# Text Encoding Initiative

# *Electronic Textual Editing: Storage, Retrieval, and Rendering [Sebastian Rahtz]*

## Contents

Making a electronic text is sometimes regarded as an end in itself. We spend much time making decisions on *what* to encode, and *how* to formalize it, and then invest huge effort in creating electronic files representing our text. Publishing the result in some way was until recently rather hard; the amount of information encoded in a full-scale TEI text is difficult to represent in print, and we lacked easy systems to give readers access to the work. The universality of the web changes everything; we now have a rich-featured target environment in which to present text. However, print presentation remains important, and we nearly always need to rearrange texts to publish them. In this chapter, therefore, we look at:

- transformations of encoded texts
- converting TEI XML files for the web
- preparing TEI XML files for print
- database systems suitable for storing encoded text
- managing collections of text

In general, generic solutions using open standards are described, usually with open source implementations.

## Transformation

Those used to preparing texts in a word-processor environment usually expect to see an approximation of the final result on their screens as they write. Where they want to see an abstract, they write an abstract, and where the bibliography is to appear, they start typing it in. The fully-encoded TEI XML text differs from this scenario in three important (related) ways:

1. The encoder often represents two or more strands of information in a single text; marking up part of speech in a text, while also representing the logical structure, and the physical characteristics of a particular version, are reasonable things for a TEI encoder to do.
2. The TEI offers more than one way to encode information. Notes, for instance, may be embedded in the text, or placed in a section or their own and pointed to from their insertion location. The rendering sequence may not be related to the encoding sequence.
3. There is a distinct gap in many projects between the encoding, and one or more representations, and the text is better regarded as a database of information than a

linear sequence. There is no one output, and the desired results may change over time.

The effect of these differences is that rendering a TEI text very often involves rearranging or transforming it in some way. This can involve:

- Copying elements
- Removing elements
- Moving elements
- Generating elements

In the following sections, we will look at examples of each of these in turn and then consider what tools will do the work.

### *Copying elements*

The most common reason for copying elements is to create useful navigation systems like the table of contents. If the text represents an existing book, with its own contents lists, then this will not be needed, but in many circumstances we want to take a *summary* of the text and show it at the back or front. For example, if we take this text: [1]

```
<div>
<head><q>In Ambush</q></head>
<p>In summer all right-minded boys built huts in the furze-hill behind
     the College</p>
</div>
<div>
<head>Slaves of the Lamp Part I</head>
<p>The music-room on the top floor of Number Five was filled with
the ‘Aladdin’ company at rehearsal. </p>
</div>
<div>
<head>An Unsavoury Interlude</head>
<p>It was a maiden aunt of Stalky who sent him both books, with the
  inscription, ‘To dearest Artie, on his sixteenth birthday’; </p>
</div>
<head>The Impressionists</head>
<p>They had dropped into the Chaplain's study for a Saturday night
  smoke—all four house-masters—and the three briars and the one
  cigar reeking in amity proved the Rev. John Gillett's good
  generalship.</p>
```

it would be reasonable to take each of the `<head>` elements and make a list at the front of the transformed text as a list:

```
<list>
<item><q>In Ambush</q></item>
<item>Slaves of the Lamp Part I</item>
<item>An Unsavoury Interlude</item>
<item>The Impressionists</item>
</list>
```

The text nodes of each `<head>` element are copied to the `<item>` elements in the `<list>` .

This example is trivial, but the application is widespread. Texts for reading by humans very often contain duplicate material (like many other books, every even-numbered page of *Stalky & Co* repeats the title, to remind us of what we are reading). As usual in TEI work, the extent to which the encoding captures the duplication explicit in an original work very much depends on basic encoding decisions. Some TEI texts may need no transformation by copying at all.

When considering the uses of copying, it is important to remember that the `<teiHeader>` section may contain material which is needed for rendering, and will be copied into place. For instance, the human-readable version might be annotated on each page with the date on which the text was encoded.

### Removing elements

Most TEI texts contain elements which we do not need to keep as-is when rendering the material. The obvious example is the `<teiHeader>` containing metadata, most of which is not needed for typical rendering, but we may also choose:

- not to show back matter or front matter
- to show only a sample of the text divisions
- to remove editorial notes
- To suppress detailed metadata resulting from linguistic analysis
- To remove stage directions from a play
- To show only those portions of text relating to a particular speaker or character

As a simple example, we might choose to transform this:

```
<p>They had dropped into the Chaplain's study for a Saturday night
  smoke—all four<note place='end' type="editorial">King, Prout, White, Hartopp;
  but sometimes Macrea is mentioned as a house-master as well.  Is
  this reflecting the passage of time?</note>house-masters—and the
  three briars<note place="end"
  type="gloss">A briar is a type of pipe</note> and
  the one cigar reeking in amity proved the
  <rs key="JG11" reg="Gillett, John" type="person">
  <name>Rev. John Gillett</name></rs>'s good
  generalship.</p>
```

into this

```
<p>They had dropped into the Chaplain's study for a
  Saturday night smoke—all four house-masters—and the three
  briars<note place="end" type="gloss">A briar is a type of pipe</note>
  and the one cigar reeking in amity proved the Rev. John Gillett's good
  generalship.</p>
```

by suppressing one type of `<note>` and removing the metadata about the chaplain which might be used in other circumstances to *only* show paragraphs in which he is mentioned.

### Moving elements

It may seem at first sight strange to propose re-ordering our carefully-assembled text. But there are several occasions when we may wish to do so. For example, a text which exists in several versions may be encoded as set of separate `<text>` elements, but for presentation purposes we want to show each paragraph together with its variants; so the sequence of paragraphs A1, A2, A3, A4, A5, B1, B2, B3, B4, B5 will become A1, B1, A2, B2, A3, B3, A4, B4, A5, B5, as the two versions are interleaved.

Another situation when movement of elements might be appropriate is when bibliographical citations are placed in their own section in the `<back>` element, but are to be presented as footnotes. Thus the combination of

```
<xref type="cite" doc="BIB" from="id (TEIP4)">(TEI, 2002)</xref>
```

and later

```
<bibl id="TEIP4">TEI <title level="m">Guidelines for
Electronic Text Encoding and Interchange (TEI P4)</title>.
Sperberg-McQueen, C.M., Lou Burnard, Steve DeRose, and Syd Bauman,
eds. Bergen, Charlottesville, Providence, Oxford: Text Encoding
Initiative Consortium, <date>2002</date>.</bibl>
```

is turned into

```
(TEI,2002)<note><bibl id="TEIP4">TEI <title
level="m">Guidelines for Electronic Text Encoding and Interchange (TEI
P4)</title>.  Sperberg-McQueen, C.M., Lou Burnard, Steve DeRose, and
Syd Bauman, eds. Bergen, Charlottesville, Providence, Oxford: Text
Encoding Initiative Consortium, <date>2002</date>.</bibl></note>
```

by a process of interleaving.

Naturally, any sorting of elements is a process of moving, and any merging of material from a secondary file will involve movement of material.

An extreme form of moving (and copying) material is the rearrangement of a text into a concordance or wordlist.

### Generating elements

We have already seen above some generation of new elements; for instance, when we copied `<head>` elements to a series of list items, we had to create the surrounding `<list>` . There is also a TEI element `<divGen>` which explicit expects a rendering application to do some work and create material. The action is determined by the **type** attribute, for which TEI P4 suggests possible values:

**index**

an index is to be generated and inserted at this point.

**toc**
: a table of contents

**figlist**
: a list of figures

**tablist**
: a list of tables

These actions again involve element copying of some kind; the ‘index’ value may involve working out index entries, for instance, from all the `<name>` elements in a text.

A common form of generation is the addition of automated linking, or the making explicit of implicit information. Thus the text used above to identify a character:

```
<rs key="JG11" reg="Gillett, John" type="person">
  <name>Rev. John Gillett</name></rs>
```

could be the key to generating a link to the section of the document where Gillett is first mentioned, or to detailed metadata about the person referred to, perhaps in a bibliography.

