Journal of the Text Encoding Initiative

Issue 6 | December 2013
Selected Papers from the 2012 TEI Conference

Rebooting TEI Pointers

Hugh A. Cayless

Electronic version
URL: http://journals.openedition.org/jtei/907
DOI: 10.4000/jtei.907
ISSN: 2162-5603

Publisher
TEI Consortium

Electronic reference
Hugh A. Cayless, « Rebooting TEI Pointers », *Journal of the Text Encoding Initiative* [Online], Issue 6 | December 2013, Online since 04 September 2017, connection on 19 April 2019. URL : http://journals.openedition.org/jtei/907 ; DOI : 10.4000/jtei.907

Rebooting TEI Pointers

Hugh A. Cayless

1. Background

1 TEI documents can be thought of as having (at least) three structural layers. On the bottom level lies the text, in whatever encoding it uses. Unicode, generally UTF-8, is the most common encoding in modern TEI markup. The layer above this may be understood in terms of the native XML structures TEI employs, such as elements, text nodes, and attributes. TEI leverages the tree structure of the XML document for a lot of its semantics. Text or elements inside a `<div>`, for example, are understood to be part of the section of document that the `<div>` represents. Containment within a structure implies membership in that structure. And the correctness of this kind of structure can, to a large extent, be validated by an XML schema (without resorting to Schematron).

2 The third layer may be thought of as comprising the meta-structures of a document. By "meta-structures" I mean those document features implemented using the various linking mechanisms that TEI provides. These do not rely primarily on the tree-based XML structure of the TEI document for their meaning, but instead form a lattice of connections between components both within and outside the document. This layer most closely resembles a graph, with nodes (mainly elements) and arcs (links).

3 For example, a `<certainty>` element is thought of as attaching to some document structure and annotating it with a measure of confidence in the correctness of the assertion in that structure. For simplicity's sake, most users of TEI place `<certainty>` inside the element it comments upon, but with the `@target` and `@match` attributes, `<certainty>` can be attached to any node in the document, or even in another document. A `<ref>` or `<ptr>` links one part of a document to another part, or to a different document. The `@scheme` attribute on `<keywords>` links back to a `<taxonomy>` defined elsewhere in the header. The global linking attributes can similarly attach nodes across the normal document hierarchy. The `<relation>`

element (despite its confusing and clumsy syntax) explicitly defines a directed or undirected graph, connecting elements in or across documents.

4 These three layers may also be thought of as representing three different data types: text as stream, text as tree, and text as graph. This article will address a shortcoming of the last data type, as currently implemented in TEI, and suggest ways in which it may be remedied. There are essentially two pointing mechanisms TEI employs to implement its graph data structure: XPath, used in the `@match` attribute,[1] and URIs, which can indicate documents, or (using fragment identifiers) elements in the current document which have `@xml:id` attributes. They cannot indicate non-element portions of a document, such as ranges of text, without employing XPointer schemes, which are quite problematic as currently defined. To put it another way, the level three structure can really only reference a subset of the level two data structure, and cannot properly address the text as stream, meaning that its functionality is incomplete.

2. XPointers

5 XPointers are an extension to URLs. A URL is an address. It indicates where a resource can be retrieved on the web. If a fragment identifier (`#foo`, for example) is appended to a URL, then the URL can be said to reference the element with the ID `"foo"` within the document the URL addresses. Like fragment identifiers, XPointers can be tacked onto a URL that locates an XML document and address a location inside the document referenced by the URL. Unlike fragment identifiers, XPointers are not limited to addressing elements. Although XPointers look very much like programming language functions, they are not, because they do not dictate what happens to the things they address; they just tell you how to get there.

6 The TEI Guidelines (TEI Consortium 2012) define a set of XPointer schemes, which have been the victim of a Catch-22. The section is difficult to grasp, and as a result, most people never bother to try using it. So few people have tried using it that its problems have never been brought to light and it has never accumulated a critical mass of implementations and use-cases. TEI pointers address *nodes*, which may be elements, attributes, or text nodes; *points*, which are conceptual locations between elements or inside text nodes; and *ranges*, or fragments of document between two points.

