Preserving a Listserv Archive in TEI: The Case of TEI-L

Syd Bauman and Elisa Beshero-Bondar

ABSTRACT

"Can the archives of an email list be stored in TEI?" This paper addresses this question with special attention to the challenges of data retrieval from the TEI-L Listserv, and the mapping of email contents and metadata to XML and TEI. Our experiments with TEI data modeling involve adapting encoding models from the new "Computer Mediated Communication" chapter, as well as correspondence and personography in the TEI Guidelines. We experimented with very different approaches in retrieving and mapping Listserv data, and we faced challenges that reflect changing technologies in character encoding and email transmission, as well as conceptual difficulties in reliably tracing individual contributions over years and decades. Ultimately we believe that

yes, it is possible (although difficult) to store the archives of an email list in TEI, that there are some problems with establishing a satisfactory encoding, and the methods one chooses will likely depend on the scope of a Listserv archival project.

INDEX

Keywords: TEI, Listserv, character, email, computer-mediated communication

1. Introduction

1 One of the priorities of digital humanities is cultural memory. It is a common goal to preserve the results of humanistic endeavors in a format suitable for scholarly examination and inquiry that may be easy to access many years in the future. From the last quarter of the twentieth century to the present, one venue for communication among humans has been email, and in particular the email discussion list. Digital humanists will likely want to preserve the data of email discussion lists in a way that is likely to survive the ups and downs of the software market and is amenable to scholarly processing.

2 It may be surprising that the Text Encoding Initiative does not yet provide a dedicated mechanism for encoding email. TEI does provide mechanisms for encoding *physical* letters and chains of correspondence. Certainly there are quite a few similarities between letters and email. Like a letter, a piece of email is written by an author and sent to a recipient at a particular point in time. But physical letters do not have such a sharp demarcation between data & metadata, rarely contain multiple copies of the same content in different formats (typically HTML and plain text), and never suffer character encoding ambiguities. Conversely, email does not have seals or stamps or quite the same kinds of inconsistencies we expect to see in any set of manuscripts.

3 As of 2024, TEI now provides a mechanism for encoding computer-mediated communication (CMC). It is intended, however, for systems like Twitter or Tik-Tok rather than for email, and it is not yet known how well its encoding system could be applied to email. Thus the impetus for this project: can TEI, with its new CMC chapter, be used to encode an email discussion list?

4 We decided the TEI-L discussion list would make an excellent test case. It is a large resource that spans decades from mailing list software (Listserv) with which we are somewhat familiar. Furthermore, because we had recently moved the list from Brown University to Penn State University, accessing the raw archives from its beginning to the day the list was moved would be trivial. But most of all, the list is obviously of significant interest to the TEI community. For us, its contents comprise a heritage "dig site" of resources and perspectives from the early days of text encoding and digital humanities. The first post was on Monday, 08 January 1990, so the list was started months before the first edition of P1, and years before the advent of XML.

5 Our initial thought was that converting the data to XML would be easy, and that most of our attention would be directed towards the more interesting problem of deciding exactly how to encode it in TEI. However, it turns out that both parts were problematic. It was very difficult to convert the data into XML, and in several cases there is no TEI encoding that is really satisfactory.

6 We took two approaches to encoding the TEI-L Listserv data in TEI:

- a craft encoding of a "core sample" of the list, holding a short range of months or years worth of posts.
- an automated conversion of mass data to a basic TEI representation for archiving and potential later enhancement

2. Conversion from Source to XML

2.1 Scrape from the Web (attempted) Approach

7 In an effort to retrieve some "core samples" to support the craft encoding approach and retrieve all the messages in a selected range of months or years, we attempted to "scrape" these from the web archive of the TEI-L, which can be accessed on the web in its current home at Penn State University at https://lists.psu.edu/cgi-bin/wa?A0=TEI-L. By adjusting this public web URL, one could retrieve a "core sample" from the TEI-L by entering a date range, such that if one were seeking to retrieve all of the messages between January and March of 1990, the URL that serves these is: https://lists.psu.edu/cgi-bin/wa?A1=199001-199003&L=TEI-L. A significant limitation of this approach, at least in Listserv 17.0 currently used by Penn State, is that the website access to

the archives limits the number of results on a screen so that when large quantities of messages (say over 100) are retrieved, some results can only be retrieved on click of a “next” button. In a programmatic scraping context, this posed some obstacles:

- With no clear way to request all the results in a single output, we would have to programmatically anticipate how to locate the last entry and retrieve those that follow it.
- Outputs are alphabetically sorted by the first portion of the sender’s name (often a first name, but sometimes a surname), when one would expect them to be chronologically sequenced. (This is not a serious problem because we could control sorting the output in the scraping process).

