Journal of the Text Encoding Initiative

Issue 13 | May 2020 - November 2022
Selected Papers from the 2018 TEI Conference

Using ODD for HTML

Martin Holmes

Electronic version

URL: <https://journals.openedition.org/jtei/3106>
DOI: 10.4000/jtei.3106
ISSN: 2162-5603

Publisher

TEI Consortium

Electronic reference

Martin Holmes, "Using ODD for HTML", *Journal of the Text Encoding Initiative* [Online], Issue 13 | May 2020 - November 2022, Online since 15 May 2020, connection on 24 November 2022. URL: <http://journals.openedition.org/jtei/3106> ; DOI: <https://doi.org/10.4000/jtei.3106>

For this publication a Creative Commons Attribution 4.0 International license has been granted by the author(s) who retain full copyright.

Using ODD for HTML

Martin Holmes

ABSTRACT

Although the ODD (One Document Does it all) language is normally used to create TEI customizations or extensions, it is also a highly effective tool for editors working in other XML markup languages. This paper will discuss the use of ODD to define a highly constrained schema for HTML5 that will enforce stylistic rules and encoding practices, define custom attributes and value lists, and enable easier editing and validation of project content in the Oxygen XML Editor environment. I will provide a brief history of the project, whose first incarnation, created with the Dreamweaver HTML editor, was somewhat chaotically coded, and show how the implementation of an ODD-based schema provides huge advantages for authors, editors, and encoders, as well as substantially simplifying the code itself.

INDEX

Keywords: ODD, HTML, non-TEI projects,

1. Introduction

- 1 Although ODD (One Document Does it all) is a feature of the TEI language, and used primarily for creating TEI schemas, ODD in fact “goes beyond this to provide a generic tool for the documentation and management of any XML encoding scheme, not necessarily one based on the TEI” (Burnard and Rahtz 2014). Syd Bauman (2019) points out that the TEI ODD language “can be used for two related but distinctly different purposes: (1) to *create* a markup language, including documentation and schemas; and (2) to *customize* a markup language that was already written in ODD.” This paper describes a use case which does not quite fall into either of those categories: the use of ODD to create and document a highly constrained customization of a markup language not originally written in ODD. The language in this case is HTML5, in its XHTML serialization.¹
- 2 In 2017, our unit was approached by Kim Blank, a faculty member who had for some years been building a fascinating website called *Mapping Keats’s Progress*.² The website is in one aspect a biography of the poet John Keats, but it has many other features. Blank describes its purposes as follows:
 - To map some of Keats’s life in London
 - To account for Keats’s remarkable poetic development
 - To re-imagine the book

and explains the third point in these terms:

The site’s structure of *progressive reduplication* (between multiple, overlapping micro-chapters) acknowledges and attempts to embrace the fact that the dominant means to access information—via the technology you have in front of you right now—changes the way we find, look at, and engage such information.

(Blank 2018)

- 3 The site maps out the life of John Keats by year, month, and day, with entries for specific days in the form of short articles, along with yearly summaries. The “progressive reduplication” technique involves repetition of core information across multiple articles, so that any reader jumping into the site at any point will find enough information to make the article they are reading completely comprehensible. This acknowledges the fact that web-based materials are

- rarely consumed serially, but rather sampled somewhat at random, so each page must be to some extent self-contained. Balancing this requirement against the need to cater also to the more assiduous reader who may read many articles in sequence is a difficult feat of style and authorship.
- 4 The site had been developed by the author and a collaborator using the Dreamweaver web-authoring software. The researcher initially asked for help with a single problem: the fact that the site banner was slightly different in appearance on some pages than on others, and the effect of navigating through the site was slightly jarring as the banner shifted around from page to page. Examination of the code quickly revealed that the code for the banner was actually different on almost every page on the site. A deeper investigation determined that this was merely the tip of a vast iceberg of source-code chaos. Although its interface and arrangement were functional and attractive (see [figure 1](#)), the HTML code had become a huge mass of incomprehensible nested structures, including twenty-four JavaScript and sixty-nine CSS files. A rather haphazard approach to development had resulted in a tendency to add new features as they occurred to either collaborator, usually by taking example code or pre-built JavaScript and CSS libraries from the Web and dropping them into the project. An example of the kind of needless complexity that had resulted from the dependence on a WYSIWYG tool to manage style and layout can be seen in [figure 2](#). Most developers will have encountered projects like this frequently, and will be familiar with the bleak anguish that afflicts a coder who inherits such a codebase.
 - 5 In the fall of 2017, I began the process of rewriting it, with the aim of keeping it as simple as possible while reproducing and enhancing the design and functionality. The result has only one CSS file and a few hundred lines of JavaScript, none of which is essential.

