Journal of the Text Encoding Initiative

Issue 12 | July 2019 -
Selected Papers from the 2017 TEI Conference

A TEI customization for writing TEI customizations

Syd Bauman

Electronic version
URL: http://journals.openedition.org/jtei/2573
DOI: 10.4000/jtei.2573
ISSN: 2162-5603

Publisher
TEI Consortium

Electronic reference
Syd Bauman, « A TEI customization for writing TEI customizations », *Journal of the Text Encoding Initiative* [Online], Issue 12 | July 2019 -, Online since 15 November 2019, connection on 03 March 2020. URL : http://journals.openedition.org/jtei/2573 ; DOI : 10.4000/jtei.2573

A TEI customization for writing TEI customizations

Syd Bauman

ABSTRACT

A schema, in general, is intended to be used to check a document for errors before those errors cause problems in processing. However, schemas can also help us write our documents. The TEI ODD language (and the more modern version thereof, Pure ODD), in particular, 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. There are several examples of (1), including the TEI Guidelines, the Music Encoding Initiative, the ISO Feature Structure encoding system, and the W3C Internationalization Tag Set. And there are several well known examples of (2), including TEI Lite, TEI Tite, TEI Simple Print, Comic Book Markup Language, Digital Humanities Quarterly, TEI-in-Libraries, and the markup language for this journal.[1]

Of all these various uses of the TEI ODD language, the most common (by far) is to create a customized TEI for use in a particular project. This is because the TEI Guidelines are not meant to be used out of the box—every TEI project is expected to customize the TEI. For example, in raw

(i.e., uncustomized) TEI, the `@type` attribute of `<stage>` has a robust set of nine suggested values: `"business"`, `"delivery"`, `"entrance"`, `"exit"`, `"location"`, `"mixed"`, `"mixed"`, `"novelistic"`, and `"setting"`. But a project may very well wish to expand this list (e.g., by adding `"onStage"`, `"prop"`, `"remains"`) and require that encoders use a value from this expanded list. This sort of molding of the TEI to local purposes is done by creating a TEI customization using the TEI ODD language.

In this paper I will present a TEI ODD customization of the TEI language that is intended to help a user *write* a TEI ODD customization of the TEI language. It is not intended to check a TEI ODD customization document for errors, and in fact will likely flag things as "errors" that an ODD processor would find perfectly acceptable. But it does allow a user to more quickly, easily, and accurately write a TEI customization ODD using an XML editor.

INDEX

Keywords: XML, schema languages, ODD, customization

ACKNOWLEDGEMENTS

The author would like to thank the Women Writers Project (now part of the Digital Scholarship Group), which supported most of the development of `tei_customization`; the students in the WWP customization and data modeling workshops that helped refine it; the TEI Technical Council for their support; Martin Holmes for his assistance in getting it into the TEI oXygen framework; and George Bina (of SyncRO Soft) for his suggestions on handling the infamous "Conflicting ID-types" problem.

1. Schemas

1 A *schema* is a formal declaration of various combinations of XML constructs that are required, permitted, or disallowed in a document. A schema might express, e.g., "an `<article>` *may* have any number of `<citation>`s, but each `<citation>` *must* have exactly 1 `<quotation>` followed by exactly 1 `<attribution>`."

1.1 Schema Languages

2 Schemas are expressed using a *schema language*. Many schema languages have been developed over the years. However, only a few remain in common use, and even fewer are pertinent for the validation of TEI documents, and are thus important here.

3 Table 1 lists some of the schema languages for XML documents, arranged roughly by family of language. Those that are pertinent and thus discussed herein are *emphasized*.

Table 1. Some of the schema languages for XML documents, arranged roughly by family of language.

SGML Document Type Declaration Family	W3C Schema Language Family	Regular Expression Family	Others
DTD	XML-Data	RELAX	DSD
XDTD	XDR	XDUCE	*Schematron*
DTD++	DCD	TREX	Examplatron
DTD++ 2.0	SOX	*RELAX NG*	X-definition
	DDML		*TEI ODD*
	XSD		

(See, e.g., van der Vlist 2002.)

4 The schema languages in common use are the three main *closed* or *grammar-based* languages: DTD, RELAX NG, and the W3C Schema Language; and the *open* or *rule-based* language: Schematron.

5 The DTD language is historically extremely important, as it is really the first significant attempt at application-independent document constraint, and was built into both SGML of 1986, and its descendant XML a dozen years later (Bray, Paoli and Sperberg-McQueen 1998). Its creation was a leap forward in text processing. However, it has been completely eclipsed by more modern languages that are either easier to use, more expressive, or more flexible, and thus is no longer important in TEI document preparation.

6 The W3C Schema Language (W3C 2004) is an extremely powerful schema language that is complex and hard to learn. In addition to expressing a grammar against which an XML instance document (or a portion thereof) may be tested for validity, it also supports augmentation of information contained in the XML instance being validated with information that is in the schema (e.g., default values for attributes that do not appear in the instance, and datatype information).[2] It has the advantage of being a recommendation of the World Wide Web Consortium.

7 RELAX NG is a powerful, clean, and simple schema language that is relatively easy to learn. It is basically a regular expression language. The "normal" regular expression languages most of us are familiar with operate over the set of strings. RELAX NG operates instead over the set of XML trees. That is, we are used to thinking of characters as the primitives used in our regular expressions, and of our regular expressions as matching zero or more substrings of a document. In RELAX NG the primitives are constructs like elements and attributes, and our regular expression (schema) either matches the entire document instance (it is valid) or not (it is invalid).

8 RELAX NG has additional advantages. It was the native schema language underlying TEI P5 up until the introduction of Pure ODD (see Burnard 2013) with version 3.0.0 in January 2016. Furthermore, it is an ISO standard, grouped with several other XML technologies (including Schematron).

9 The Schematron language is also clean, simple, and relatively easy to learn; furthermore it is arguably more powerful than any of the others. However, while it is very easy to use Schematron for what it was designed to do—namely, to catch specific problems (like "the `@ref` on this `<persName>` points to a `<biblStruct>`, not a `<person>`"), it is somewhat more difficult to use Schematron to govern the structure of an entire document (like "the outermost element is `<TEI>`, which must have two children `<teiHeader>` and `<text>`; `<teiHeader>`, in turn, consists of a `<fileDesc>` followed by one or more of …"). Furthermore, it is much slower, and (at least for now) editing tools will only tell you that a document is invalid against a Schematron rule after the invalidity has been introduced; they would not, for example, give you a pop-up box of only the allowed values, so that you can avoid creating the invalidity in the first place.