7 The Guidelines define the current set of pointer schemes thus:

- **xpath1()**
  - Addresses a node or nodeset using the XPath syntax. (16.2.5.2 xpath1(Expr))
- **left() and right()**
  - addresses the point before (left) or after (right) a node or node set (16.2.5.3 left() and right())
- **range()**
  - addresses the range between two points (16.2.5.4 range())
- **string-range()**
  - addresses a range of a specified length starting from a specified point (16.2.5.5 string-range(fragmentIdentifier, offset [, length]))
- **match()**

- addresses a range which matches a specified string within a node (16.2.5.6 match(fragmentIdentifier, string [, index]))

> The xpath1() scheme refers to the existing XPath specification which is adopted without modification or extension.
> The other five schemes overlap in functionality with a W3C draft specification known as the XPointer scheme draft, but are individually much simpler. At the time of this writing, there is no current or scheduled activity at the W3C towards revising this draft or issuing it as a recommendation.
> (TEI Consortium 2012, section 16.2.5, http://www.tei-c.org/Vault/P5/2.2.0/doc/tei-p5-doc/en/html/SA.html#SATS)

8 The schemes as currently defined suffer from a number of problems. To begin with, the Guidelines lack examples of usage, and some of the provided examples are incorrect or confusing; namespace mapping is not dealt with at all; XPointer parameters are defined as fragment identifiers, when XPaths should probably also be allowed; and in general, the question of behavior (what precisely it is that TEI pointers address) is left unanswered.

9 In late 2011, a working group on TEI pointers was convened. The members include myself, Piotr Bański, Syd Bauman, Gabriel Bodard, Martin Holmes, and Laurent Romary. We have been discussing what to do about the Pointers section in the guidelines, starting with the development of a set of use cases and moving on to the development of a draft update to the section, which I have been editing (with feedback from the other members). Discussions are still ongoing, and the proposals documented here represent my own views, not those of the working group as a whole, though they have certainly been informed and improved by their input.

10 First and foremost, the specification needs to be made more explicit and less abstract, so that it is actually possible to develop implementations. This includes detailing what exactly a TEI pointer addresses when it indicates a point or a range, tightening up the syntax, and modifying or adding to the definitions in the specification to fill in conceptual gaps.

11 In concert with the work on the new pointer specification, I have begun developing an XPointer resolver, using Javascript and TEI Boilerplate, that can highlight a section of a document based on an XPointer in the page URI.[2] This prototype provides a mechanism for creating and resolving TEI pointers that could plug into (for example) an annotation system.

12 The revised pointer schemes in the draft proposal[3] are as follows:

- **xpath**
  - Sequence xpath(XPATH)
    - **definitions**
      - Sequence: An ordered collection of zero or more items.
        XPATH: an XPath 1.0 or 2.0 path expression that returns an element or text node. The behavior of range schemes on XPaths returning multiple nodes is undefined.
    - **comments**
      - XPaths that return atomic values (e.g. substring()) are illegal because they represent extracted values rather than locations in the source document. XPath expressions that address attribute nodes are only legal in the xpath () scheme.
- **left**

- Point left(IDREF | XPATH)
  - **definitions**
    - Point: a dimensionless position index adjacent to a node or inside a text node.
      IDREF: the value of an @xml:id attribute in the source document.
- **right**
  - Point right(IDREF | XPATH)
- **string-index**
  - Point string-index(IDREF | XPATH, OFFSET)
    - **definitions**
      - OFFSET: a positive, negative, or zero integer. An offset of 0 represents the position immediately before the first character in either the first text node child of the node addressed in the XPATH|IDREF parameter or the first following-sibling text node, if the addressed element contains no text node descendants.
- **range**
  - Sequence range(POINTER, POINTER[, POINTER, POINTER ...])
    - **definitions**
      - POINTER: IDREF | XPATH | left() | right() | string-index()
- **string-range**
  - Sequence string-range(IDREF | XPATH, OFFSET, LENGTH[, OFFSET, LENGTH ...])
    LENGTH: a positive integer denoting the length of the string being addressed.
- **match**
  - Sequence match(IDREF | XPATH, 'REGEX'[, INDEX])
    - **definitions**
      - REGEX: a regular expression as defined in *XQuery 1.0 and XPath 2.0 Functions and Operators (Second Edition)*, section 7.6.1.[4]
        INDEX: a positive integer, beginning at 1, denoting the index of the match in the set of matches which is addressed by the scheme.
    - **note**
      - The use of single quotes to delimit the REGEX means that there must be a way to escape single quotes inside the REGEX. An escape syntax like \' would work, despite being nonstandard syntax, because it could be substituted before the expression was evaluated.