Figure 1. The site as it currently appears (not substantially different from the original site design).

Figure 2. The long journey through fourteen nested <div> s to the <h1> element in the original Dreamweaver-generated HTML.

```

<div class="p7TP3cwrapper_03">
  <div id="p7TP3cwp_1" class="p7TP3_vp">
    <div id="p7TP3pw_1" class="p7TP3_slide_wrapper">
      <div id="p7TP3w1_1" class="p7TP3_panel">
        <div id="p7TP3c1_1" class="p7TP3_content_03 p7TP3content">
          <div class="p7tp3-col-wrapper no-columns"> </div>
        </div>
      </div>
    <div id="p7TP3w1_2" class="p7TP3_panel">
      <div id="p7TP3c1_2" class="p7TP3_content_03 p7TP3content">
        <div class="p7tp3-col-wrapper no-columns">
          <div id="p7AP3_1" class="p7AP3-02 p7AP3 responsive">
            <div id="p7AP3tb_1" class="ap3-toolbar closed"><a href="#" title="Hide/Show Menu">&equiv;</a></div>
            <div id="p7AP3rw_1" class="p7AP3root-wrapper closed">
              <div class="p7AP3trig p7ap3-theme-02">
                <h3><a href="#p7AP3c1_1" id="p7AP3t1_1">16 March 1816: Keats, Joseph Severn, and Chivalric Infatuations </a></h3>
              </div>
            <div id="p7AP3w1_1" class="p7AP3cwrapper p7ap3-theme-02">
              <div id="p7AP3c1_1" class="p7AP3content p7ap3-theme-02">
                <div id="p7AP3p1_1" class="p7AP3panelcontent p7ap3-theme-02">
                  <div class="p7ap3-col-wrapper multi-columns">
                    <div class="p7ap3-column width-50">
                      <div class="p7ap3-column-content p7ehc-2">
                        <h1>6 Goswell Street Road, London</h1>

```

2. Why Not Use TEI?

- 6 The instinctive response of a seasoned TEI encoder to content like this is to get it into TEI as soon as possible, and then build a rendering toolchain to create a fresh website. However, while the original HTML was severely out of control, the content itself was already basically complete, and encoded in HTML. Project participants were quite comfortable with HTML and preferred it as their master format for both editing and final result. They were not interested in developing a system to export the same content in different formats, so for their needs, moving the master files to TEI would be a gratuitous impediment to their work.
- 7 I was able to use a toolchain consisting of [HTML Tidy](#),³ XSLT, and Python to clean up and simplify the content to a point where it required only some proofing and enhancement. Since the project author was already familiar with HTML, but not with TEI, and the markup itself was relatively simple, it seemed easier to stick with HTML5 encoding. Given a sufficiently rigorous schema for that encoding, it would be trivial to generate TEI from the markup if we wanted it in the future. And I was intrigued with the idea of using ODD for a non-TEI language, something that is rarely done, and that would provide an opportunity for testing the ODD processing toolchain in ways that it is not normally tested.

3. Why Use ODD?

- 8 The W3C provides an excellent validation tool for HTML5 in the form of the [Nu Html Checker](#).⁴ This is the tool we use for final validation of all HTML sites we produce. However, it is a generic tool; it checks conformance against the entire schema (in fact, against any of multiple schemas, depending on the input document type). I wanted to constrain the HTML quite aggressively, provide closed value lists for normally open attributes such as `@class`, define custom attributes (as HTML5 allows) with closed value lists, and incorporate Schematron rules, to ensure that the site style and structure remain consistent throughout the document set. Good praxis also requires documentation of the rules, along with encoding guidelines and examples. ODD is the perfect choice for this (Bauman 2019; Romary and Riondet 2018). ODD files, being TEI files, are also easily processable, a very useful feature whose value will become apparent below.

- 9 In designing the structure of the document collection, the first decision I made was prompted by the problem which had given rise to this work in the first place. Editing a page in the original Dreamweaver setup had involved editing an entire page, including its banner, footer, and so on, and this had given rise to a sort of speciation whereby originally identical blocks of boilerplate code had gradually diverged, resulting in different versions of core site components. To avoid anything like this, it made sense to specify that content documents only have content, which is easy to achieve using the @start attribute on <schemaSpec> (example 1).