10 However, if the Schematron schema contains a "Quick Fix" for the error, software can read it and may be able to perform an automated correction or prompt the user to perform a correction.[3] For example, the rule "every `<lg>` that contains 5 or more `<l>`s must have an `@xml:id`" could be expressed as shown in `SQF_lg`. In this case, the oXygen editor will prompt the user to automatically insert an `@xml:id` to fix this error by selecting a generated contextual menu item.

Example 1. Schematron rule that includes a method for correction of the error.

```
  <sch:rule context="tei:lg[ count( descendant::tei:l ) gt 4 ]">
   <sch:assert test="@xml:id" sqf:fix="addID"> Sizable groups of metrical
   lines should be identified. </sch:assert>
   <sqf:fix id="addID">
    <sqf:description>
     <sqf:title>Insert an @xml:id attribute</sqf:title>
    </sqf:description>
    <sqf:add target="xml:id" node-type="attribute" match="." select="generate-
id(.)"/>
   </sqf:fix>
  </sch:rule>
```

1.2 The Purpose of a Schema

11 In XML document production, and particularly in digital humanities XML document production, we use schemas for various purposes.

1.2.1 Constraint

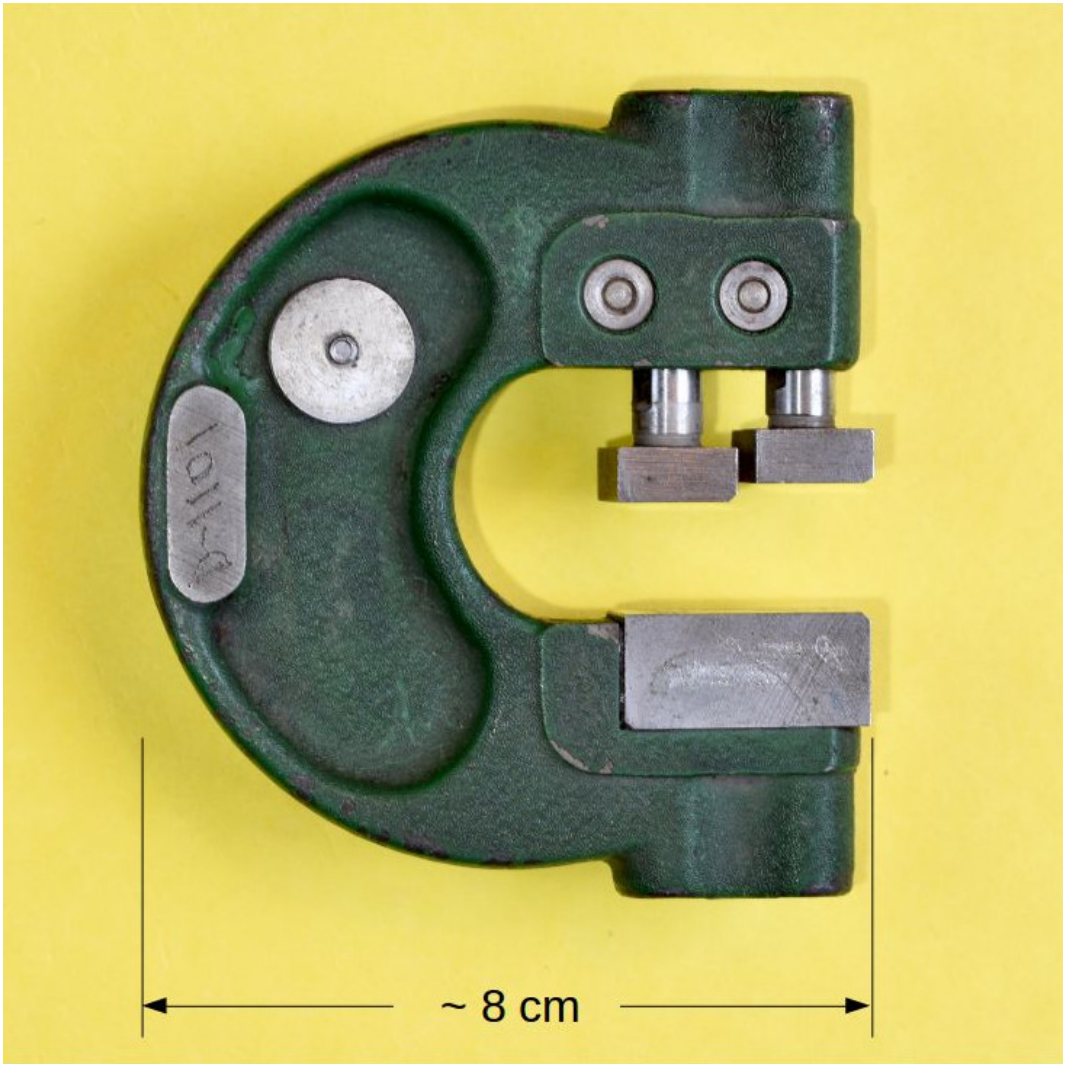
12 Strictly speaking, the primary purpose of a schema is to constrain our data. A schema serves as a gate-keeper; it divides the world of XML documents into two piles: those that are valid against it, and those that are not. By rejecting those files in the "not valid" file pile, we can spare the rest of our processing system from needing to deal with errors that can be quickly and easily detected by an automated process, namely validation.

13 For example, let's say we are publishing a large collection of letters from prisoners, each of which is stamped (by the prison guards) with the day it was sent. If our schema *requires* that there be a date in a particular format in a particular place (say, in W3C format on the `@when` of `<correspAction type="sent">`), then our publication software does not have to worry about what to do if it finds a

letter without a date when it tries to sort them. The publication software (or, to be more precise and stop anthropomorphizing code, the computer programmer who writes the publication software) can depend on the fact that every letter will have a date in exactly the right spot.

14 As Wendell Piez points out (Piez 2001), this is exactly analogous to the use of "go/no-go" gauge.

Figure 1. snap go no-go gauge.



The snap go/no-go gauge shown has two (adjustable) apertures. The first (shown on the right) is slightly wider than the second. With this gauge in one hand, a worker on an assembly line can quickly pick up a cylinder (say, a gun barrel or a bicycle frame tube) in the other and test to see if

it has the correct outer diameter. If the cylinder slips between both the first and second apertures, it is too small. If it cannot fit past even the first gap, it is too large. If it passes the first but not the second, it is the right size. A direct, physical application of the Goldilocks principle, as it were. In document production, our documents are the cylinders, and the schema is the gauge. If a document passes through both apertures it is invalid because it is missing a required construct (say, a TEI document without a `<fileDesc>`). If a document cannot pass through the first aperture, it is invalid because it has an extraneous construct (say, a TEI document in which an `<lb>` element has text content). Only a document that passes the first gap but not the second may be considered valid.[4]

15 This strict use of schemas is helpful to us in digital humanities because it allows us to catch errors in our documents very early in the document creation process. It is theorized (and there is anecdotal evidence to suggest) that it is much easier (read "cheaper") to catch errors earlier in the document production assembly line rather than later.[5] For example, in example 2 we have an erroneous TEI encoding of a passage from *A summary history of New-England* by Hannah Adams.

