a schema.

Caution: while these stylesheets do their best to handle "normal" input, *they should not be taken as an indicator of a norm*. "Variant" here 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 local practice and requirements.

The remainder of this section outlines the main limitations of the stylesheets and the major assumptions they make about their input, grouping the entries into sections on known limitations; autonumbering, labels, and cross-references; and citations.

Known limitations of these stylesheets

Users should keep in mind that any and all of these behaviors and limitations (as well as those not described) can be addressed directly in customizations of these stylesheets; they are not inherent in XSLT (although sometimes the target format, HTML or XSL-FO, presents challenges), but only a result of the design requirement that the Preview stylesheets address the broadest possible set of "reasonable" approaches to tagging NISO JATS, without privileging any in particular.

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.

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

If 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), then it is formatted in line.

Otherwise, the address is formatted as blocks broken into separate lines, one for each element in the address.

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; this may result in disproportionately large images when the images have been made large to give them higher resolution. In the PDF preview, images will be scaled down (as necessary) to fit in available space.

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 else 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, then either graphics must be designated with absolute path names, or else they must be located relative to the *stylesheet*.

Alternative text provided as `alt-text` on graphics is 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 in the browser and in the CSS values passed through to the output.

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

In the parlance of document markup, "floating" can refer to two distinct things. If a rendering application is trusted, at runtime, to determine (and presumably optimize) the precise position on the page of an object, this object is said to "float". A figure, for example, may be allowed to float, while a graphic appearing in a paragraph is not (it is anchored to its place in the flow of text); similarly a boxed text may be allowed to float, or it may be anchored, depending on whether its exact placement on the page is important to its meaning. But the term is also used to refer to objects (such as figures and tables) that are encoded in the source data -- but not necessarily rendered -- apart from the flow of the text, typically by gathering them into a group at the end of the document. Such "floating" objects may be displayed there, or moved by the application to fixed (anchored) positions, or allowed to "float" in the first sense.

The first feature in JATS 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'`.

Note also 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 '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 could float, even if they don't in fact float. (For example: a `graphic` inside a `fig` will not float, even if its `position` is set to float and even if the `fig` does not float.)
- `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 as a whole 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.

Note that when a float group is large (e.g. a table with footnotes provided both in table form and as a graphic), it may not all fit on a single page.

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 rendering (typically to the bottom of the page). Or footnotes may be 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 as such. `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.

For discussion of footnote numbering, see 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.[5] 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` element) is given as empty. (When an `xref` element has text content, it is used as the text of the cross-reference, irrespective of whether a label appears with its target.)

This approach is meant to be as transparent and easy to understand as possible: rather than assume any autolabeling method and format, it assumes that when labels are wanted on content, they will be provided explicitly in the source data, 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.

There are two exceptions to this principle: numbering is 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, and in most instances they will not function without labels. 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 all cases, to ensure legible and transparent output, warning messages may be generated for unlabelled elements where labels are needed (due to an empty cross-reference pointing to the element). This is described further below.

When generated for footnotes and references, automated numbers are assigned based on the order of their appearance in the document, not of first references to them. (Nor are `ref` elements sorted to reflect their order of first reference.) Accordingly, 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 different page. This also means

[5] 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.

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 both these problems are (a) to use footnotes as endnotes (group them in `fn-group` in the back matter) and confirm that they are placed in order of first reference; or (b) use inline footnotes consistently (resulting in actual footnotes in the PDF) and make sure that in every case, an `fn` element appears *before* any other (second or subsequent) reference 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

In working with these stylesheets, the simplest approach to using labels and ensuring 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), in order of first reference, *or else* all footnotes 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.

  Of course, such forward references are not a problem and need not be avoided if footnotes are labelled in the source data, or if `xref` elements have data content; in that case, of course, it is an editorial task to maintain correct referencing and numbering.
- *Either* label all footnotes explicitly in the source data (using `label` elements or `symbol` attributes), *or else* let the system number all of them.
- Similarly, *either* no reference elements (`ref`), whether they are cross-referenced or not, should be given `label` elements, *or else* all of them should.
- *Either* every object type (such as figures or tables) that is cross-referenced should be given `label` elements consistently and throughout, *or else* cross-references (`xref` elements) should never be empty.

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

Warning messages for missing labels

The logic outlined above is fairly simple (apart from the complications of footnotes and references), but an error condition may still arise (even in a valid document) if an object has no `label` element, and a cross-reference (`xref`) pointing to it has no content of its own. 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 can be corrected (1) by providing the targeted object with a `label` element (or alternatively, for a footnote, a `symbol` attribute), (2) by providing its cross-reference(s) with content, or (3) by doing 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. One of those ways uses 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 `element-citation` into two different bibliographic formats.[6]

- **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.

  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,[7] 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.

---

[6] Note that even in the best of cases, it is not possible to automate the formatting of arbitrary bibliographic citations completely and systematically. Testing shows that while these stylesheets will handle the majority of common cases properly, adjustments often have to be made. When this occurs with the provided stylesheets, users have the same options as they have when they need an output format that has not been implemented: either extend the stylesheet to produce the desired output, or else fall back on manual formatting using `mixed-citation`.

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

Special-purpose transformations

There are four types of resources 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`, `oasis-tables` and `post` subdirectories);
- XProc pipelines that implement more comprehensive transformations by connecting two or more of these stylesheets in sequence. These can efficiently execute a sequence of transformations without writing interim results to the file system.
- "Shell" or "wrapper" stylesheets used to the same effect, by relying on extended functionality in the Saxon processor (for those who have a recent version of Saxon[2] and do not wish to use XProc).

Users wanting a simple approach can acquire an XProc processor, or a recent version of Saxon, identify the pipeline or shell stylesheet that performs the necessary tasks, and apply it to their data.

All of the stylesheets can also be run on their own, and several 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 when citation formatting is otherwise accounted for.

Publishers and content creators are free to experiment. The structure of this distribution is intended 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 pipelines included in this package are as follows (each available in two forms, XProc and extended XSLT 2.0):

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

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

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

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

`jats-PMCcit-web-html`
: This works like `jats-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/jats-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*.

`jats-PMCcit-xhtml`
: This also works like `jats-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 Documentation*.

`jats-PMCcit-print-fo`

This shell is the complement of `jats-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.

`jats-oasis-PMCcit-fo`

This shell excludes elements with a `specific-use` attribute of "web-only", processes OASIS tables, formats citations in NLM/Pubmed citation format, and transforms the document to XSL-FO results for conversion to PDF.