13 The term "Sequence" above is based upon the definition in the XPath 2.0 Data Model, with one amendment:

> [Definition: A **sequence** is an ordered collection of zero or more items.] [Definition: An **item** is either an atomic value or a node.]
> [Definition: An **atomic value** is a value in the value space of an **atomic type**, as defined in [XML Schema].]
> [Definition: A **node** is an instance of one of the **node kinds** defined in [XQuery 1.0 and XPath 2.0 Data Model (Second Edition)].]
> (Berglund, Boag, et al. 2010, section 2, http://www.w3.org/TR/xpath20/#id-basics)

In the XPointer definition of a sequence, an "item" may be either a node or a partial text node, but not an atomic value. All of the pointers above function in broadly the same fashion: they rely on the resolution of a context node, via either XPath or the use of an ID, and then (except for `xpath()`) they offset in some way from the context node and resolve to either a point or a sequence. Changes from the original definitions include the

renaming of `xpath1()` to `xpath()`, the addition of a `string-index()` pointer capable of resolving a point inside the text stream,[5] and the addition of repeatable parameters in the `range()` and `string-range()` pointers.

14 The proposed redefinition of TEI pointers deals with points, nodes, and sequences. The latter two are well-understood parts of the XML data model. Nodes are elements, attributes, text nodes, or the document node itself. Sequences are lists of nodes or node parts. But what exactly is a point? This is tricky, because unlike nodes and sequences, they do not exist. They are a useful construct for saying where to start or stop when grabbing a chunk of document, or maybe for describing where to insert a bookmark, but they do not correspond to anything actually in an XML document, and particularly not to anything in the XML Infoset[6]. So points are rather theoretical. The old specification defines them as the positions next to elements or between characters in a text node. The old specification is not explicit on the subject, but there have to be strict limitations on where points can be. They cannot be in the middle of an element tag (i.e. between the brackets), for example. Perhaps they could reference characters inside an attribute, but attributes are outside the text stream, so it would not be possible to have a point that started inside one attribute and ended inside another, for example.

15 Points define the start and end of a range. Therefore a range is the piece of document between two points. But again, we have to be quite careful about what that means. In the draft specification, I have taken a hard line that ranges cannot contain partial elements, so, given an XML document:

```
<foo type="barbecue">
  <bar>
    <p>Here be <n xml:id="d1">dragons</n>.</p>
  </bar>
</foo>
```

and a pointer which points at something like

```
#range(left(/foo),string-index(//p,10))
```

That is, start before the `<foo>` element and end at a point ten character spaces inside the `<p>` element (skipping over tags but including the carriage returns and whitespace indenting the elements)—after the letters "dr"—which is itself inside the `<n>` element. The resulting sequence doesn't include the elements `foo`, `bar`, `p`, or `n` at all. It contains only the text nodes captured by the range. Elements are retrieved only when the whole element is part of the range. This solves a big implementation roadblock, because it is very difficult (maybe impossible) to know how to deal with a range containing what would be ill-formed markup. There is no such thing as half an element node in the XML DOM, for example. By defining exactly what a range can be—an XPath 2.0 sequence—we can eliminate that obstacle to implementation.

16 Further, an XPointer sequence must consist of nodes, as defined above, except for the first and last items in the sequence, which may be portions of text nodes. A sequence addressed by `range()`, for example, might start in the middle of one text node and end inside another. In this case, the first and last items in the sequence would be partial text

nodes. It must be understood that TEI pointers cannot address attribute nodes (except via the `xpath()` scheme) or partial element nodes (at all).