Example 2. Invalid TEI encoding of a short passage from *A summary history of New-England* by Hannah Adams.

```
<lb/>this propoſal to the
  general court. After ſome <lb/>debate, their plan was accepted, and the
  company <lb/>proceeded to a new election of officers, who were <lb/>to
  repair to and ſettle in <placeName>New- England</placeName>.
  <persName>John <lb>Winthrop</lb></persName>, Eſq. Of
  <placeName>Groton</placeName>, in <placeName>Suffolk</placeName>, a
  gentle¬ <lb break="no"/>man of diſtinguiſhed piety and ability, was
  choſen <lb/>governor.
```

The error, while perhaps obvious to seasoned TEI encoders, is easy to miss. However, it is common for software projects to format their TEI documents for the web to include a rule (like that in Example 3) to simply replace the `<lb>` element with a space, in order to ignore original lineation and re-flow the document content. Such rules often summarily ignore the "contents" of the `<lb>`, because in TEI `<lb>` is not allowed to have any content. Thus in many cases the resulting output will be

```
… After ſome debate, their plan was accepted, and the company proceeded to a new
 election of officers, who were to repair to and ſettle in New-England. John, Eſq.
of Groton, in Suffolk, a gentle¬man of diſtinguiſhed piety and ability, was choſen
 governor.
```

The error, of course, is that the word "Winthrop" has been dropped. Since the passage reads reasonably well without Governor Winthrop's surname, it could be very hard to catch this error after the transformation. But validation against a TEI schema would catch it instantly.

16 Besides catching individual errors earlier, schemas provide a mechanism to ensure that members of a set of documents have sufficient similarity to one another for interchange or even interoperability, [6] and "… it is standards-based interchangeability, when applied to information objects, that provides us with the coveted advantages for our data of vendor- and application-independence, of modular architectures and layered systems, commodity tool markets, and long-term data stability. (Not that any of these things become easy to achieve even on a standards basis: but at least with standard ways of judging correctness, there is some hope for them.)" (Piez, 2001). The pithy summary for the logic here has been provided by Michael Sperberg-McQueen: "to paraphrase an old Chicago election adage, constrain your data early and often" (Sperberg-McQueen, 1992).

Example 3. XSLT template that converts an `<lb>` into a space.

```
<xsl:template match="tei:lb">
   <xsl:if test="not( @break eq 'no' )">
    <xsl:text> </xsl:text>
   </xsl:if>
  </xsl:template>
```

1.2.2 Uses other than constraint

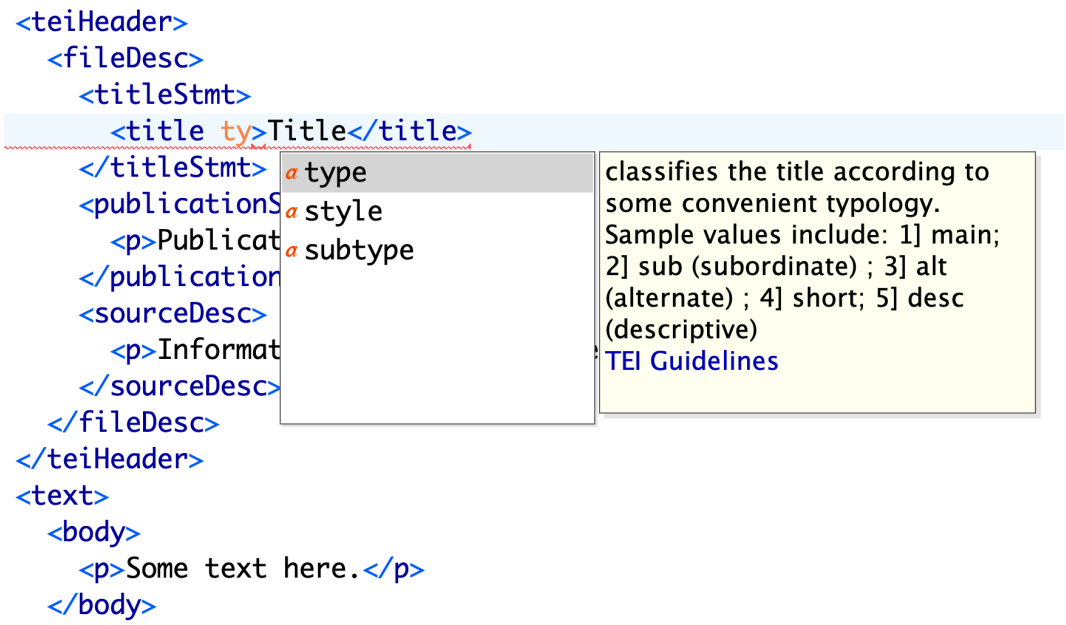
17 Although schemas are intended for document constraint, there are several "off-label" uses for them as well.[7] Wendell Piez points out (Piez, 2001) that "Sometimes a schema might be more than a 'go/no-go gauge,' becoming a diagnostic and investigatory instrument." That is, we can use a schema and validation technology to discover things about our data, rather than merely to divide it into two piles. David Birnbaum presented an early use-case for what Piez (also in Piez, 2001) later called "mimetic" and "exploratory" markup[8] at the ACH/ALLC '97 conference in Queens, Ontario; that paper was later published (Birnbaum, 1999).

18 Indeed, the TEI Guidelines have recognized this fact since P1 in 1990: "it may be beneficial to define a restrictive set of rules relating to one particular view or hypothesis about the text—if only as a means of testing the usefulness of that view or hypothesis" (TEI P1–P5)[9] Furthermore, both Liam Quin (1996) and Birnbaum (1999) have suggested methods of using schemas (DTDs in particular) as exploratory or suggestive tools.

1.2.2.1 Help—learning about the schema

19 In addition to the well established tradition of using schemas loosely as a tool for investigating our data, we often use schemas as a research tool to investigate our own markup languages. This may sound silly, since the schema (and its documentation) *define* the markup language. But we are all immediately familiar with a common demonstration of such an investigation. Often, a digital humanist faced with an encoding problem will hypothesize a solution, and then ask the question "does my encoding language support this solution already?" For example, during document analysis of *Flatland* by A. Square (i.e., Edwin Abbott Abbott), an encoder wonders how to capture the subtitle (*A Romance of Many Dimensions*) separately and to differentiate it from the main title. She plans to use TEI P5, so her first thought is "Does the TEI `<title>` element have a `@type` attribute?" So she switches into oXygen, creates a new `tei_all` document, puts her cursor immediately before the closing angle bracket of the `<title>` start-tag[10] and types a space. The result (figure 2) immediately tells her that yes, the TEI `<title>` element has a `@type` attribute; furthermore, it tells her that `"sub"` for "subordinate" is one of the TEI suggested values.