Example 1. The <schemaSpec> element, rooting the HTML5 document on <div> instead of <html>.

```
<schemaSpec ident="keats" start="div" ns="http://www.w3.org/1999/xhtml">
  <!-- [...] -->
</schemaSpec>
```

- 10 A content document (which would be edited by the site author) thus consists of a very simple structure as shown in example 2.

Example 2. A content document, rooted on the <div> element.

```
<div>
  <h3>31 October 1795: Keats is Born</h3>
  <div>
    <h4>Swan and Hoop Livery Stables, 24 Moorfields
    Pavement Row, London, where Keats is likely
    born</h4>
    <p>Keats's mother's family run and own the livery
    stable; Keats's father takes over the business
    in 1802/3. Part of the business is known as
    <q>Keat's [or Keates's] Livery Stables.</q>
    They make a very decent living from the
    relatively large business. </p>
  </div>
</div>
```

- 11 Analysis of the site content revealed that its features could actually be encoded using fewer than 20% of the elements available in HTML5, and only around 10% of the attributes (see figure 3).

Figure 3. The site uses a small subset of what HTML5 offers.

- 12 Constraining the encoding choices to this limited subset results in fewer options and less learning for the encoder, and easier styling and processing when building the site. The author (along with a student editor who assisted with the correction of the converted content) was thus able to edit very effectively using the Oxygen interface, and customized element descriptions embedded in the ODD file appeared both in the schema (and thus in the Oxygen interface when editing) and in the HTML documentation generated from the ODD file, which provided additional encoding guidance. To further support easy and efficient authoring and encoding, I created an Oxygen project which provides template files for creating new content documents, Author Mode CSS for rapid proofing of authored content, and a quick-and-dirty build process which creates a complete HTML file from the current content document in the form in which it will eventually appear on the site. This enables encoders to see their work rendered in two different styles and get a clear sense of what it will look like when it is published, without their having to deal with editing complete HTML files.
- 13 HTML5 also allows the use of custom attributes. These are attributes whose names are prefixed with `data-`, and which are ignored by HTML5 validators. I was able to make use of this feature to provide a simple method of encoding a common scenario in the site, where a small graphic

appearing in the text is paired with a larger version of the same graphic (typically not just a higher-resolution graphic but one which actually includes more content) by specifying a custom attribute for the `<img>` element in the ODD file.⁵

Example 3. Defining a custom attribute for a larger variant of an image on the `<img>` element.

```
<elementSpec ident="img" ns="http://www.w3.org/1999/xhtml" module="xhtml5">
  <desc>Image element. May be rendered inline
  or as a block, depending on where it appears
  in the document structure.</desc>
  <!-- [...] -->
  <attList>
    <!-- [...] -->
    <attDef ident="data-lg-version" usage="opt">
      <gloss>path to a larger version of the image to
      show as a popup (usually a relative path)</gloss>
      <datatype>
        <dataRef key="teidata.pointer"/>
      </datatype>
    </attDef>
  </attList>
</elementSpec>
```

14 This allows encoding such as:

Example 4. The custom `@data-lg-version` attribute as it is used in the site encoding.

```
<figure class="rightFloat">
  
  <figcaption>
    <q>Z</q> – John Gibson Lockhart, 1824, by G. S. Newton </figcaption>
</figure>
```

15 The TEI attribute class structure was used to create generic attribute classes such as `att.classable`, to which all HTML elements which need the `@class` attribute belong; then, at the element level, the definition of the `@class` attribute was overridden to constrain it to a fixed value list appropriate for that element:

Example 5. The class attribute, inherited from `att.class`, is overridden for the `<img>` element to allow only two values.

```

<elementSpec ident="img" ns="http://www.w3.org/1999/xhtml" module="xhtml5">
  <desc>Image element. May be rendered inline
  or as a block, depending on where it appears
  in the document structure.</desc>
  <!-- [...]-->
  <attList>
    <!-- [...]-->
    <attDef ident="class" mode="change" usage="opt">
      <valList type="closed" mode="add">
        <valItem ident="normal">
          <gloss>Present the image in normal rectangular format.</gloss>
        </valItem>
        <valItem ident="oval">
          <gloss>Present the image in oval portrait-style format.</gloss>
        </valItem>
      </valList>
    </attDef>
  </attList>
</elementSpec>

```

- 16 This approach provides a considerable advantage over regular HTML5 editing and validation with the W3C tools, because the latter would allow any value in the required form for @class; using ODD allows us to constrain the values at different points in the schema.
- 17 A further advantage of ODD is that it allows us to include Schematron rules. A simple example of how effective this can be is shown by a constraint applied to the HTML5 element. is a generic inline element with no inherent semantic force at all. The only reason for using it is to apply some specific styling to a piece of inline text. Therefore we can formulate the following rule:

Example 6. Defining a Schematron constraint to control how is used.

```

<elementSpec ident="span" ns="http://www.w3.org/1999/xhtml" module="xhtml5">
  <desc>General-purpose phrase-level element.
  Use only if there is no more specific alternative
  for what you want.</desc>
  <!-- [...] -->
  <constraintSpec ident="spansMustHaveStyleOrClass" scheme="schematron">
    <constraint>
      <sch:assert test="@class or @style"> A span element must
      have either a style or a class attribute. </sch:assert>
    </constraint>
  </constraintSpec>

```

```
</constraint>
</constraintSpec>
<!-- [...] -->
</elementSpec>
```

- 18 This prevents the encoder from using `<span>` without a specific reason (`@style` or `@class`).
- 19 Following the model of many TEI projects, we also separated out components resembling authority records such as the list of biographies, as well as index entries and bibliography lists, into separate files, all rooted on the `<div>` element and constrained by the same schema.

4. Building Output from the ODD

- 20 I created a build process, using Ant and Saxon, to turn our ODD file into the various products we need to create from it, as shown in [figure 4](#).

Figure 4. A diagram of the ODD building process.

21 The five phases of this process are:

1. *Refreshing the ODD content.* This is a process we commonly use in our TEI projects,⁶ where specific attributes are intended to be used only as pointers to particular items elsewhere in the project. One example involves the tagging of people in the text. A file `people.xml` contains a list of brief biographies, each of which is a `<li>` element with a unique ID:

```
<li id="keats_t_sen"> [...] </li>
```

When a name is tagged in the text, it needs to be linked to one of these IDs, using the custom `@data-id` attribute. In order to ensure that links are only made to real IDs in `people.xml`, the “Refresh” part of the build process collects all those IDs, then uses them to [re]construct the `<valList>` for the `@data-id` attribute, so that it is impossible to link to a nonexistent ID. The encoder in Oxygen is also helpfully prompted when linking a name:

Figure 5. Linking IDs built into the schema provide help for the encoder in Oxygen.

2. *Building the documentation.* This part of the build process uses the standard TEI Stylesheets process to create HTML documentation from the ODD file (and then follows it up with some post-processing tweaks to make it more reader-friendly). The resulting documentation can be seen [on the project site](#).⁷
3. *Building the schema.* Again, this uses the standard TEI Stylesheet process to create a RELAX NG schema from the ODD file.
4. *Extracting the Schematron.* Schematron rules are then extracted from the RELAX NG schema using XSLT available from the [Schematron project](#).⁸

5. *Compiling the Schematron.* Although Schematron rules inside the RELAX NG file are enforced within the Oxygen editing environment, the automated site-build process needs to do a complete validation of all the files, and this is most effectively done by compiling the extracted Schematron into an XSLT file, again using XSLT from the Schematron project; the site-build process can then do automatic validation of the content documents before building the site.

5. Building the Site

22 Although the focus of this article is on the utility of ODD as a tool for managing encoding projects not based on TEI, it is important to include the final stage of the process, which builds the final HTML website from the fragmentary content documents created by the author and encoder. These are the stages in building the site, in a process which is managed by Ant and based largely on XSLT transformations:

1. Validate the content documents against the RELAX NG schema, using the Jing validator. Stop the build if anything is invalid.
2. Validate the content documents against the Schematron rules as extracted into XSLT, using the Saxon XSLT transformer. Again, stop the build if anything is invalid.
3. Parse the source to list all media files which are *actually used* in the site, and copy them to the output. This avoids cluttering the web server directories with images or other media which are not actually being referenced.
4. Create complete HTML files from the content documents, by integrating them into templates and inserting menus, banners, footers, and other boilerplate components.
5. Build index files for the JavaScript `staticSearch` search engine, which the site uses in preference to a server-based search.⁹ This provides a simple, rapid, and robust method of searching the site, without any dependency on an external service or a database back end.
6. Validate all the HTML and CSS with the VNU validator. Stop the build if anything is invalid.

23 When a new version of the site is published, it is a complete, freshly built version which has passed all the build tests and which we know to be completely valid. In accordance with the principles of our Endings project it is “coherent, consistent and complete.”¹⁰ Anyone working on the project

can run the site-build process and create a new version if they have the credentials to push it to the web server, but the build process cannot be completed unless both the edited content and the generated HTML and CSS are valid.