## The toolkit

Unless we want to simply turn a linear sequence of TEI XML elements into exactly the same sequence of material marked up in some other way, most real-life applications involve a combination of all the techniques discussed above. It makes sense to consider all these steps *before* we worry about how precisely we want to publish the result; all of the examples above apply just as much to print as to the web, and we can operate more efficiently if we (conceptually, at least) perform all the transformations at the level of TEI XML before converting to a particular result.

A wide range of software tools can do the transformation for us. Any programming or scripting language which can handle XML has the power to do almost all the tasks, but unless you are an experienced programmer, it makes sense to stick to the Recommendations defined by the World Wide Web Consortium. There are three useful ‘standards’ from the W3C which are relevant here:

**DOM**
: A specification for how programming languages should provide access to, and manipulate, the tree structure of an XML document

**XPath**
: A specification for how to address parts of an XML document

**XSLT**
: A high-level language for specifying transformations, using XPath

For our purposes XSLT (well described in many reference and tutorial texts, including Kay and Tennison as well as the extensive pointers at http://www.xslt.com/ ) fits the bill nicely. It is available in a number of efficient implementations for most computer platforms and environments, including web servers and clients, editors, as well as free-standing programs. The commonly used implementations are MSXML, Saxon, Xalan, libxslt and Sablotron; they are mostly open source and/or free. The language has some interesting and important characteristics:

- It is expressed in XML. You can write scripts using your existing XML editor and use your existing syntax-checking tools.
- It is based on the use of a series of templates, or rules, which specify what is to happen to particular elements in from the incoming text; the processor works out how to apply the rules.
- Using the XPath language, any template can access any part of the document at any time. You are not constrained by a linear processing model.
- XML namespaces are used to differentiate the XSLT language itself from the output.

The result is that it is simple to implement all of the techniques we have discussed.

It is beyond the scope of this chapter to provide a detailed introduction to XSLT, but it should be helpful to see some simple scripts.

Firstly, a script which takes each `<div>` element and puts the corresponding `<head>` element as an item in list:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">

  <xsl:template match="body">
    <list>
    <xsl:for-each select=".//div">
      <item><xsl:apply-templates select="head"/></item>
    </xsl:for-each>
   </list>
   <xsl:apply-templates/>
  </xsl:template>

</xsl:stylesheet>
```

Note the **xsl:** prefix for statements which are part of the XSLT language; other elements, such as `<list>` and `<item>` are placed in the output. The fragment has a single `<xsl:template>` , whose **select** attribute identifies which input element (here, `<body>` ) the rule applies to. The `.//div` value for the **select** attribute of **xsl:apply-templates** is an expression in the XPath language; it means ‘every div element you can find below here in the document tree’. The final `<xsl:apply-templates>` tells the processor to carry on and process (again, in the case of `<div>` s) all the elements inside the current element (the `<body>` ).

Secondly, a script fragment (we omit the wrapper `<xsl:stylesheet>` element from here onwards) which *removes* some elements:

```
<xsl:template match="note[@type='editorial']"/>

<xsl:template match="teiHeader"/>
```

By simply supplying no body for the templates for `<teiHeader>` , and `<note>` elements with a **type** attribute of `editorial`, the entire element and its contents are passed-by.

Thirdly, to *move* some elements, we specify explicitly for an element ( `<text>` ) what to do; here, we say that we want the `<back>` , `<front>` and `<body>` elements to be processed in that order:

```
<xsl:template match="text">
    <xsl:apply-templates select="back"/>
    <xsl:apply-templates select="front"/>
    <xsl:apply-templates select="body"/>
</xsl:template>
```

Note that if we omit one of the `<xsl:apply-templates>` lines, the corresponding elements would never be processed.

Lastly, the *generation* of elements. Our example of copying elements already showed how to make a crude table of contents; the following script appends a `<note>` element to each `<name>` listing the number of times that name occurs in the text:

```
<xsl:key name="NAMES" match="name" use="@key"/>

<xsl:template match="name">
  <xsl:apply-templates/>
  <note>This name occurs
  <xsl:value-of select="count(key('NAMES',@key))"/> times</note>
</xsl:template>
```

The `<xsl:key>` element keeps a table in memory of all the occurences of the `<name>` element, indexed by the value of the **key** attribute; the **key** function retrieves a list from that table, and the **count** function counts them.

## Making webs

If we have arrived at a transformation of a text which has the correct components in the right sequence, we can start to render it in a form suitable for human readers. The most common way to achieve this is to convert it to HTML; thus the TEI markup

```
<div>
<head><q>In Ambush</q></head>
<p>In summer all right-minded boys built huts in the furze-hill behind
     the College</p>
</div>
```

might become

```
<h1>‘In Ambush’</h1>
<p>In summer all right-minded boys built huts in the furze-hill behind
     the College</p>
```

This is, of course, no different from the transformations we discussed in the last section; the `<head>` element of the `<div>` is used to construct the HTML `<h1>` element, and the TEI `<p>` is turned into the similar HTML `<p>` . Note also the simplification of the `<q>` element, which is a replaced by a matching pair of single quotes.

This conversion to HTML involves no more nor less than the transformation using copying, moving, removing and generating which we described in the previous section, if we regard the HTML language as another XML dialect. Luckily, all the XSLT implementations have a special output mode in which they massage their output into HTML; if the transformation turns a TEI `<lb>` into `<br/>` (which it has to do, because the script must be well-formed XML), the XSLT engine quietly makes that the standard HTML `<br>` , to satisfy older Web browsers.

Although we describe it here as two stages, transformation from TEI to TEI followed by transformation to HTML, most applications do both at the same time, so

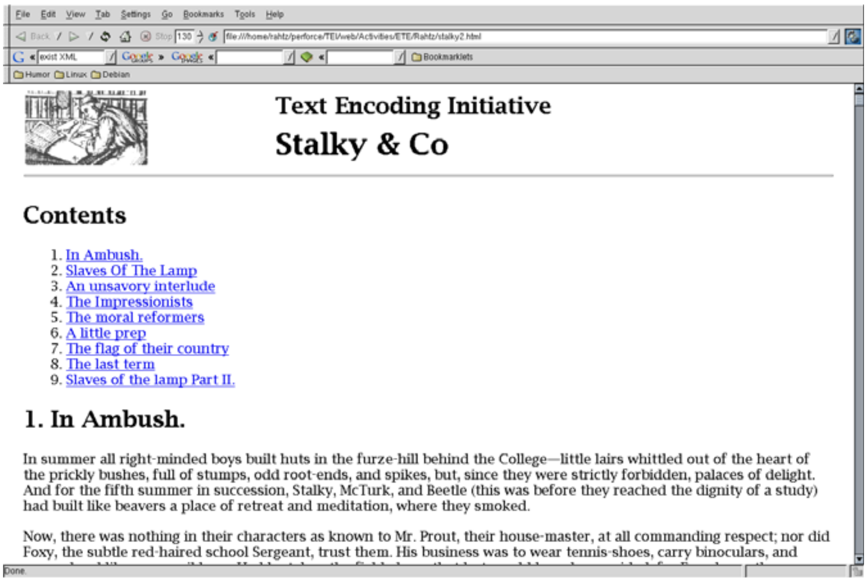
```
<xsl:template match="body">
    <list>
    <xsl:for-each select=".//div">
      <item><xsl:apply-templates select="head"/></item>
    </xsl:for-each>
   </list>
</xsl:template>
```

would be written in XSLT as

```
<xsl:template match="body">
    <ol>
    <xsl:for-each select=".//div">
      <li><xsl:apply-templates select="head"/></li>
```