Figure 2. Choosing the `@type` attribute from a drop-down in oXygen.

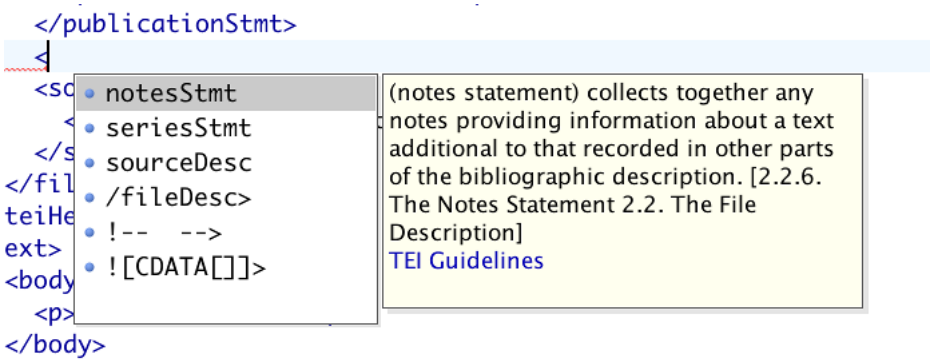


In this fictional—but completely believable—example the encoder has used the schema (`tei_all`) through a tool (oXygen) as a way of finding out about the markup language (TEI). Yes, she could just as well have read the documentation and obtained the same information, but she didn't have to; she used the schema as an investigative tool to help her learn about the markup language.

1.2.2.2 Help—writing documents

20 In addition to helping us learn about our markup languages, a schema can help us write our documents. Editing software can read the schema and restrict the markup or even text that can be entered based on the rules in the schema and the position of the cursor. Figure 3 demonstrates oXygen helping an encoder. Here oXygen is not just answering a common question: is the TEI element for a notes statement `<notesStmt>` or `<noteStmt>`? It is also requiring that the user enter only one of three elements allowed by the schema (or a comment, etc.).

Figure 3. Inserting an element in oXygen.



By using an editor that understands the schema, an encoder can avoid making common mistakes (like misspelling an element name) at the time of the original encoding. This is extremely helpful in constraining one's data early.

2. The Purpose of an ODD

21 As explained in ED W 29,[11] the ODD system[12] was explicitly created to bring the principles of literate programming to the creation of SGML (now XML) markup languages. The point is "that code and documentation [should] be written and maintained as a single integrated resource, from which both working [schemas] and readable documentation can be generated" (Flanders and Bauman, 2010). That is, the main stated purpose of an ODD is to create a well-documented markup language. However, there is more than one way an ODD can be used in this pursuit.

2.1 Create a complete markup language

22 An XML markup language has both a syntactic aspect (the vocabulary of elements and attributes, and the grammar of how they fit together) and a semantic aspect (what those elements and attributes mean). The ODD system permits, and outright encourages, the simultaneous specification of both of these aspects:

1. an XML schema using special-purpose formal elements; and
2. prose definition of the constructs specified in that schema using "normal" TEI markup.

Many think of the ODD system as the language used to define the TEI markup language. Indeed, at its outset, the "One document does it all" system was developed in TEI P2 explicitly for the creation of TEI P2 itself (ED W 29), and further iterations of it including the major re-design in XML for P5 (at one point referred to as "Son of ODD" (Burnard and Rahtz 2004)) and the recent move to Pure ODD (Burnard, 2013) are used to define the TEI markup language. However, ODD was never intended to be only for defining TEI. It was and is intended to be a general-purpose language for defining markup languages.

23 In fact, several markup languages other than TEI have been defined using ODD. These include the Music Encoding Initiative,[13] the ISO Feature Structure representation system,[14] and the W3C Internationalization Tag Set.[15] Furthermore a version of the Itsy-bitsy teeny-weeny simple hypertext DTD[16] has been expressed in ODD as an early proof-of-concept, and a highly-constrained subset of HTML itself has been re-written in ODD.[17]

2.2 Customize a language

24 The TEI ODD language is also used to *customize* an existing ODD-defined markup language. Take as an example the fictional language "RHPSML." When creating this system for encoding, the RHPS project manager decides to use TEI, but then says to herself "I need the `drama` module; I don't need all this stuff about dictionaries, linguistics, or manuscripts; I (of course) need to constrain all those `@type` attributes; but I also need a special new element for audience participation." She can create such a customized TEI markup language using the TEI ODD system. To do so she writes a "customization ODD" that expresses only the *differences* between RHPSML[18] and uncustomized ("vanilla") TEI P5.

25 Customizations, expressed via customization ODDs, can serve a variety of audiences ranging from the lone scholar working on the encoding of a single document to field-wide customizations used by dozens or even hundreds of projects; see for instance EpiDoc,[19] TEI Tite, Comic Book Markup Language,[20] and the TEI-in-Libraries Best Practice Guidelines.[21] Quite commonly, customizations are used by a single project for a corpus of related documents. Examples include the *Women Writers Project*,[22] the *Digital Archive of Letters* in Flanders,[23] *Digital Humanities Quarterly*,[24] *Deutsches Textarchiv*,[25] and the imaginary RHPSP.

26 The TEI Consortium publishes a large set of customizations. These can be loosely divided into two groups: “sample” customizations, which are intended to be actually used by projects or at least provided as useful examples of complete customizations; and “template” customizations, which are provided as little more than shells, which may be used as starting points for creating complete customizations.

27 sample customizations:

- TEI Lite
- TEI SimplePrint
- jTEI — the markup languge used for this journal
- ENRICH

28 templates customizations:

- tei_corpus
- tei_dictionaries
- tei_docs
- tei_drama
- tei_math
- tei_minimal
- tei_ms
- *tei_odds* — which may be used to validate ODDs
- tei_speech
- tei_svg

2.3 Use matrix

29 As we have just seen, there are two possible purposes for ODD: to create a markup language, and to customize a language so created. Furthermore we might divide the world of markup languages that are created and customized using ODD into two broad categories: TEI and non-TEI. Those two binary axes give us four uses to which ODD might be put. The language defined by TEI ODD (and minimally represented by the `tei_odds` schema) needs to be able to handle all four of these use cases.

Table 2. A use matrix for ODD.

language→ purpose↓	non-TEI	TEI
create	`tei_odds`	`tei_odds`
customize	`tei_odds`	`tei_odds` and `tei_customization`

However, everyone using TEI is expected to use a customized version of the complete TEI markup language. For most projects, this means writing a local customization. Thus I suspect that the lower right corner of that table—customizing the TEI markup language—is *by far* the most common use of ODD.

