DRAFT TEI XPointer Requirements and Notes

(Hugh Cayless – philomousos@gmail.com)

See http://www.tei-c.org/release/doc/tei-p5-doc/en/html/SA.html#SATS for the original specification. There is a demo implementation available at https://github.com/hcayless/tei-xpointer.js and a working demo at http://tei.philomousos.com/.

This draft is meant as a provocation. Please do react, either by leaving comments or via email to the editor.

Notes on Requirements

- The original XPointer specification envisioned the pointers as extensions of XPath, and as such, they were meant to be actual functions. The TEI Pointers described here are a notation describing parts of XML documents. They are meant to be resolved rather than executed.
- Each pointer specification must have a full signature in the specification, that is, a listing of allowed parameters and a “return” type.
- The precise meaning and function is dependent on the context within which they are used.
- The issue of namespaces needs to be addressed, either via the xmlns() scheme or by declaring the TEI namespace as the default. I would prefer the latter. The xmlns() scheme would then only need to be employed for elements outside the TEI namespace.
- The object types for addressed parts of a document need to be clarified. I will argue for only two types: Sequences (as defined in XPath 2.0, see http://www.w3.org/TR/xpath20/#dt-sequence) and Points.
- A Range, as specified in the original pointer specification is just a Sequence, and as a result, a Range can not contain a not-well-formed document fragment. Or at least, not one that wouldn't be well-formed if it were wrapped by an element.
- Rules for how elements partially contained by a Range are to be resolved must be established. My current thinking is that partially contained elements (i.e. where only the start or end tag is inside the range) are not to be considered part of the returned sequence. Since a child node of the partially-contained parent will be a member of the addressed Sequence, it will always be possible to grab its parent, or a copy of it. An exception should perhaps be made where addressed text nodes are entirely contained by a parent element.
- We need either xpath2() in addition to xpath1() or just xpath(), which would be understood to cover both versions.
- These schemes have been most commonly thought of in the context of XInclude, i.e. they are meant to be used to import a portion of an external document during

processing. But other uses are equally possible: they could be used during XSLT processing to convert standoff markup to inline markup, for example, or in a web browser to scroll the page to the part of the document referred to by the pointer (like fragment identifiers currently work).

- As designed, the current pointer schemes (or some of them at least) are composable, i.e. you can nest them. I will argue against such nesting except in the case of range functions, which may take functions returning points as parameters.
- The first parameter of any function must be either an IDREF or an XPath (obviously only an XPath in the case of the xpath*() function(s).
- The type of the first parameter can be distinguished in the following way: an IDREF must be an NCName as defined by http://www.w3.org/TR/1999/REC-xml-names-19990114/#NT-NCName whereas an XPath expression must either begin with a "/" or be wrapped in a function, such as id(). Put more simply, XPaths will contain characters illegal in IDREFs: either "(" and ")", or "/" and so the two can always be distinguished trivially.
- left() and right() can only work on single elements or text nodes, not attributes or sequences.
- A new scheme is needed to address points inside a text node. I suggest string-index().
- If an XPath returning an element or text node is specified as the beginning or end of a range, it will be returned as part of the resulting sequence.
- range() and string-range() Pointers will need to be able to address non-contiguous sections of text because of the way TEI structures like choice and app operate on the text stream—marking places where it forks and then rejoins. A name might begin outside an app, for example, and complete in one reading, but not the lemma.

New Pointer Schemes

xpath

Sequence xpath(XPATH)
XPATH: an XPath path expression, following the latest scheme adopted by the W3C, that returns an element or text node. The behavior of range schemes on XPaths returning multiple nodes is undefined. XPaths returning 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)
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)
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 …])
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])
REGEX: a regular expression
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: Because the REGEX portion of the pointer is delimited with single quotes, those must be escaped if they occur inside the regular expression. The current escape syntax is \', so to indicate the string "don't" inside an element "foo", you could write #match(foo,'don\'t'). This is subject to change

Points and Sequences

Points are adjacent to element nodes or characters in text nodes. A Point represents a point between element nodes or individual characters in a text node. Every point is adjacent to either

characters or elements, and never to another point. In fact, in the character representation of an XML document, every position between data characters, start-tags or end-tags is a point, and there are no other points. If one treats all character content as if it were broken into single-character text-nodes, every point is definable as either

- the point preceding a node, and if that node has a predecessor in document order, then it is the same as the point following that predecessor; or
- the point following a node, and if that node has a successor in document order, then it is the same as the point preceding that successor.