```
     </xsl:for-each>
   </ol>
</xsl:template>
```

Writing a set of XSLT specifications for a markup scheme as large as the TEI is not a trivial task to undertake. It is likely, for instance, that you will want to create a *set* of HTML files, rather than just one, in which case you need to create lots of navigational links, and take care of internal cross-references which go from one generated output file from another. For example, shows a simple rendition of a TEI text, with a table of contents at the start and then the sections following on, while shows a delivery in which chapters are presented in individual documents with a table of contents on the left.


File Edit View Tab Settings Go Bookmarks Tools Help
Back Stop 130
file:///home/rahtz/perforce/TEI/web/Activities/ETE/Rahtz/stalky2.html
exist XML
Google
Bookmarklets
Humor Linux Debian
Done.

Text Encoding Initiative

Stalky & Co

Contents

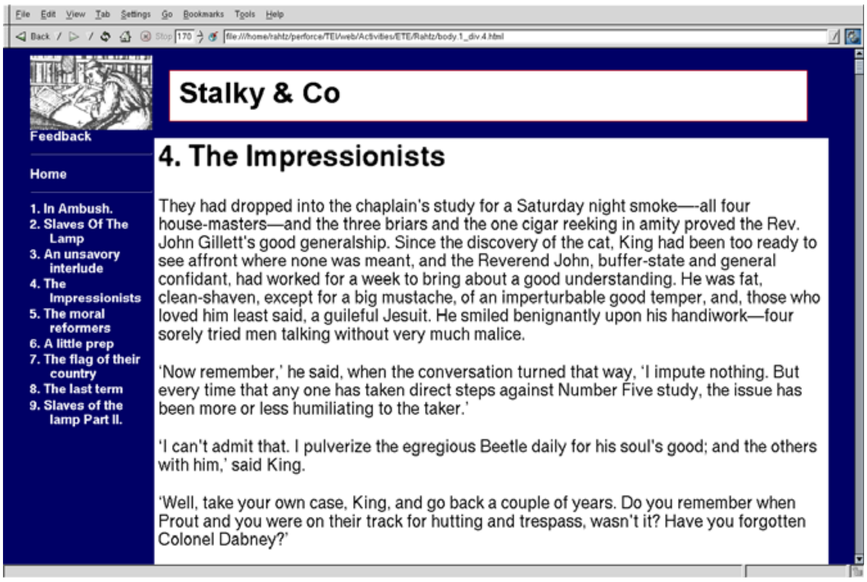
1. In Ambush.
2. Slaves Of The Lamp
3. An unsavory interlude
4. The Impressionists
5. The moral reformers
6. A little prep
7. The flag of their country
8. The last term
9. Slaves of the lamp Part II.

1. In Ambush.

In summer all right-minded boys built huts in the furze-hill behind the College—little lairs whittled out of the heart of the prickly bushes, full of stumps, odd root-ends, and spikes, but, since they were strictly forbidden, palaces of delight. And for the fifth summer in succession, Stalky, McTurk, and Beetle (this was before they reached the dignity of a study) had built like beavers a place of retreat and meditation, where they smoked.

Now, there was nothing in their characters as known to Mr. Prout, their house-master, at all commanding respect; nor did Foxy, the subtle red-haired school Sergeant, trust them. His business was to wear tennis-shoes, carry binoculars, and

Figure 1. Simple web rendition of TEI document


File Edit View Tab Settings Go Bookmarks Tools Help
Back Stop 170
file:///home/rahtz/perforce/TEI/web/Activities/ETE/Rahtz/body.1_div.4.html
Feedback
Home

1. In Ambush.
2. Slaves Of The Lamp
3. An unsavory interlude
4. The Impressionists
5. The moral reformers
6. A little prep
7. The flag of their country
8. The last term
9. Slaves of the lamp Part II.

Stalky & Co

4. The Impressionists

They had dropped into the chaplain's study for a Saturday night smoke—-all four house-masters—and the three briars and the one cigar reeking in amity proved the Rev. John Gillett's good generalship. Since the discovery of the cat, King had been too ready to see affront where none was meant, and the Reverend John, buffer-state and general confidant, had worked for a week to bring about a good understanding. He was fat, clean-shaven, except for a big mustache, of an imperturbable good temper, and, those who loved him least said, a guileful Jesuit. He smiled benignantly upon his handiwork—four sorely tried men talking without very much malice.

'Now remember,' he said, when the conversation turned that way, 'I impute nothing. But every time that any one has taken direct steps against Number Five study, the issue has been more or less humiliating to the taker.'

'I can't admit that. I pulverize the egregious Beetle daily for his soul's good; and the others with him,' said King.

'Well, take your own case, King, and go back a couple of years. Do you remember when Prout and you were on their track for hutting and trespass, wasn't it? Have you forgotten Colonel Dabney?'

Figure 2. Multi-part web rendition of TEI document

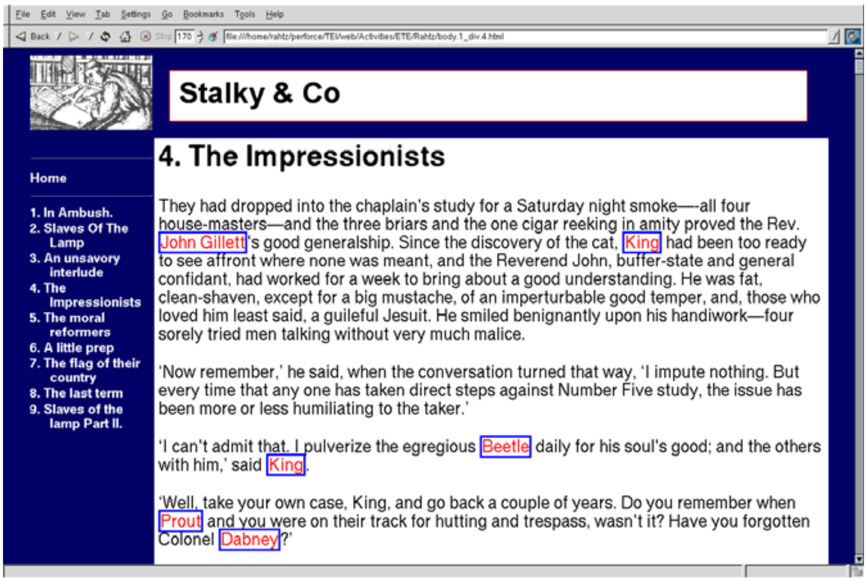
Obviously there is an enormous number of other web display features which can be driven by TEI markup or by editorial choice. Thus shows a visual rendition of the `<name>` element, from this XSLT:

```
<xsl:template match="name">
  <span class="name"><xsl:apply-templates/></span>
</xsl:template>
```

shows the `<foreign>` elements with different colours according to the **type** attribute, using this XSLT fragment:

```
<xsl:template match="foreign">
  <span class="foreign{@lang}"><xsl:apply-templates/></span>
</xsl:template>
```

in which a CSS **class** attribute is created using the word `foreign` and the value of **lang**. The key to these simple results, and very much more complex ones, is deciding how HTML markup in the output can be created by transforming input XML.


File
Edit
View
Tab
Settings
Go
Bookmarks
Tools
Help
Back
Stop
170
file:///home/rahtz/perforce/TEI/web/Activities/ETE/Rahtz/body.1_div.4.html

Stalky & Co

Home

1. In Ambush.
2. Slaves Of The Lamp
3. An unsavory interlude
4. The Impressionists
5. The moral reformers
6. A little prep
7. The flag of their country
8. The last term
9. Slaves of the lamp Part II.

4. The Impressionists

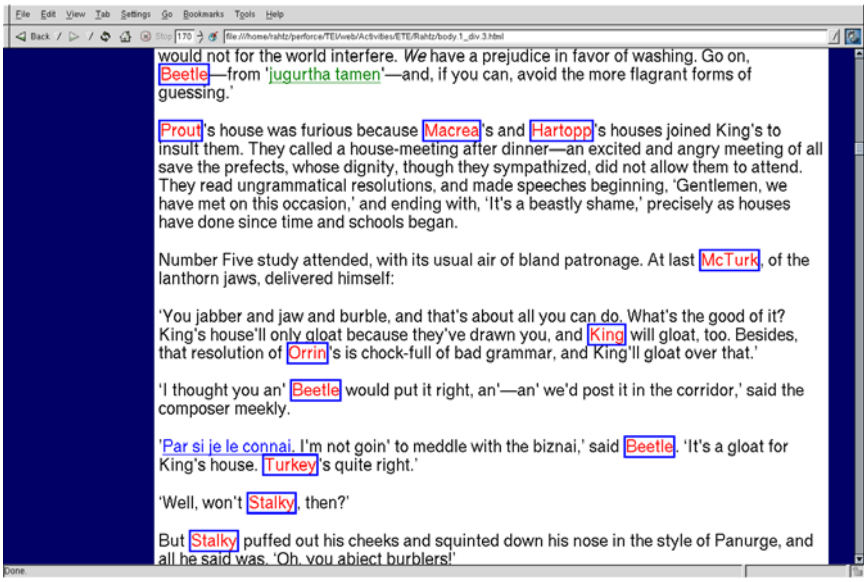
They had dropped into the chaplain's study for a Saturday night smoke—all four house-masters—and the three briars and the one cigar reeking in amity proved the Rev. John Gillett's good generalship. Since the discovery of the cat, King had been too ready to see affront where none was meant, and the Reverend John, buffer-state and general confidant, had worked for a week to bring about a good understanding. He was fat, clean-shaven, except for a big mustache, of an imperturbable good temper, and, those who loved him least said, a guileful Jesuit. He smiled benignantly upon his handiwork—four sorely tried men talking without very much malice.

'Now remember,' he said, when the conversation turned that way, 'I impute nothing. But every time that any one has taken direct steps against Number Five study, the issue has been more or less humiliating to the taker.'

'I can't admit that. I pulverize the egregious Beetle daily for his soul's good; and the others with him,' said King.

'Well, take your own case, King, and go back a couple of years. Do you remember when Prout and you were on their track for hutting and trespass, wasn't it? Have you forgotten Colonel Dabney?'

Figure 3. Web rendition of TEI document highlighting names


File
Edit
View
Tab
Settings
Go
Bookmarks
Tools
Help
Back
Stop
170
file:///home/rahtz/perforce/TEI/web/Activities/ETE/Rahtz/body.1_div.3.html
Done.

would not for the world interfere. *We* have a prejudice in favor of washing. Go on, Beetle—from 'jugurtha tamen'—and, if you can, avoid the more flagrant forms of guessing.'

Prout's house was furious because Macrea's and Hartopp's houses joined King's to insult them. They called a house-meeting after dinner—an excited and angry meeting of all save the prefects, whose dignity, though they sympathized, did not allow them to attend. They read ungrammatical resolutions, and made speeches beginning, 'Gentlemen, we have met on this occasion,' and ending with, 'It's a beastly shame,' precisely as houses have done since time and schools began.

Number Five study attended, with its usual air of bland patronage. At last McTurk, of the lanthorn jaws, delivered himself:

'You jabber and jaw and burble, and that's about all you can do. What's the good of it? King's house'll only gloat because they've drawn you, and King will gloat, too. Besides, that resolution of Orrin's is chock-full of bad grammar, and King'll gloat over that.'

'I thought you an' Beetle would put it right, an'—an' we'd post it in the corridor,' said the composer meekly.

'Par si je le connai. I'm not goin' to meddle with the biznai,' said Beetle. 'It's a gloat for King's house. Turkey's quite right.'

'Well, won't Stalky, then?'

But Stalky puffed out his cheeks and squinted down his nose in the style of Panurge, and all he said was, 'Oh, you abject burblers!'

---

Figure 4. Web rendition of TEI document highlighting foreign words and names

The TEI Consortium hosts a set of XSLT stylesheets by Sebastian Rahtz on its web site at http://www.tei-c.org/Stylesheets which are designed for creating web pages. The results can be varied by providing values for over 80 parameters, and by overriding individual templates.

XSLT stylesheets for TEI documents can be applied in a number of ways:

1. A reference to the stylesheet can be embedded in the XML file using a processing instruction. Some web browsers, including Internet Explorer and the Mozilla family, can then view the XML file directly by transforming it to HTML dynamically. This is a very easy and effective way to publish TEI XML documents, if you can be sure the potential readers use the the right web client.
2. The web server can be set up to transform the XML to HTML on request; the Cocoon and AxKit systems are examples of this. They trap incoming requests for XML documents, perform the transform, and return the HTML results. As the results are cached, this can be very efficient, and has the advantage that it works with any browser. The disadvantage is that the reader does not get access to the original XML (this may also be seen as an advantage). Different renditions can be made on the server by providing appropriate instructions as part of the URL (ie a URL ending in `xyz.xml?style=print` will be processing to pass the parameter **style** with the value ‘print’ to the stylesheet engine).
3. The XML can be transformed to HTML once at source, and the result uploaded to a normal web server. This is the most reliable, and causes the least load on the server, but it is also the most cumbersome, as the text owner has to remember to remake the HTML each time there is a change in the original. Any different renditions also have to be precreated.

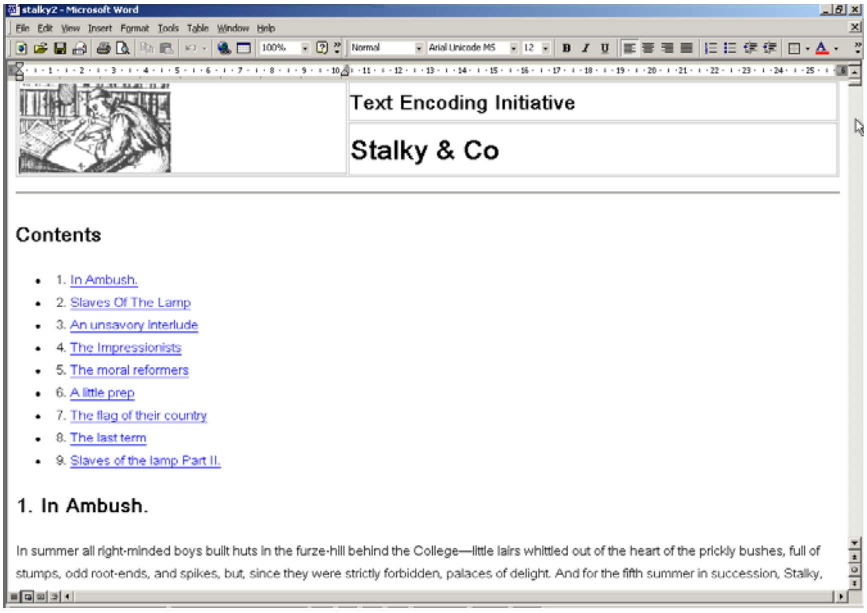
## Preparing for print

It should be clear by now that the print version of our texts follows much the same line as the web version. First, we define the transformations needed to present the right bits of text in the right order, and then we turn it into a print format. This is likely to be Portable Document Format (PDF), created by a page makeup engine. The choice here is not as clear as it is for the web, however. We may prefer any of:

1. A dedicated publishing system with its own system of stylesheets and/or sophisticated interfaces for making up pages; Quark Express and InDesign are good examples.
2. Commercial or free systems which can create typeset pages from a semi-fixed page specification (XSL FO or DSSSL)
3. Conversion to a well-understood non-XML formatting language, such as LaTeX.

Which you use depends on whether you want to feed material to a typesetting system where you can subsequently fine-tune and manipulate the results, or whether you require automated page makeup with little need for intervention.

If you plan to use a traditional desktop-publishing system, one simple way to proceed is to turn your TEI file into simple HTML and then read it into Microsoft Word or another similar word-processor; many DTP programs can import such files. shows our sample text imported into Word.



Text Encoding Initiative

Stalky & Co

Contents

- 1. In Ambush.
- 2. Slaves Of The Lamp
- 3. An unsavory interlude
- 4. The Impressionists
- 5. The moral reformers
- 6. A little prep
- 7. The flag of their country
- 8. The last term
- 9. Slaves of the lamp Part II.

1. In Ambush.

In summer all right-minded boys built huts in the furze-hill behind the College—little lairs whittled out of the heart of the prickly bushes, full of stumps, odd root-ends, and spikes, but, since they were strictly forbidden, palaces of delight. And for the fifth summer in succession, Stalky,

Figure 5. TEI document converted to HTML and imported into Word

If this method is not suitable, the proprietary methods of the system will have to be used. Some programs support XSLT as part of their process (eg Arbortext Epic, or Advent 3B2).

The XSL Formatting Objects language (XSL FO for short; see (XSLFO), and many pointers to software and tutorials at http://www.w3.org/Style/XSL/ ) is a W3C Recommendation which provides a good basis for automated page makeup. It is a markup, expressed as XML, which describes a set of pages and the characteristics of layout objects on those pages. The core principles of the language design were

- conceptual compatibility with DSSSL;
- compatibility with CSS properties;
- screen properties as well as print; and
- no compromises on internationalization (ie many writing directions are supported)

When using FO, the input XML is transformed into an output tree consisting of:

- **page masters**, which define named styles of page layout; and
- **page sequences**, which reference a named page layout and contain a **flow** of text. Within that **flow**, text is assigned to one of five (rectangular) **regions** (the page body, areas at the top, bottom, left and right)

There is also allowance for floating objects (at the top of the page), and footnotes (at the bottom), and the model covers writing in left to right, right to left, top to bottom, and bottom to modes.

Within a region of text, we find one or more

- **blocks**,
- **tables**,
- **lists**, and
- **floats**

and within a block, we find

- **inline sequences**,

- **characters**,
- **links**,
- **footnotes**, and
- **graphics**

All of these objects have a large number of properties, including

- aural properties,
- borders, spacing and padding,
- breaking,
- colors,
- font properties (family, size, shape, weight etc),
- hyphenation,
- positioning,
- special table properties, and
- special list properties

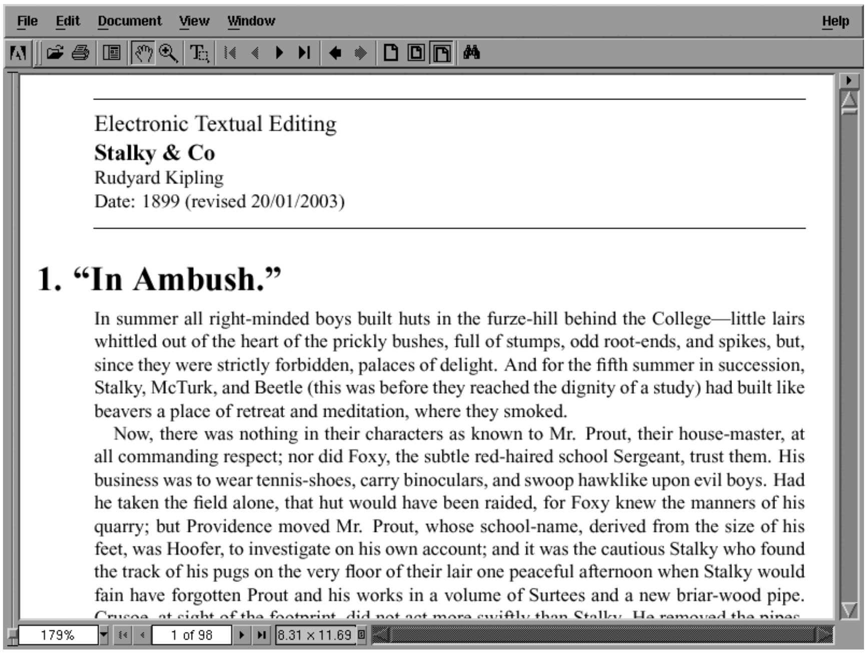
although supporting all of them is not mandatory for processors.

XSL FO does not specify where line- or page-breaks are to occur, or precisely how to justify a paragraph, so the results from different processors can vary. A sample of XSL FO markup is:

```
<fo:block font-size="10pt"
 text-indent="1em"
 font-family="Times-Roman"
 space-before.optimum="0pt"
 space-before.maximum="12pt">&#x2018;<fo:inline
 font-style="italic">Ti-ra-ra-la-i-tu</fo:inline>! I gloat! Hear
me!&#x2019; Stalky, still on his heels, whirled like a dancing dervish
to the dining-hall.</fo:block>
```

in which a `<block>` object is described, containing an embedded `<inline>` object. The concept, and some of the attribute names, will be to familiar to web developers from the CSS language.

There are several good implementations of XSL FO, both commercial and free (XEP, Antenna House XSL Formatter, FOP, and Arbortext EPIC) most of which take an FO document and produce PDF pages some can generate other formats, including PostScript, TIFF, and Windows GDI). shows our sample text displayed in Acrobat after conversion to XSL FO and then formatted to PDF using PassiveTeX; shows the same FO file formatted and displayed by the Antenna House formatter; the differences are small, but instructive. Note that the line-endings are not the same, for instance.



Electronic Textual Editing
**Stalky & Co**
Rudyard Kipling
Date: 1899 (revised 20/01/2003)

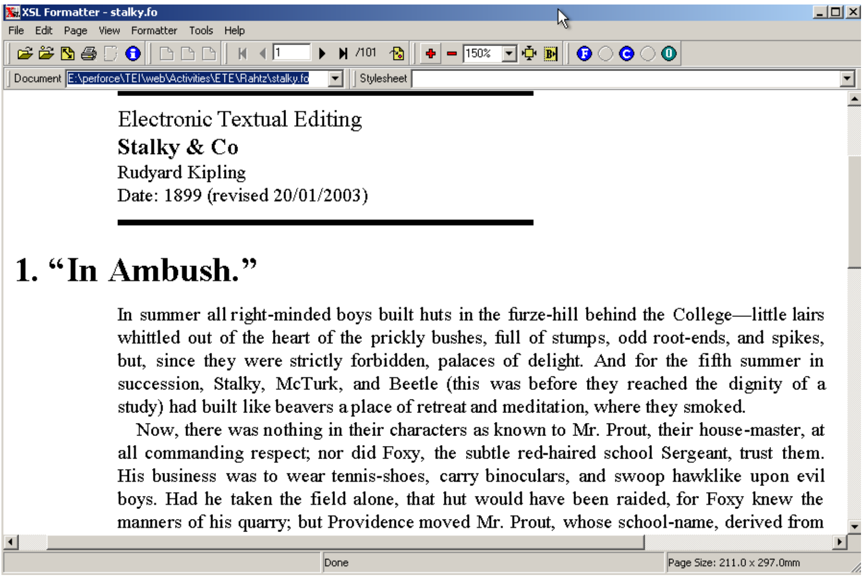
**1. “In Ambush.”**

In summer all right-minded boys built huts in the furze-hill behind the College—little lairs whittled out of the heart of the prickly bushes, full of stumps, odd root-ends, and spikes, but, since they were strictly forbidden, palaces of delight. And for the fifth summer in succession, Stalky, McTurk, and Beetle (this was before they reached the dignity of a study) had built like beavers a place of retreat and meditation, where they smoked.

Now, there was nothing in their characters as known to Mr. Prout, their house-master, at all commanding respect; nor did Foxy, the subtle red-haired school Sergeant, trust them. His business was to wear tennis-shoes, carry binoculars, and swoop hawklike upon evil boys. Had he taken the field alone, that hut would have been raided, for Foxy knew the manners of his quarry; but Providence moved Mr. Prout, whose school-name, derived from the size of his feet, was Hoofer, to investigate on his own account; and it was the cautious Stalky who found the track of his pugs on the very floor of their lair one peaceful afternoon when Stalky would fain have forgotten Prout and his works in a volume of Surtees and a new briar-wood pipe. Crusoe, at sight of the footprint, did not act more swiftly than Stalky. He removed the pipes

179% 1 of 98 8.31 x 11.69

Figure 6. TEI document converted to XSL FO and then PDF using PassiveTeX



Document E:\perforce\TEI\web\Activities\ETE\Rahtz\stalky.fo

1 /101 150%

Electronic Textual Editing
**Stalky & Co**
Rudyard Kipling
Date: 1899 (revised 20/01/2003)

**1. “In Ambush.”**

In summer all right-minded boys built huts in the furze-hill behind the College—little lairs whittled out of the heart of the prickly bushes, full of stumps, odd root-ends, and spikes, but, since they were strictly forbidden, palaces of delight. And for the fifth summer in succession, Stalky, McTurk, and Beetle (this was before they reached the dignity of a study) had built like beavers a place of retreat and meditation, where they smoked.

Now, there was nothing in their characters as known to Mr. Prout, their house-master, at all commanding respect; nor did Foxy, the subtle red-haired school Sergeant, trust them. His business was to wear tennis-shoes, carry binoculars, and swoop hawklike upon evil boys. Had he taken the field alone, that hut would have been raided, for Foxy knew the manners of his quarry; but Providence moved Mr. Prout, whose school-name, derived from

Done

Page Size: 211.0 x 297.0mm

Figure 7. TEI document converted to XSL FO and then PDF using Antenna House XSL Formatter

The chief disadvantage of the XSL FO approach is that it neither allows direct manipulation or specification of the page, nor guarenteed page fidelity. You cannot say ‘I always want this figure on page 54, half way down, with text flowing around it, regardless of how much the text changes’, and neither can you be *sure* that your pages will have same breaks if you use two different implementations. In practice, TEI documents tend to be neither designed for the page, or require legal precision, so FO is a reasonable choice.

XSL FO is generally created by a transformation from higher-level XML; it is highly verbose and hard to write by hand. XSLT is again a good tool to perform this transformation, as the task is very similar to the creation of HTML. The TEI Consortium again hosts a set of parameterized XSLT stylesheets by Sebastian Rahtz on its web site at http://www.tei-c.org/Stylesheets which can be used to create XSL FO from TEI XML. These allow for obvious layout changes including

- change of page-size and basic font-size;
- size and style of headings;
- whether or not headings are numbered;
- single or double column;
- management of cross-references.

Orthogonal to the methodology adopted (automated or interventionist) is the method of software integration. Some systems offer closely integrated publishing solutons (eg Arbortext's Epic, http://www.arbortext.com/html/products.html ), which cover all aspects of the document management process, including editing, stylesheets, print publication, web publication, database integration etc. The alternative is a set of loosely-related solutions for each situation along the line, perhaps drawn from both commercial and open-source software. Both approaches are equally valid, and generally speaking the TEI encoder can be slightly relaxed, since the overwhelming effort is in creating the neutral XML source in the first place.

## Database systems for TEI and XML

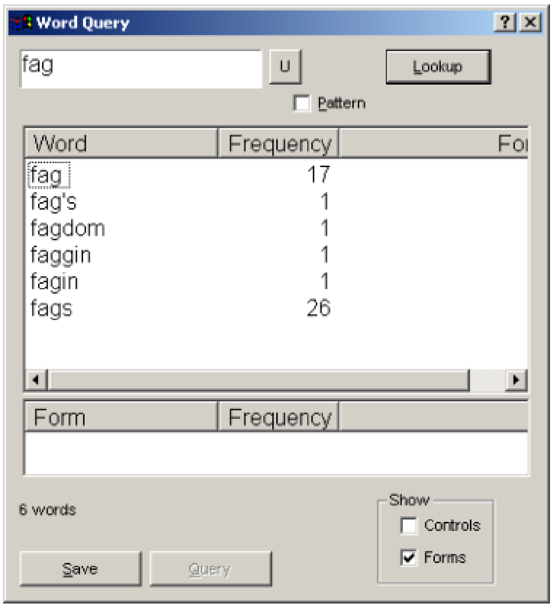
Our discussion so far has been on the level of simple XML files, and manipulating them individually. What if we want to present composite results from a whole collection of TEI texts? On a small scale, this can be done with XSLT, since that language allows for reading multiple input files, selecting arbitrary subsets, and making a composite output. However, all current XSLT implementations work in memory, and read the documents afresh each time a script is run. This makes the technique unsuitable for documents over a certain size, or when many queries need to be run at the same time; in that situation we need to turn to a dedicated database.

Database systems have two important characteristics for our purposes; firstly they do not store all the data in memory, and are designed to cope with arbitrarily large collections, with equally efficient acecss to any part; secondly, they support multiple simultaneous queries. How do they work in the XML context? Ronald Bourret (at http://www.rpbourret.com/xml/XMLAndDatabases.htm ) has an exhaustive discussion of the details; here we summarize four simple ways to proceeed:

1. We can store entire XML documents as single text objects in a conventional database, and query it using substring operators. This works well if we always want entire documents back, but it becomes clumsy if we want to pick out small subsets. For example, we might want all the speeches by Horatio in the play *Hamlet*
2. The XML file can be decomposed into separate elements, and each one stored as a separate row in a relational database. This is rather complex to achieve, but works quite well for some sorts of query. The problem is the overhead of reconstructing complex XML structures again (the text of *Hamlet* would be thousands of rows, off of which may have be carefully stitched together again.)
3. If the XML files are complex and static (a large linguistic corpus, perhaps) a fixed index to all the elements can be created. This allows for fast querying of certain types, and can be very efficient.
4. A dedicated XML database can be used. Typically, these are queried using the XPath language we saw in relation to XSLT. The big advantage is that there is no difference between retrieving the entire text of *Hamlet* and just a few lines from all Shakespeare plays, using a consistent query language. The problem is that XML databases are far less mature than big relational databases, and present problems when it comes to updating portions of the tree.

TEI documents are likely to be fairly static, but with complex nested structures, so the second two choices above are the best to consider.

Xaira is an example of an XML database which works by pre-processing the input and building indices. It provides an analytical view of a text, mainly for those interested in linguistic analysis, but it returns XML fragments which are styled under control of the user. shows a session with Xaira, selecting words which will be treated as variants of fag. shows a concordance of fag, while shows the same concordance, but with XML markup visible.



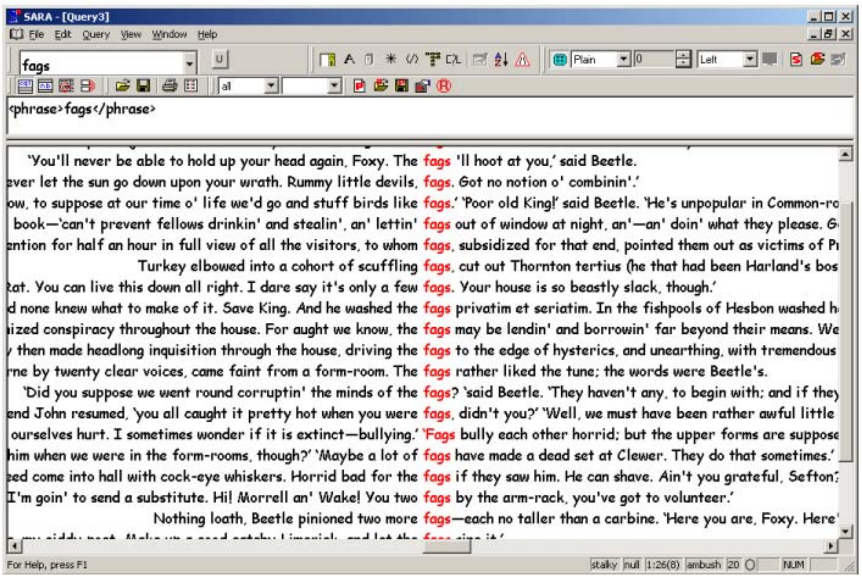
fag

| Word | Frequency |
|---|---|
| fag | 17 |
| fag's | 1 |
| fagdom | 1 |
| faggin | 1 |
| fagin | 1 |
| fags | 26 |

| Form | Frequency |
|---|---|

6 words

Figure 8. Xaira, selecting similar words

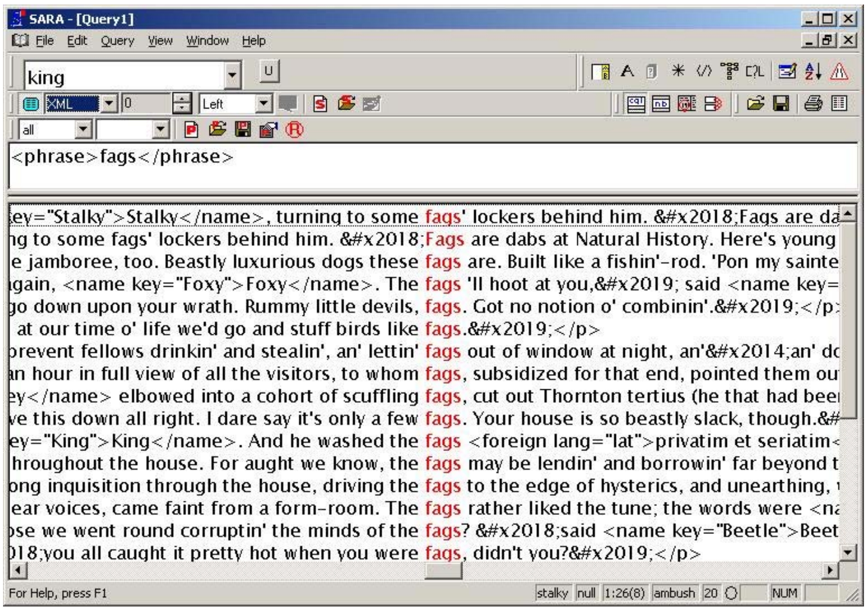


fags

`<phrase>fags</phrase>`