3. `tei_customization`

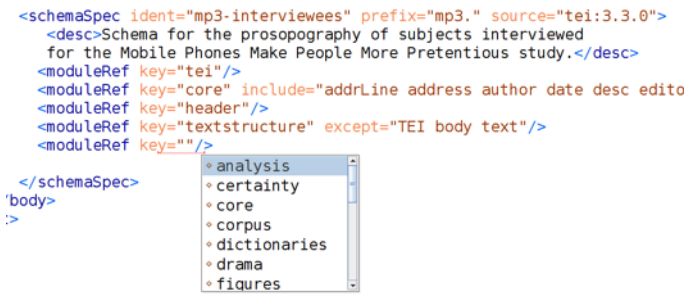
3.1 Use case

30 I want help (both for myself and those in my workshops) *writing* TEI customizations. For example, I don't want to have to remember if the module that includes the elements for a personography is `<namesdates>`, `<namesDates>`, or `<namesanddates>`. I'd like to be able to just pick the module name (or whatever) from a list. `tei_odds`, the generic "template" schema provided by the TEI, necessarily has to be able to validate documents in all four categories, as shown in table 2, so it is not very helpful in this regard. This is because an ODD in either of the non-TEI categories is likely to have module names that are not from the TEI Guidelines, so that schema cannot restrict the values of the `@key` of `<moduleRef>`. But because it is only addressing (a subset of) those ODDs intended to customize the TEI language, `tei_customization` can, and does, use a controlled vocabulary for that attribute and others like it.

3.2 What it does

31 Like any good schema, the `tei_customization` customization helps you by limiting the choices available at any point as far as reasonable. XML-aware tools, in particular XML editors[26] can take advantage of the limited choices available in the schema by presenting them to the user and objecting if she enters something other than one of the choices. For example, figure 4 shows the oXygen XML editor presenting the list of limited possibilities allowed as the value of the `@key` attribute of `<moduleRef>` (that is, the list of TEI module names).

Figure 4. oXygen displaying the list of TEI module names as the controlled vocabulary for `@key` attribute of `<moduleRef>`.



A user need only select the desired value from the list. Both oXygen and Emacs/nxml permit the user to enter a value other than one in the list, but then immediately underline it in red.

32 The constraints `tei_customization` imposes on a customization are those that its primary author —and to a lesser extent the TEI Technical Council and TEI community at large—think are probably helpful for *writing* a customization. The important point here is that `tei_customization` is *not* intended as an arbiter of what is a valid TEI ODD or not. That is, unlike most schemas, `tei_customization` is not interested in the division of the world of XML documents into two sets of documents (valid and invalid), because a file invalid against this schema may be a perfectly reasonable ODD, even a reasonable TEI customization ODD.

33 For example, `tei_customization` does not include the TEI `<gap>` element. Any use of `<gap>` will be considered invalid by `tei_customization.rng`. But TEI does allow the `<gap>` element in an ODD. And it is possible to come up with a plausible, if not likely, scenario in which use of `<gap>` might be

useful. But `tei_customization` takes the view that it is so unlikely that `<gap>` would be used in a customization ODD that its occurrence would be a mistake, or at least that the cost of having one more element name in the pop-up list of possible elements outweighs the benefit of its being valid.

3.3 How it does it

34 Like any TEI customization (including `tei_all`), the `tei_customization` customization has both a closed or grammar-based component expressed in RELAX NG[27] and an open or rule-based component expressed in ISO Schematron.

35 The grammar-based component is used whenever the constraint can be expressed directly in ODD. That is, when the constraint depends on nothing but the element type, or the attribute name and element type.[28] For example, the constraint that the `@key` attribute of `<moduleRef>` element should be one of the 21 TEI module identifiers depends on nothing other than the element type (`<moduleRef>`) and attribute name (`@key`). Other constraints, those that depend on more conditions than just the element type or attribute name and element type, are expressed in ISO Schematron. For example, the constraint that there should only be one `<moduleRef>` with any particular one of those 21 possible values depends on more than the element type and the attribute name, it depends on the values of the `@key` attributes of other occurrences of the element type (`<moduleRef>`).

36 It is worth noting that the expression of these constraints can be quite cumbersome. The worst offender is the definition for the `<ident>` attribute of `<elementSpec>`, which comprises over 3800 elements and attributes. See example 4.

Example 4. Excerpt of the definition of `@ident` of `<elementSpec>` from `tei_customization`.

```
<attDef ident="ident" mode="replace" usage="req">
   <datatype minOccurs="1" maxOccurs="1">
    <dataRef key="teidata.enumerated"/>
   </datatype>
   <valList type="semi">
    <valItem ident="TEI">
     <gloss versionDate="2005-12-24" xml:lang="en">TEI document</gloss>
     <desc xml:lang="en" versionDate="2018-02-16">module:
     textstructure</desc>
    </valItem>
    <valItem ident="ab">
```

```
      <gloss versionDate="2005-01-14" xml:lang="en">anonymous
      block</gloss>
      <desc xml:lang="en" versionDate="2018-02-16">module: linking</desc>
     </valItem>
     <valItem ident="abbr">
      <gloss versionDate="2005-01-14" xml:lang="en">abbreviation</gloss>
      <desc xml:lang="en" versionDate="2018-02-16">module: core</desc>
     </valItem>
     <!-- … "abstract" through "writing" (567 <valItem>s) removed for journal
article … -->
     <valItem ident="xenoData">
      <gloss versionDate="2015-05-30" xml:lang="en">non-TEI
      metadata</gloss>
      <desc xml:lang="en" versionDate="2018-02-16">module: header</desc>
     </valItem>
     <valItem ident="xr">
      <gloss versionDate="2005-01-14" xml:lang="en">cross-reference
      phrase</gloss>
      <desc xml:lang="en" versionDate="2018-02-16">module:
      dictionaries</desc>
     </valItem>
     <valItem ident="zone">
      <desc xml:lang="en" versionDate="2018-02-16">module: transcr</desc>
     </valItem>
    </valList>
   </attDef>
```

3.4 How it is created

37 There are two obvious major problems for maintaining an ODD like `tei_customization.odd`. First, the aforementioned size of the lists involved would make hand-editing annoying, if not outright error-prone. But more importantly, TEI P5 is (deliberately) a moving target. The TEI Consortium issues updates roughly twice a year. New elements, attributes, and classes may be added, lists of possible attribute values change, occasionally an element or attribute is even removed, etc. Every time one of these changes is made, several places in `tei_customization` would need to be changed. For example, if the `<teiCorpus>` element were removed from TEI P5, seven different places in `tei_customization.odd` would need to be updated: it would have to be removed from

1. he list of possible values for `@start` of `<schemaSpec>`,
2. the list of possible values for `@include` of `<moduleRef>` (grammar-based, lists all possible values),
3. the list of possible values for `@include` of `<moduleRef key="core">` (rule-based, lists only values from the `"core"` module),
4. the list of possible values for `@except` of `<moduleRef>` (grammar-based, lists all possible values),
5. the list of possible values for `@except` of `<moduleRef key="core">` (rule-based, lists only values from the `"core"` module),
6. the list of possible values for `@key` of `<elementRef>`, and
7. the list of possible values for `@ident` of `<elementSpec>`.