17 A real-world implementation that could resolve the example above might return the sequence of affected nodes, with the first and last paired with instructions on how to clip them (if need be), so the result would be something like

```
(TEXT("\n  "), TEXT("\n    "), TEXT("Here be "),  (TEXT("dragons"),
substring(.,0,2)))
```

i.e. a sequence of text nodes, where the last item is paired with a function that extracts an atomic value from the text node. Here I am using the `TEXT()` notation to indicate that the bracketed string is a text node in its document context rather than simply an atomic string value and the `\n` notation to indicate a newline character. Since functions are not first-class objects in XSLT 1.0 or 2.0, and therefore cannot be passed around,[7] the preceding return is technically impossible. The atomic string value returned by the substring function could be returned, however:

```
(TEXT("\n  "), TEXT("\n    "), TEXT("Here be "), (TEXT("dragons"), "dr"))
```

Unfortunately, here we run into the problem that XPath 2.0 sequences do not nest (a sequence cannot be an item in a sequence),[8] so the example above would resolve to the sequence

```
(TEXT("\n  "), TEXT("\n    "), TEXT("Here be "), TEXT("  dragons"), "dr")
```

In other words, the returned result would be the three text nodes (in their document context), plus the atomic value of the truncated final node. This sort of return would permit any operation you might want to perform on it. If you wanted to copy the elements that partially wrap these nodes, for example, you could do it by looking at the ancestors of the text nodes in the sequence. If not, you could simply ignore the penultimate item in the sequence. The precise anatomy of a sequence returned when a TEI pointer is resolved will vary depending on the context and capabilities of the XPointer implementation in question.

18 To provide a more concrete example, in the document fragment below, from http://papyri.info/ddbdp/o.leid;;24/source, addressing individual lines is difficult because they are not contained by any element. It is common practice to refer to the `<lb>` element as a surrogate for the line, but this does not actually permit the text of the line to be retrieved or manipulated directly. A range like `#range(left(//lb[@n='3']),left(//lb[@n='4']))`, however, could unambiguously address the sequence of nodes comprising line 3, even if the `<lb>` happened to be a child of another element that began in the previous line.

```
<div type="edition" xml:lang="grc" xml:space="preserve">
  <ab>
    <lb n="1"/><num value="8">η</num> Κράτης διὰ <surplus>διὰ</surplus>
    <lb n="2"/>Διονύσιος <expan><ex>δραχμαὶ</ex></expan>
    <num value="30">λ</num>
    <lb n="3"/>Πανίσκος <expan><ex>δραχμαὶ</ex></expan>
    <num value="21">κα</num> <expan><ex>ὀβολοὶ</ex></expan>
    <num value="3">γ</num> <num value="1/2">𐅵</num>
    <lb n="4"/>Ἀτ<unclear>ᾶ</unclear>ς <expan><ex>δραχμαὶ</ex></expan>
    <num value="3">γ</num>
    <lb n="5"/><unclear>Ἰσ</unclear>ά<hi rend="diaeresis">ι</hi>ς ἁλιεύς
    <num value="5">ε</num>
    ...
  </ab>
</div>
```

19 Moreover, such a pointer could be automatically created in the service of a structure that wished to link to lines of text, like a commentary. This would allow, for example, the display of the line (or part of it) in the commentary without repetition, or the highlighting of the line when a comment is selected.

20 There are some basic requirements for environments in which we wish to implement TEI pointers. All of the pointer schemes here allow the context node to be defined using XPath, so dynamic XPath resolution is essential. Already, there is a problem resulting from the paucity of implementations of XML technologies:[9] dynamic XPath evaluation in XSLT requires either the use of XSLT 1.0 with extensions (EXSLT) or the commercial-only versions of the only Free/Open Source XSLT 2.0 implementation, Saxon. So one must rely either on a commercial solution or on a custom extension. Regular expression support is a requirement for the `match()` pointer. Apart from these functional requirements, it is also clear that anything that might affect the interpretation of offset parameters must be tightly controlled. Whitespace should therefore never be treated as "ignorable", and the normalization form of the character encoding (assuming it is Unicode) should never be changed.