6. Conclusion

- 24 The use of ODD provides a range of substantial advantages when using a non-TEI XML language, even when other validators exist for that language:
- A substantially reduced schema can be created, excluding elements from the larger schema which are not required for the project.
 - Components such as attributes which are only loosely constrained in the main language can be aggressively constrained through the ODD file.
 - Content documents can be rooted on elements other than the standard root element, so they can be simpler than full documents and contain only what is relevant.
 - Intra-project linking can be constrained by the schema through processing of the site content itself to generate some of the schema specification components.
 - Editors can be supported in their use of an XML editor such as Oxygen through the embedding of information from the `<schemaSpec>` into the output schema, providing popup prompts to help the encoder.
 - Detailed project documentation can be integrated with the schema specification and built into a comprehensive guidelines document.
- 25 These advantages have paid off for the *Mapping Keats's Progress* project. The site has now been live for over a year at the time of writing, and the author is steadily adding to it and refining the existing content. This experience shows that, even where TEI is not used on a project, ODD may still provide a highly effective tool for managing schemas, encoding, and documentation.

APPENDIXES

Appendix 1. ODD File for the *Mapping Keats's Progress* project

`keats.odd` (also available from <https://hcmc.uvic.ca/svn/keats/schema/keats.odd>).

BIBLIOGRAPHY

- Bauman, Syd. 2019. "A TEI Customization for Writing TEI Customizations." *Journal of the Text Encoding Initiative* 12. <https://journals.openedition.org/jtei/2573>; doi:10.4000/jtei.2573.
- Blank, G. Kim. 2018. "Mapping Keats's Progress & the Holy Grail." *Mapping Keats's Progress: A Critical Chronology*, edition 2.8, last modified April 2, 2020. <http://johnkeats.uvic.ca/mkpHolyGrail.html>.
- Burnard, Lou, and Sebastian Rahtz. 2014. "ODD: One Document Does It All." Workshop delivered at TEI 2014 Conference and Members Meeting, Evanston, IL, October 25. <http://www.tei-c.org/Vault/MembersMeetings/2014/workshops/odd-one-document-does-it-all/index.html>.
- Romary, Laurent, and Charles Riondet. 2018. "EAD-ODD: A Solution for Project-Specific EAD Schemes." *Archival Science* 18 (2): 165–84. doi:10.1007/s10502-018-9290-y.

NOTES

- 1 ODD has been used to model non-TEI schemas in other projects. Bauman (2019) mentions the Music Encoding Initiative and the W3C Internationalization Tag Set among others, and Romary and Riondet (2018) describe the use of ODD to generate schemas and documentation for EAD projects.
- 2 The site is at <https://johnkeats.uvic.ca/>, and the SVN repository for the project is at <https://hcmc.uvic.ca/svn/keats/>.
- 3 Accessed April 17, 2020, <http://www.html-tidy.org>.
- 4 Accessed April 17, 2020, <https://validator.github.io/validator/>.

5 HTML5 provides more sophisticated methods of encoding responsive images, but given the simple needs of this project, a single custom attribute was chosen as the encoding method. At build time this is transformed into a simple linking structure which pops up the larger version when the user clicks on the smaller one.

6 An example would be the *Digital Victorian Periodical Poetry* project (version 0.9beta, updated March 3, 2020, <https://dvpp.uvic.ca/>), in which taxonomies of, for example, rhyme types or sonic devices used in the poetry are encoded using TEI <taxonomy> and <category> structures, and then these definitions are harvested into the ODD file to provide <valList>s for @ana attributes.

7 Martin Holmes, “Schema and Documentation for *Mapping Keats’s Progress* project,” last updated March 20, 2020, <http://johnkeats.uvic.ca/documentation/keats.html>.

8 “The Schematron ‘Skeleton’ Implementation,” accessed April 18, 2020, <http://schematron.com/front-page/the-schematron-skeleton-implementation/>.

9 Accessed April 17, 2020, <https://github.com/projectEndings/staticSearch>.

10 Release Management principle 5.2, “Principles,” The Endings Project: Building Sustainable Digital Humanities Projects, accessed April 18, 2020, <https://projectendings.github.io/principles/>.

AUTHOR

MARTIN HOLMES

Martin Holmes is a programmer in the University of Victoria Humanities Computing and Media Centre. He served on the TEI Technical Council from 2010 to 2015 and was managing editor of the *Journal of the Text Encoding Initiative* from 2013 to 2015.