Furthermore, the value `"teiCorpus"` appears in the value of the `@except` attribute of the `<moduleRef key="core">` in the `tei_customization.odd` (as opposed to the `<moduleRef>` it is defining) — this one should be removed, too, but since current ODD processors would ignore it, would not cause a problem if it were left.

38 Either way, in order to avoid this potential maintenance nightmare, `tei_customization.odd` is not a static file, but rather is generated by running an XSLT program that reads as its input the source to TEI P5[29] and writes `tei_customization.odd` as its output. Running this program is one of the steps in the "build" process for creating a new set of TEI P5 generated outputs.[30]

3.5 What it does not (yet) do

3.5.1 Processing Model

39 An ODD file may describe not only the syntax and semantics of conforming document instances, but also an intended processing of them. This intended processing is described in mildly abstract terms by a small set of new ODD elements introduced in TEI P5 release 3.0.0. This capability, known as the "Processing Model," is not yet covered by `tei_customization`, and therefore is summarily

ignored in the rest of this paper. This does not mean the TEI Council considers this an appropriate state of affairs—we can expect a future release of `tei_customization` to include processing model elements.

3.5.2 Deprecation

40 At times the TEI Council decides to change the recommendation for an encoding practice. This typically consists of two steps: removing support for the practice as it stands, and adding support for the new method of recommended encoding. Rather than enacting both changes to the schema at once, practical support for the new method will be added to the schema first, and then several releases later support for the old practice will be removed. This gives users who wish to use the most recent version of the TEI Guidelines a period of time, typically two years, to switch from the old practice to the new.

41 During the period between addition of the new practice and outright removal of the old practice, the old practice is *deprecated*.[31] It is not flagged as an invalid practice, but it will be flagged in the TEI technical documentation as being phased out, and its use will generate a warning message, typically advising the user that it has been superseded by the new practice.

42 This deprecation can take place in one of two ways: automatically or manually. Automatic deprecation can be used when the practice being deprecated is directly represented in the TEI with an element that is a member of `@att.deprecated`. If this is the case, automatic deprecation can be accomplished simply by placing a `@validUntil` attribute on the element that defines the construct to be deprecated. The value of this attribute is a date which represents the earliest date on which the feature will be removed. ODD processing software, including the routines that build the TEI Guidelines from its source, will flag as deprecated any constructs with a `@validUntil` that is in the future, and actually ignore any that have a `@validUntil` that is in the past.

43 Manual deprecation is used when the practice being removed is not so simple as to be represented in the TEI source by a single element that is a member of `@att.deprecated`, e.g., changing the content model of an element, rather than removing it entirely. In these cases a hand written constraint is added to the TEI to give users a warning that the old practice will be removed in the future.

44 Currently the `TEI-to-tei_customization.xslt` program does nothing special with deprecated constructs, whether automatically or manually deprecated. That is, it ignores `@validUntil` attributes, thus including names of deprecated constructs in the list of attribute values that populate the pop-up lists when using `tei_customization` schema to write an ODD.

45 However, it would be a reasonable enhancement for `TEI-to-tei_customization.xslt` to handle automatically deprecated constructs more intelligently. The TEI Technical Council has not yet decided, however, what that handling would be. It would probably be reasonable for `TEI-to-tei_customization.xslt` to either

- ignore constructs whose `@validUntil` is in the past;
- ignore constructs that have a `@validUntil` whether or not it is the past;
- ignore constructs whose `@validUntil` is in the past, and, where possible, add a warning to the description of values representing constructs whose `@validUntil` is in the future; or
- stop processing and raise an error when there is a `@validUntil` that is in the past.

Handling manually deprecated constructs is another matter. To do so `TEI-to-tei_customization.xslt` would need to be able to parse, understand, and act on the Schematron code in one or more `<constraint>` elements. Thus it is probably prohibitively difficult, and there are no plans to do so at this time.

3.6 How to get it and use it

46 The XSLT program used to generate `tei_customization.odd` can be found in the TEI GitHub repository. It is currently called `TEI-to-tei_customization.xslt`. The generated `tei_customization` ODD file and the schemas generated from it can be found in each release of the TEI from 3.3.0 on.[32]

47 Furthermore, the current version of `tei_customization` is available from within oXygen as part of the TEI oXygen framework. However, the RELAX NG schema (tei_customization.rng or tei_customization.rnc) has the behavior discussed in Appendix 1. While this is not a bug or broken in any way, it is likely to be confusing and problematic for most users of oXygen. The TEI_Council is interested in finding a way around this difficulty.

4. Refining purpose

48 A customization can readily be categorized into one of four groups: restriction, isomorphism, extension, or transmogrification. (For an explanation of these terms, further details, and diagrams see Bauman and Flanders, 2004.) A customization may also produce a completely disjoint markup language, but such a case is more of a replacement than a customization, and thus will not be considered here[33] Furthermore isomorphisms (because although they do not change the structure of conforming documents, they do change the names of XML constructs) and transmogrifications (because they include extensions) are, for our purposes, extensions.

49 Thus, for our purposes, a customization can fall into one of two major categories:

- a strict subsetting customization, which defines a language for which every document instance that is valid against the customized schema is also valid against `tei_all`; or
- something else, which we will call an extension.

Using this somewhat simplified view, the TEI ODD language can be used for three purposes: to *create* a markup language, to subset a markup language so created, or to extend a markup language so created.

50 Remembering the broad division of markup languages into TEI and non-TEI, the use matrix described above (section 1.2.2) becomes a 2 by 3 (instead of 2 by 2) matrix. In table 3 the content of each of the 6 cells expresses whether or not it is useful to use `tei_customization` for that purpose and category of markup language.

Table 3. A use matrix for `tei_customization`.

language→ purpose↓	non-TEI	TEI
create	no	no: 7000+ errors[34]
extend	no	maybe: errors on names of newly created constructs
subset	no	*yes!*

Although it is not strictly true, it may be useful to think of `tei_customization` as intended for strict subset customizations. This is because definitionally use of any non-TEI names in, say, the `@ident` of `<schemaSpec>` will cause a validation error against it, even though such a use is perfectly valid against `tei_odds`, and is a very appropriate and reasonable thing to do in a TEI customization ODD. Remember, `tei_customization` does not differentiate proper ODD customizations from improper ones; `tei_customization` makes it easier to *write* a TEI customization ODD.