3. Implementing TEI Pointers

21 The TEI has satisfactorily proved by example that trees make for a very useful (though certainly not perfect) data structure for documents of many kinds. Trees have a number of advantages as a basic model for text. Their hierarchical structure can map onto the hierarchical structures we find in many texts, such as book, chapter, and paragraph. They are easy to navigate—the XML tree-structure even has a path language which can be used to address nodes in a document or retrieve values from it. With XML and XPath, we can get at pretty much any part of a TEI document that we might want. So it is fair to ask why we would need pointers.

22 The advantage of XPointers is that they make it possible to address unmarked pieces of a document, for example a name embedded in a chunk of text, or an apparatus lemma spanning a series of elements and partial text nodes. One might object that this can be done with XPath too, but in fact, it cannot. XPath does have a number of functions that facilitate the retrieval of bits of text (the substring, substring-before, and substring-after functions, for example), but, crucially, these functions return atomic values rather than

nodes in the document tree. In other words, they return the text but discard the context. This is good enough for many modes of work, but if what you actually want to do is *address* the text in its context, then XPath may not work for you, because it can only address whole nodes in their context. XPath's inability to access the text as stream means in practice that the efficacy of the TEI graph is limited.

23 The obvious response is that if one wants to say something about a chunk of text in an XML document, typically markup is put around it with an `@id` attribute that can be pointed to. This can be done even for irregular, non-tessellated document fragments by marking the start and end of the range with `milestone` elements or by linking together a series of text containers. The XPointer use case comes in when for some reason one cannot (or really does not want to) insert new elements into the target document. This might be the case if

- you want to create markup that annotates a document published by someone else, and you do not want to or cannot re-host a derivative version with your own markup added
- you are using a limited or opinionated subset of TEI but want to layer additional data on top of it as stand-off annotations
- the explicit marking of text for every purpose you want to fulfil would make the document too complex to manage.

24 Cases like stand-off markup and annotation, particularly where the needed text fragments cut across the tree structure of the document, are the areas where TEI pointers come into their own. They make it possible to mark the text stream in ways that reconfigure the document without forcing extra complexity into the second layer.

25 The management of complexity is really the fundamental argument for employing XPointers: the lowest layer of TEI, the text stream, is a very simple beast. It is just an ordered list of characters. The second layer, the tree, imposes some hierarchical structure on the stream, and may even (virtually, not actually) reconfigure it, by asserting that at certain points it forks and rejoins—this is the effect of elements like `<choice>` and `<app>`.[10] But it too is a fairly simple and rigid data structure. Too much complexity makes it hard to cope with. But the third layer, the graph, is ideally suited to representing complex sets of relationships. It is harder for a human being to read and understand the graph, but because its data structures are simple, building tools that can aid in manipulating it is not hard.

26 I noted earlier that the set of use cases I see this technology targeting are cases where you want to layer new markup on top of existing markup that you either cannot, or do not want to, change. There is a caveat built in here: as with any URI, you are taking a risk that the thing you point at will not always be there or will change. If you are using pointers with documents that are subject to change, you will need to come up with strategies for mitigating the risk of broken links. But this is true of any kind of stand-off markup.

27 Most of the time, pointing from one part of a TEI document to another, it is easy to use IDs. Simply wrap the thing in an element, give that element an `@xml:id` attribute (say, `xml:id="foo"`), and point at the ID using either a URI with an appended fragment identifier (`"#foo"`) or using an attribute whose type is IDREF. Occasionally, however, this is not convenient, and this is where TEI pointers come in. They fill in some gaps in functionality for the meta-structural layer. They are capable of addressing sections of a TEI document that cannot be straightforwardly pointed to by means of IDs or XPath alone. Put another way, TEI pointers offer a means of breaking out of the hierarchical

XML straitjacket in those cases where a tree-structure is not adequate to represent the concepts that need to be modelled.