```
  'You'll never be able to hold up your head again, Foxy. The fags 'll hoot at you,' said Beetle.
ver let the sun go down upon your wrath. Rummy little devils, fags. Got no notion o' combinin'.'
w, to suppose at our time o' life we'd go and stuff birds like fags.' 'Poor old King!' said Beetle. 'He's unpopular in Common-r
 book—'can't prevent fellows drinkin' and stealin', an' lettin' fags out of window at night, an'—an' doin' what they please. 
ntion for half an hour in full view of all the visitors, to whom fags, subsidized for that end, pointed them out as victims of P
                    Turkey elbowed into a cohort of scuffling fags, cut out Thornton tertius (he that had been Harland's bos
at. You can live this down all right. I dare say it's only a few fags. Your house is so beastly slack, though.'
 none knew what to make of it. Save King. And he washed the fags privatim et seriatim. In the fishpools of Hesbon washed h
ized conspiracy throughout the house. For aught we know, the fags may be lendin' and borrowin' far beyond their means. We
 then made headlong inquisition through the house, driving the fags to the edge of hysterics, and unearthing, with tremendous
ne by twenty clear voices, came faint from a form-room. The fags rather liked the tune; the words were Beetle's.
  'Did you suppose we went round corruptin' the minds of the fags? 'said Beetle. 'They haven't any, to begin with; and if the
nd John resumed, 'you all caught it pretty hot when you were fags, didn't you?' 'Well, we must have been rather awful little
 ourselves hurt. I sometimes wonder if it is extinct—bullying.' 'Fags bully each other horrid; but the upper forms are suppos
him when we were in the form-rooms, though?' 'Maybe a lot of fags have made a dead set at Clewer. They do that sometimes.'
ed come into hall with cock-eye whiskers. Horrid bad for the fags if they saw him. He can shave. Ain't you grateful, Sefton
I'm goin' to send a substitute. Hi! Morrell an' Wake! You two fags by the arm-rack, you've got to volunteer.'
                    Nothing loath, Beetle pinioned two more fags—each no taller than a carbine. 'Here you are, Foxy. Here'
```

Figure 9. Xaira, word concordance