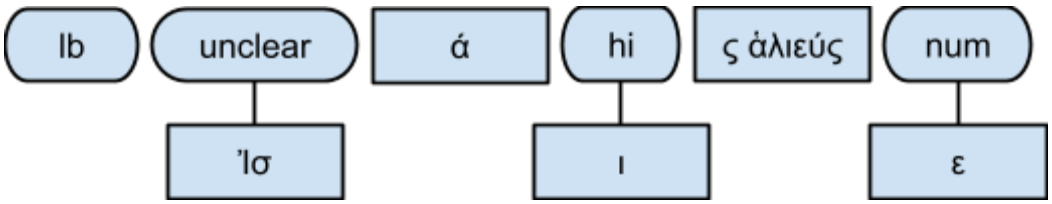
A Sequence follows the definition in the XPath 2.0 Data Model, with one change:

> [Definition: A **sequence** is an ordered collection of zero or more items.]
> [Definition: An **item** is either a partial text node (was: "an atomic value") or a node.]
> [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)].]

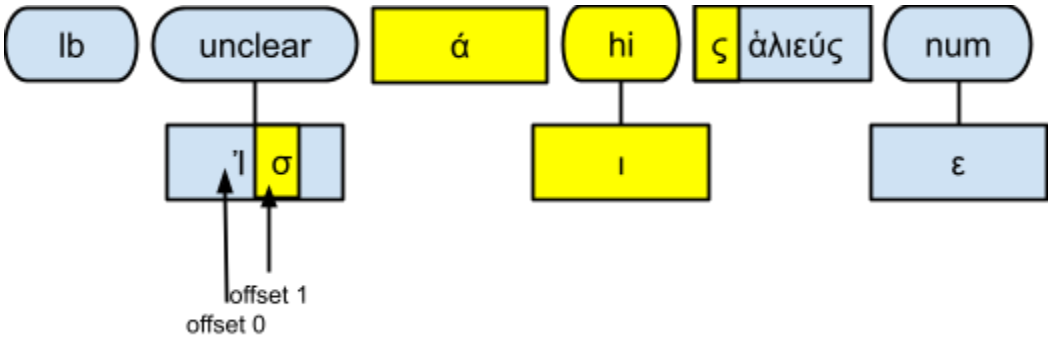
Further, an XPointer sequence must consist of nodes, as defined above, except for the first and last items in the sequence, which may be portions of text nodes. A sequence addressed by **range()** for example, might start in the middle of one text node and end inside another. In this case, the first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

```
<lb n="5"/><unclear>Ἰσ</unclear>ά<hi rend="diaeresis">ι</hi>ς ἁλιεύς <num value="5">ε</num>
```

which could be graphically represented as:



If we addressed this line using `#string-range(//lb[@n='5']/following-sibling::unclear[1], 1, 5)`, we would be addressing the sequence as highlighted below:



In other words, even though the end tag of the unclear is within the range addressed by the string-range, the unclear element is not technically a member of the range because if <unclear> were a member of the sequence, its children would necessarily come with it.

TEI XPointer functions comprise a declarative mini-language. As such, they do not directly return values and cannot be executed. They *address* pieces of XML documents, but what is to be done with those pieces is wholly dependent on the context in which they are being interpreted. So when **left()** is defined as addressing a Point, this does not mean that it is a function returning a Point. Rather, it *describes* the position immediately before the node identified in its parameter. We can imagine it as being like an instruction to a word processor to put the cursor at a particular point in a document being edited.

Notes

An algorithm for implementing range()

1. parse the range expression to retrieve the start and end points
2. parse the start point. It will be either an IDREF, an XPath, or a pointer expression
3. if the latter, unpack it to get the IDREF or XPath and the offset within or following it
4. evaluate the IDREF or XPath to get the context node
5. repeat 2-4 for the end point
6. walk down the following:: axis from the start node until the end node is reached
7. if the start and/or end nodes are to be discarded (i.e. the first parameter is a right(), or the second a left()), discard them from the nodeset.
8. If the first or second parameter is a string-index(), the text nodes at the beginning or end of the sequence will need to be truncated.

An algorithm for implementing string-range()

1. Evaluate the first parameter to obtain the text node or element it addresses and store it in a variable.
2. Find the text node containing the start index given in the second parameter. An XPath like the following should retrieve this text node:

text()[string-length(preceding::text()[preceding-sibling::node() = $node or ancestor::* = $node]) lt $position]/last() where $node is the node obtained by evaluating the first parameter of match() and $position is the position of the match.
3. Find the text node containing the end index.
4. Find all nodes (if any) between the start node and end node.
5. Truncate the start and end nodes if necessary.

An algorithm for implementing match()

1. retrieve the element or text node targeted by the fragment id or XPath in the first parameter.
2. If it contains descendant text nodes, then concatenate[1] them, otherwise, get the text descendants of nodes on the following-sibling axis and concatenate them.
3. Check whether the match regex can be found in the resulting string. If it cannot, then fail.
4. If it can be found, then get the string from the set of matches with the index specified by the index parameter, or the first match if there is none.
5. Compute the position of the match in the concatenated string (this time without eliminating whitespace around <lb break="no"/> elements).
6. Now that the start position has been determined, that plus the length of the matched string means there is sufficient information to implement the algorithm for string-range() above, starting at step 2.

[1] Concatenation may involve the removal of whitespace around <lb break="no"/> elements.