APPENDIXES

Appendix 1. Technical Problem: conflicting ID-types

There are several features of XML that DTDs support but RELAX NG does not. This lack of support is deliberate: the designers of RELAX NG felt that a clean separation between validation and other functions is important. One feature that RELAX NG does not natively support is ID/IDREF checking. The ID/IDREF mechanism allows an encoder to point from one XML element (`<A>`) to another in the same document (`<B>`) by putting the same value on an attribute declared as type IDREF (or as one of the space-separated tokens in the value of an attribute declared as type IDREFS) on `<A>` as an attribute declared as type ID on `<B>`. TEI P5 does not use this feature at all; there are no attributes in P5 declared as type IDREF or IDREFS.[35] To support this feature a validator has to place various constraints on the values of those attributes declared as ID or IDREF (or IDREFS):

1. The value of each ID attribute is an NCName.
2. The value of each ID must be unique (on any ID attribute in the given document).
3. The value of each IDREF attribute (or each space-separated token of each IDREFS attribute) must match the value of an ID attribute somewhere in the document.

The philosophy RELAX NG follows is that constraints #2 and #3 are outside its scope. One short explanation for why, above and beyond the desire for a clean separation of validation proper from other functionality, is that it is impossible, or at least very difficult, to support ID/IDREF without requiring an unambiguous relationship between each instance element or attribute and its declaration in the schema. This unambiguous relationship is required in DTDs, but may not be available when validating against a RELAX NG schema. That is, a RELAX NG

schema may have multiple patterns that match `<B>`, and it is not always possible to tell which pattern matched a particular `<B>` in the document instance. For more information see the DTD Compatibility specification (Clark and Murata, 2001).

RELAX NG provides a DTD compatibility mode which, in addition to other things, turns on validation of (2) and (3). But in order to make this work, the schema designer has to be careful not to have two patterns that could match a particular element and attribute combination where the attribute is declared as an ID in one pattern and as something else in another. When this happens an error is reported in the schema itself, and no validation of the document instance takes place.[36]

TEI P5 goes to great lengths to ensure that the vanilla `tei_all.rng` schema does not have any such cases of ambiguity: when using a vanilla `tei_all.rng` schema, every instance element is matched by one and only one pattern. But `tei_customization` does not have this limitation. Thus if a user attempts to validate a document against `tei_customization` with DTD compatibility mode turned on, she will get the "conflicting ID-types for attribute ..." error, and her document will not be tested for validity.

There is one and only one attribute in the entire TEI scheme that is declared as type ID, namely `@xml:id`. And (as mentioned above), there are no attributes in the TEI scheme of type IDREF nor IDREFS. So it would be very reasonable to think that, for TEI, this is a non-problem: just don't use DTD compatibility mode.[37] However, there are two arguments against this idea, the first is very minor, but the second one is serious enough that it is likely to trip up users of `tei_customization` in the oXygen framework.

The first argument against just turning off DTD compatibility mode is that it means losing the other features this mode provides, too. But the main feature other than ID/IDREF checking that DTD compatibility mode provides is default attribute values, which most TEI users (including most, if not all, of TEI Council) do not like or want, anyway.

The second is that ID/IDREF checking is on by default in oXygen. While turning it off is not particularly difficult,[38] there does not seem to be any obvious, easy, eye-catching way to inform a user who wants to write a customization that she has to do this. The consequences of failing to turn it off are severe, though: although completion pop-up boxes still work, validation (both automatic validation as you type and static validation, for example ⌘-⇧-V) completely stops working. Furthermore, oXygen leaves this feature on by default for a reason. Even though ID/IDREF checking itself is of almost no use to a user working with TEI P5 documents,[39] use of this feature is how oXygen goes about providing content completion pop-ups for URL attributes (like `@who`, `@who`, `@next`, `@prev`). And that is a very well-loved feature of oXygen. There are several possible solutions to this problem, each of which has its drawbacks. The TEI Council will hopefully implement one of them soon, making use of `tei_customization` from the oXygen framework much less problematic.

BIBLIOGRAPHY

Bauman, Syd and Flanders, Julia. 2004. "Odd Customizations." Presented at Extreme Markup Languages 2004, Montréal, QC, Canada, August 2004. http://conferences.idealliance.org/extreme/html/2004/Bauman01/EML2004Bauman01.html.

Bauman, Syd. 2011. "Interchange vs. Interoperability." Presented at Balisage: The Markup Conference 2011, Montréal, Canada, 2–5 August 2011. In *Proceedings of Balisage: The Markup Conference 2011*. Balisage Series on Markup Technologies, 7. https://doi.org/10.4242/BalisageVol7.Bauman01.

Birnbaum, David and Mundie, David A. 1999. "The Problem of Anomalous Data: A Transformational Approach." In *Markup Languages: Theory and Practice*, 1. 4(Fall 1999): 1–19. 1099–6622.

Burnard, Lou. 2013. "Resolving the Durand Conundrum." *Journal of the Text Encoding Initiative*, 6. https://journals.openedition.org/jtei/842. DOI: 10.4000/jtei.842.

Burnard, Lou and Rahtz, Sebastian. 2004. "RelaxNG with Son of ODD." Presented at Extreme Markup Languages 2004, Montréal, QC, Canada, August 2004. http://conferences.idealliance.org/extreme/html/2004/Burnard01/EML2004Burnard01.html; also http://projects.oucs.ox.ac.uk/teiweb/Talks/extreme2004/paper.htm.

"deprecated." *Wiktionary*. https://en.wiktionary.org/wiki/deprecated.

Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen. 1998. "Extensible Markup Language (XML) 1.0." World Wide Web Consortium (W3C). http://www.w3.org/TR/1998/REC-xml-19980210.

Flanders, Julia and Bauman, Syd. 2010. "Using ODD for Multi-purpose TEI Documentation." Presented at Digital Humanities 2010, London, UK, 7–10 July 2010. Abstract.

Piez, Wendell. "Beyond the 'descriptive vs. procedural' distinction." In *Proceedings of Extreme Markup Languages 2001*, http://conferences.idealliance.org/extreme/html/2001/Piez01/EML2001Piez01.html#t1-1. Or see instead the author's PDF.

Quin, Liam. "Suggestive Markup: Explicit Relationships in Descriptive and Prescriptive DTDs." Presented at SGML '96, Boston, MA, USA, December 1996. http://www.holoweb.net/liam/papers/1996-sgml96-SuggestiveMarkup/.

RELAX NG DTD Compatibility. James Clark and MURATA Makoto, eds. 2001. OASIS. http://relaxng.org/compatibility-20011203.html.