```
<phrase>fags</phrase>
```

```
ey="Stalky">Stalky</name>, turning to some fags' lockers behind him. &#x2018;Fags are da
ng to some fags' lockers behind him. &#x2018;Fags are dabs at Natural History. Here's young
e jamboree, too. Beastly luxurious dogs these fags are. Built like a fishin'-rod. 'Pon my sainte
gain, <name key="Foxy">Foxy</name>. The fags 'll hoot at you,&#x2019; said <name key=
go down upon your wrath. Rummy little devils, fags. Got no notion o' combinin'.&#x2019;</p
 at our time o' life we'd go and stuff birds like fags.&#x2019;</p>
prevent fellows drinkin' and stealin', an' lettin' fags out of window at night, an'&#x2014;an' d
an hour in full view of all the visitors, to whom fags, subsidized for that end, pointed them ou
ey</name> elbowed into a cohort of scuffling fags, cut out Thornton tertius (he that had bee
ve this down all right. I dare say it's only a few fags. Your house is so beastly slack, though.&#
ey="King">King</name>. And he washed the fags <foreign lang="lat">privatim et seriatim<
hroughout the house. For aught we know, the fags may be lendin' and borrowin' far beyond t
ong inquisition through the house, driving the fags to the edge of hysterics, and unearthing,
ear voices, came faint from a form-room. The fags rather liked the tune; the words were <na
ose we went round corruptin' the minds of the fags? &#x2018;said <name key="Beetle">Beet
018;you all caught it pretty hot when you were fags, didn't you?&#x2019;</p>
```

Figure 10. Xaira, word concordance showing XML markup

eXist (the Apache Project's XIndice offers similar features) is an example of a dedicated XML database engine. It has no interface of its own, but can talk to other programs or services using HTTP, XML-RPC, SOAP and WebDAV. This makes it easy to deploy inside dynamic web page systems, such Apache's Cocoon. The only job of the database is to return a well-formed XML tree which other applications can process. It is queried using XQuery or XPath expressions, with support for some small extensions to permit more reasonable substring searches within element content. Examples of XPath queries follow, where the root is expressed with document and collection functions to identify a single document or named group of documents:

**document('test.xml')//div**

All `<div>` elements in document test.xml collection('xx')//div All `<div>` elements in all documents in the *demo* collection

**collection('xx')//div[.//p[contains(.,'Hello')]]**

All `<div>` elements which contain `<p>` elements containing the phrase 'Hello'

**collection('xx')//div[.//xref[@url='www.lexmark.co.uk']]**

a `<div>` with an `<xref>` with a **url** attribute pointing to Lexmark

**collection('xx')//list[count(item)>9]**

Lists with more than 3 items

**collection('xx')//div[starts-with(head,'Introduction')]**

`<div>` elements with a title starting *Introduction*

**collection('xx')//q[string-length(.)>100]**

quotations longer than 100 characters

**collection('xx')//person[.//surname[.='Keats' or .='Shelley']]**

`<person>` elements for Keats or Shelley

All these queries use standard syntax. eXist also supports some extra functions and operators, which is uses to give efficient access to a full-text index which it maintains in addition to the structural database:

**collection('xx')//p[match(.,'[A-Z]IED\b')]**

paragraphs with words ending in 'IED'

**collection('cem')//ab[near(.,'BOY YEARS',4)]**
<ab> elements with the words BOY and YEARS in a proximity of four words

**collection('cem')//ab[.&='BORN IN ROME']**
<ab> elements with the phrase 'BORN IN ROME'

**collection('cem')//ab[.|='GOD LOVE']**
<ab> elements with the words 'GOD' or 'LOVE'

While these extensions are not necessary for most applications, they are likely to be relevant to TEI users.

We can see how eXist presents its results by looking at what comes back from the query against *Stalky & Co:* of

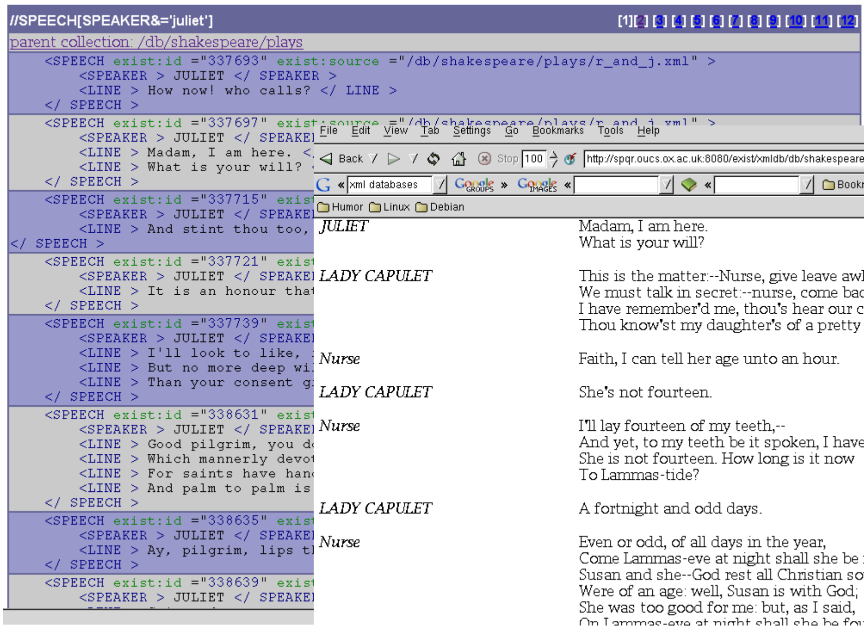
```
//foreign[@lang='fr']
```

ie all the words marked as `<foreign>` with the value `fr` for the **lang** attribute:

```
<?xml version="1.0" encoding="utf-8"?>
<exist:result
   xmlns:exist="http://exist.sourceforge.net/NS/exist"
         hitCount="19" queryTime="41">
   <foreign lang="fr"
     exist:id="12751728" exist:source="/db/kipling/stalky.xml">
             aujourd'hui</foreign>
   <foreign lang="fr"
     exist:id="12751730" exist:source="/db/kipling/stalky.xml">
             Parceque je</foreign>
   <foreign lang="fr"
     exist:id="12752720" exist:source="/db/kipling/stalky.xml">
             langue de guerre</foreign>
   <foreign lang="fr"
     exist:id="12793848" exist:source="/db/kipling/stalky.xml">
             Twiggez-vous</foreign>
   <foreign lang="fr"
     exist:id="12793938" exist:source="/db/kipling/stalky.xml">
             Nous twiggons</foreign>
   <foreign lang="fr"
     exist:id="12799254" exist:source="/db/kipling/stalky.xml">
             a la</foreign>
   <foreign lang="fr"
     exist:id="12804738" exist:source="/db/kipling/stalky.xml">
             Moi</foreign>
   <foreign lang="fr"
     exist:id="12804740" exist:source="/db/kipling/stalky.xml">
            Je</foreign>
   <foreign lang="fr"
     exist:id="12847398" exist:source="/db/kipling/stalky.xml">
            Par si je le connai</foreign>
   <foreign lang="fr"
     exist:id="12866298" exist:source="/db/kipling/stalky.xml">
            Je vais gloater</foreign>
```

```
</exist:result>
```

in which each result fragment is given extra attributes in the eXist namespace identifying the source document and an internal identifier. These could be easily transformed using an XSLT stylesheet to present them in any desired way. shows two aspects of a web interface to Shakespeare plays using eXist; the lower view shows the XML results of searching for speeches by Juliet; the upper view shows a complete retrieval of *Romeo and Juliet* formatted using XSLT.


//SPEECH[SPEAKER&='juliet']
[1][2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12]
parent collection: /db/shakespeare/plays
File Edit View Tab Settings Go Bookmarks Tools Help
Back
Stop
100
http://spqr.oucs.ox.ac.uk:8080/exist/xmldb/db/shakespeare
xml databases
Google GROUPS
Google IMAGES
Bookr
Humor
Linux
Debian

```
    <SPEECH exist:id ="337693" exist:source ="/db/shakespeare/plays/r_and_j.xml" >
        <SPEAKER > JULIET </ SPEAKER >
        <LINE > How now! who calls? </ LINE >
    </ SPEECH >
    <SPEECH exist:id ="337697" exist:source ="/db/shakespeare/plays/r_and_j.xml" >
        <SPEAKER > JULIET </ SPEAKE
        <LINE > Madam, I am here. <
        <LINE > What is your will? <
    </ SPEECH >
    <SPEECH exist:id ="337715" exist
        <SPEAKER > JULIET </ SPEAKE
        <LINE > And stint thou too,
</ SPEECH >
    <SPEECH exist:id ="337721" exist
        <SPEAKER > JULIET </ SPEAKE
        <LINE > It is an honour that
    </ SPEECH >
    <SPEECH exist:id ="337739" exist
        <SPEAKER > JULIET </ SPEAKE
        <LINE > I'll look to like,
        <LINE > But no more deep wi
        <LINE > Than your consent g
    </ SPEECH >
    <SPEECH exist:id ="338631" exist
        <SPEAKER > JULIET </ SPEAKE
        <LINE > Good pilgrim, you d
        <LINE > Which mannerly devot
        <LINE > For saints have han
        <LINE > And palm to palm is
    </ SPEECH >
    <SPEECH exist:id ="338635" exist
        <SPEAKER > JULIET </ SPEAKE
        <LINE > Ay, pilgrim, lips t
    </ SPEECH >
    <SPEECH exist:id ="338639" exist
        <SPEAKER > JULIET </ SPEAKE
```

*JULIET*
Madam, I am here.
What is your will?

*LADY CAPULET*
This is the matter:--Nurse, give leave awl
We must talk in secret:--nurse, come bac
I have remember'd me, thou's hear our c
Thou know'st my daughter's of a pretty

*Nurse*
Faith, I can tell her age unto an hour.

*LADY CAPULET*
She's not fourteen.

*Nurse*
I'll lay fourteen of my teeth,--
And yet, to my teeth be it spoken, I have
She is not fourteen. How long is it now
To Lammas-tide?

*LADY CAPULET*
A fortnight and odd days.

*Nurse*
Even or odd, of all days in the year,
Come Lammas-eve at night shall she be
Susan and she--God rest all Christian so
Were of an age: well, Susan is with God;
She was too good for me: but, as I said,

Figure 11. Web-based XML search using eXist

Most XML databases suffer from the problem of effective *update* of documents; adding in elements to a complex tree structure requires careful checking of constraints, and rebuilding of indices. The XUpdate proposal attempts to specify standard ways of managing this. Another problem are the extensions to XPath needed for fully effective database searching; this is covered by the W3C XQuery and this is not yet fully accepted or dominant. It remains sensible to use text files as archival storage, and use the XML databases for static presentation.

## Conclusions

The publisher of a TEI XML document has a wide variety of methods to access the data locked within. While any XML-aware software can do the job, we are also able to take advantage of a wide range of open standards, with commercial and open source implementations. For transformation, typesetting, retrieval, and analysis, it is increasingly easy to use off-the-shelf packages.

---

## Notes

[1] For the curious reader, example texts are mostly drawn from Rudyard Kipling's 1899 collection of stories, *Stalky & Co.*

---

Follow @TEIConsortium | ATOM | HTML View | XML View | Printable Version |

UDM 4 | sourceforge | POWERED BY TEI | Contact/Feedback