BIBLIOGRAPHY

Berglund, Anders, Scott Boag, Don Chamberlin, Mary F. Fernández, Michael Kay, Jonathan Robie, and Jérôme Siméon, eds. 2010. *XML Path Language (XPath) 2.0 (Second Edition): W3C Recommendation 14 December 2010.* http://www.w3.org/TR/xpath20/. Last updated January 3, 2011.

Berglund, Anders, Mary Fernández, Ashok Malhotra, Jonathan Marsh, Marton Nagy, and Norman Walsh, eds. 2010. *XQuery 1.0 and XPath 2.0 Data Model (XDM) (Second Edition): W3C Recommendation 14 December 2010.* http://www.w3.org/TR/xpath-datamodel/.

Cayless, Hugh. 2013. *Draft TEI XPointer Specification*. https://github.com/hcayless/TEI_Pointers_Draft. Last updated August 26, 2013.

Kay, Michael, ed. 2012. *XSL Transformations (XSLT) Version 3.0: W3C Working Draft, 10 July 2012.* http://www.w3.org/TR/xslt-30/.

Malhotra, Ashok, Jim Melton, Norman Walsh, and Michael Kay, eds. 2010. *XQuery 1.0 and XPath 2.0 Functions and Operators (Second Edition): W3C Recommendation 14 December 2010.* http://www.w3.org/TR/xpath-functions/.

Patterson, Matt. 2013. "Where Did All the Document Kids Go? Open-source, Markup, and the Casual Developer." Presented at Balisage: The Markup Conference 2013, Montréal, Canada, August 6–9, 2013. In *Proceedings of Balisage: The Markup Conference 2013*. Balisage Series on Markup Technologies, vol. 10. doi:10.4242/BalisageVol10.Patterson01.

TEI Consortium. 2012. *TEI P5: Guidelines for Electronic Text Encoding and Interchange*. Version 2.2.0. Last updated October 25. N.p.: TEI Consortium. http://www.tei-c.org/Vault/P5/2.2.0/doc/tei-p5-doc/en/html/index.html.

NOTES

1. See http://www.tei-c.org/Vault/P5/2.2.0/doc/tei-p5-doc/en/html/ref-att.scoping.html

2. https://github.com/hcayless/tei-xpointer.js.

3. See Cayless 2013 for a stable version of the draft.

4. See Malhotra et al. 2010, section 7.6.1, http://www.w3.org/TR/xpath-functions/#regex-syntax.

5. This function could technically be fulfilled by a zero-length `string-range()`, but the specification has been tightened up so that pointers return either points or

sequences, and `string-range()` returns a sequence. Therefore a new pointer that addresses points in text nodes is necessary.

6. See http://www.w3.org/TR/xml-infoset/.

7. They are, however, in XSLT 3.0: see Kay 2012, section 1.2, http://www.w3.org/TR/xslt-30/#whats-new-in-xslt3.

8. See Berglund, Fernández, et al. 2010, section 2.5, http://www.w3.org/TR/xpath-datamodel/#sequences.

9. For a good, recent discussion of the state of open-source markup technologies, see Patterson (2013).

10. It can be argued that given the existence of the graph, this kind of reconfiguration of the text stream is a design flaw. Certainly, it is frequently inconvenient. It is this characteristic of the TEI that makes necessary the addition of repeatable parameters—and therefore the addressing of non-contiguous chunks of text—in the new `range()` and `string-range()` pointers.

ABSTRACTS

The TEI Guidelines has for years contained a section on XPointer schemes which, unfortunately, has suffered from a lack of implementations. The reasons for this include the difficulty of the concepts in this section and a lack of sufficient detail in their specification. Despite this lack of use, the author feels that TEI pointers address an important set of use cases and should receive more attention. The chief advantage of TEI pointers is that they permit stand-off markup and annotation of text that is either unmarked or marked up inappropriately for the intended annotation. Their promise is a mechanism for leveraging stand-off markup and annotations of TEI documents—for example, being able to link to annotations in a document view because those annotations point to parts of the document. This paper provides some background on TEI pointers and their possible uses, and outlines the state of efforts under way to rework this section of the Guidelines by providing a more detailed specification and reference implementations.

INDEX

Keywords: TEI, XPointer, annotation, stand-off markup