Sperberg-McQueen, C. Michael. "Back to the Frontiers and Edges." Closing remarks at SGML '92,Danvers, MA, USA, 29 October 1992.

van der Vlist, Eric. 2002. *XML Schema*. O'Reilly & Associates, 2002. https://docstore.mik.ua/orelly/xml/schema/appa_03.htm.

XML Schema. 2004. World Wide Web Consortium (W3C). https://www.w3.org/TR/xmlschema-0/, https://www.w3.org/TR/xmlschema-1/, and https://www.w3.org/TR/xmlschema-2/.

NOTES

1 https://www.tei-c.org/guidelines/customization/jtei/.

2 This situation, in which both a document and its schema are required in order to get the complete information set from the document, is not unique to the W3C Schema Language. The linking of a document instance to its schema, a DTD, is a requirement of SGML. In SGML, and subsequently in XML, the DTD could provide information that was not explicitly present in the document—in particular default attribute values and the expansion text of entity references. This can become a problem when a user either does not have access to the schema, or has the schema but does not realize that there is additional information in the schema that processing software will consider part of the document information.

3 http://www.schematron-quickfix.com/.

4 There are, of course, other ways an XML file may be invalid that do not have analogs in this analogy, e.g., having two elements that are allowed, but are in the wrong order.

5 Again, see Piez, 2001.

6 For a discussion about which, see Bauman, 2011.

7 Like a schema, a pharmaceutical drug has a particular intended purpose. In the United States, a drug must be approved by the federal government for that intended purpose before it can be sold for general use. Furthermore, a drug manufacturer is only allowed to *advertise* the drug for that intended, approved use. However, a physician is permitted to *prescribe* that drug for any use she deems fit, so long as said use is not unsafe or unethical. For example, the medication memantine is approved for the treatment of Alzheimer's disease; however doctors have found that it is also beneficial for the treatment of obsessive-compulsive disorder. Prescription of a medication that is safe and ethical, but not approved, is called *off-label* use.

8 Piez attributes the term *exploratory* to John Bradley via Geoffrey Rockwell.

9 Previous major releases of the TEI Guidelines P1–P3 are available from the TEI Vault, but are not available as web pages, so an interested reader would have to download the source and search for this quotation. It is readily available in P4 and in the current release of P5 (as of this writing).

10 The TAGC in SGML nomenclature.

11 TEI Working Paper ED W29 (http://www.tei-c.org/Vault/ED/edw29.tar).

12 Which, oddly, was called *Odd*, not *ODD*, back then.

13 https://music-encoding.org/.

14 http://www.tei-c.org/Vault/P5/3.6.0/doc/tei-p5-doc/en/html/FS.html and ISO 24610.

15 https://www.w3.org/TR/its20/its20.odd.

16 https://github.com/NEU-DSG/wwp-public-code-share/tree/master/miscellaneous/itsy_bitsy_teeny_weeny_simple_hypertext.

17 http://johnkeats.uvic.ca/documentation/keats.html#oddAndDocumentation.

18 For those who are wondering, the imaginary project is the Rocky Horror Picture Show Project, which uses the Rocky Horror Picture Show Markup Language.

19 http://epidoc.sourceforge.net/.

20 http://dcl.slis.indiana.edu/cbml/.

21 https://tei-c.org/extra/teiinlibraries/4.0.0/bptl-driver.html.

22 https://wwp.northeastern.edu/wwo/source/schema/wwp-store.html.

23 https://ctb.kantl.be/project/dalf/P5/DALF_P5-p0.1.zip.

24 http://www.digitalhumanities.org/dhq/.

25 http://www.deutschestextarchiv.de/.

26 Of which my favorites are Emacs/nxml, Emacs/psgml, and oXygen. But there are dozens of others including jEdit/XML, and several web-based editors.

27 Which can also be expressed in the W3C Schema language or as a DTD if for some reason RELAX NG is not available to the end user.

28 Unlike RELAX NG, ODD cannot (yet) express the more complex "co-occurrence constraints," such as "if the `@key` is `"core"`, then the list of possible values of either `@include` or `@except` is *this*; but if the `@key` is `"header"`, then the list of possible values of either `@include` or `@except` is *that*."

29 Remember, the TEI Guidelines are written in TEI. The source to all of P5 is a single TEI document, although for convenience it is split into well over 850 separate files.

30 These outputs include schemas in RELAX NG and ISO Schematron, as well as in W3C Schema language and DTD; documentation in HTML, ePUB, and PDF formats; and a set of "exemplars," one of which is `tei_customization`. Other well-known exemplars include `tei_lite, tei_tite`, and `tei_simplePrint`.

31 "Obsolescent; said of a construct in a computing language considered old, and planned to be phased out, but still available for use" (Wiktionary, 2018-11-15).

32 For example, the `tei_customization.odd` file from the 3.3.0 release can be found at http://www.tei-c.org/Vault/P5/3.3.0/xml/tei/custom/odd/tei_customization.odd; the corresponding RELAX NG schema can be found at http://www.tei-c.org/Vault/P5/3.3.0/xml/tei/custom/schema/relaxng/tei_customization.rnc.

33 And I have never seen one "in the wild"—the only one I have seen was deliberately designed to test the ODD system.

34 Validating the source of TEI P5 against `tei_customization` was a mildly amusing, if somewhat pointless, activity. The vast majority of the 7000+ errors were caused by multiple occurrences of sibling `<desc>` (~4000) and `<gloss>` (over 1500) elements to support internationalization of the TEI Guidelines. Because internationalization of a customization seems quite rare, `tei_customization` deliberately restricts glosses and descriptions to one each. The next most frequent were occurrences of `@rend` or `@style` (just over 700), and occurrences of `<classes>` without the `@mode` attribute (just over 650).

35 P1 to P4 made extensive use of ID/IDREF, but P5 uses URIs instead.

36 The error reported by `jing`, which is the RELAX NG validator used by oXygen, starts with "error: conflicting ID-types for attribute ..." hence the name of this section.

37 And in fact, that is what this author thinks is the best approach.

38 Uncheck the "Check ID/IDREF" box in the `<Options > Preferences > XML > XML Parser > RELAX NG>` pane.

39 Because P5 does not use the ID/IDREF mechanism, the only one of the three added constraints that is useful is (2), that the value of `@xml:id` is unique. This can easily be tested by other means. For proof-of-concept tests in Schematron, see `Xmlid uniqueness.sch` on the TEI wiki.

AUTHOR

SYD BAUMAN

Syd Bauman is the Senior XML Programmer/Analyst at the Northeastern University Digital Scholarship Group. Syd has been interested in descriptive markup since the mid-1980s, and in the TEI since he first thumbed through a copy of TEI P1.1 in 1990. He served as the North American Editor of the TEI from 2001 through 2007, and currently serves on the TEI Technical Council.