We expect it is possible to overcome these difficulties, but due to time constraints in launching this project for the 2024 TEI Conference, we limited ourselves to small samples that would be retrievable on one screen to simplify our processing requirements. We tinkered with the Scrapy Python library and lxml e-tree for parsing XPath, after a false start in attempting to adapt an excellent scraper designed by GitHub user gaalcaras for retrieving messages from Hypermail, which produces a much simpler web archive than Listserv.

8 Our limited scraping exercise helped us to learn about the Listserv web archive retrieval system. What was most interesting to us about this method was the discovery that Listserv provides each message a distinct identifier for retrieving it for web view. If we could retrieve the distinct identifiers from URLs and CGI paths, they could be a reliable way to pull message metadata and content. On the archive results page for a search parameter, we found regular structures as seen in example 1 (retrieved from a search for all of the 1990 messages from January to December). All message data is provided in an HTML table, with each row sharing information about each message, including a bit of the message text content to be shown on mouseover, sender name and email, a date and time stamp (though it is unclear whether this is for circulation by Listserv or from the original email heading), and finally an HTML anchor pointing to a CGI path to retrieve the message with a distinct identifying string, as shown here:

Example 1. Scraped Message Metadata.

```
<tr>
```

```
   <td class="emphasizedgroup row-l" scope="row"><div class="archive
forcewrap"><span onmouseover="showDesc('Dear Colleague:&lt;br&gt;&lt;br&gt;From
27 May to 29 June 1990 I am out of the country and so gone from&lt;br&gt;this
address, chiefly to attend the ALLC/ICCH conference in Siegen,&lt;br&gt;Germany.
Mail will be kept faithfully for me here, but if I do not&lt;br&gt;reply to your
message in early July, please resend your message then. [...]','Re: where is SGML
being used','Fri, 29 Jun 90 13:19:30 EDT')" onmouseout="hideDesc()"><a href="/cgi-
bin/wa?A2=TEI-L;5bb4caca.9006&amp;S=">Re: where is SGML being used</a></span></
div></td>
   <td class="emphasizedgroup row-l"><div class="archive forcewrap">Willard
McCarty &lt;MCCARTY@VM.EPAS.UTORONTO.CA&gt;</div></td>
   <td class="emphasizedgroup nowrap row-l"><div class="archive forcewrap">Fri, 29
Jun 90 13:19:30 EDT</div></td>
   <td class="emphasizedgroup right row-l"><a href="/cgi-bin/wa?MD=TEI-
L&amp;i=9006&amp;I=1741011559&amp;s=204&amp;l=16&amp;X=O96EB1622E3EEE9B193&amp;Y=eeb4%40psu.edu&amp;URL
%3DTEI-L%26s%3D0%26X%3DO96EB1622E3EEE9B193%26Y%3Deeb4%2540psu.edu&amp;M_S=Re
%3A+where+is+SGML+being+used&amp;M_F=Willard+McCarty+%26lt
%3BMCCARTY%40VM.EPAS.UTORONTO.CA%26gt%3B&amp;M_D=Fri%2C+29+Jun
+90+13%3A19%3A30+EDT&amp;M_Z=16+Lines&amp;A1DATE=&amp;A2URL=https%3A
%2F%2Flists.psu.edu%2Fcgi-bin%2Fwa%3FA2%3Dindex%26L%3DTEI-L%26O%3DA%26X
%3DO96EB1622E3EEE9B193%26Y%3Deeb4%2540psu.edu%26s%3D0%26P%3D17156">
     <img src="/archives/images/default_archive_delete_22x22.png" alt="Delete
Message" title="Delete Message"/></a></td>
  </tr>
```

The cgi-bin locational information is certainly complicated, but appears to be parsable. The distinct identifier for a message can be found in the fourth and last `<td>` element by retrieving its child `<a>` and selecting the portion of its `@href` attribute value that begins with `&X=`, its X parameter. In this example Willard McCarty's message happens to be the last on the first page of results, and it is used again by Listserv within the value for its "Next" button to retrieve the results for the second screen. One can find it as the "X" parameter once again:

Example 2. `Next` **Button Reproducing Message ID.**

```
  <input type="button" value="Next Page &#62;"
onclick="clearSessionStorage(); goTo('/cgi-bin/wa?A1=199001-199012&amp;L=TEI-
L&amp;s=250&amp;X=O96EB1622E3EEE9B193&amp;Y=eeb4%40psu.edu')"/>
```

Our Python scraping experiment could follow these paths to extract URLs for messages, but we decided not to continue the experiment to try to extract Listserv's web representation of the message data, because that data posed their own challenges for parsing "spaghetti code" and we could proceed more efficiently with a simpler resource for the email message data. Because Listserv provides much cleaner and simpler monthly log text files on request that are far more tractable, we decided to work exclusively with the log files for experiments with converting to Listserv email data to TEI. You can explore the log files for the year 1990 in our GitHub repository and see how much simpler they are to work with as a basis for conversion. What is missing in those logs, however, is Listserv's distinct identifier for each message.[1] So for a future experiment, we may pull the message identifiers from Listserv's web archive files in order to key the logs to the web retrieval system.

2.2 Mass Conversion Approach

9 Our goal for the mass conversion approach was to programmatically convert all of TEI-L from its inception up to Monday, 29 April 2024 into some form of simple, basic TEI using XSLT. That end date was chosen because it was the day the mailing list was moved from Brown University's Listserv to Penn State University's Listserv. This fact meant we could obtain the entire dataset as text files directly from the Brown listmaster without needing to worry about merging in data from PSU or about posts made just after we obtained the data. Peter DiCamillo, the Brown listmaster, sent us 412 separate files, one for each month, i.e., tei-l.log9001 through tei-l.log2404. These data are also available by GET requests from the API service of LISTS.PSU.EDU. For details see the `<samplingDecl>` in our example in section 3.1.

10 The first step was to rename the log files to use 4-digit (as opposed to 2-digit) years so that they sorted into the correct order.[2] This renaming allowed the logs to be easily examined in a reasonable order and, more importantly, allowed them to be combined into a single file in the correct order using one simple command: `cat tei-l.log* > TEI-L.txt`.[3]

2.2.1 Delimiters

11 Each log file provided by Listserv contains one or more posts to the mailing list. For our dataset, the range was 1–420 posts per monthly log file, although in several cases there is a month that has no postings, for which Listserv did not generate a .log file at all.

12 Thus an early task was to figure out how posts are separated in a log file. Upon inspecting the dataset, it was quickly apparent that Listserv uses a string of 73 equal signs (U+003D, "=") in a row as a start-post indicator, because there is such a delimiter before the first and every subsequent post in a file, but not after the last.

13 But, of course, there is nothing to stop a mailing list user from including a string of 73 equal signs inside her post. We first checked (out of curiosity) to see if any strings of 73 equal signs were obviously *not* post delimiters, because they did not occupy the entire record. A search for the delimiter string (using just `egrep -c`) revealed that it occurs 31,031 times in the dataset; a search for the delimiter string anchored to the beginning and end of the line revealed that it occurs alone on a line 30,998 times. So there are 33 cases of 73 equal signs in a row that do not occupy the entire record, and thus are not start-of-post indicators. (It is likely they exist as cases of posts being copied-and-pasted into other posts.)

14 But are there any cases of 73 equal signs in a row that do occupy the entire line, but are *not* intended to be a start-post delimiter? We know that every email message (and thus every post to an email list) starts with a sequence of metadata records. Email metadata records are in a very specific format: the field name, which must be composed of one or more of the printable 7-bit ASCII characters except colon, followed by a colon, followed by a field body. On a quick examination of the dataset, it appears that the first field in every case is the `Date:` field. So we searched for occurrences of 73 equal signs that start at the beginning of a line and are followed immediately by a newline followed by something *other* than an uppercase "D."[4] There are only three such cases, so we just examined each of them.

15 The first is simply a case of cut-and-paste: on Thursday, 10 September 1992 Wendy Plotkin, who was at the time a PhD candidate in History at UIC and who served as the TEI's research assistant (1990–1999), posted email that contained a copy of email from Professor Robert Jones of the University of Illinois Urbana-Champaign to a CETH mailing list, including a line of 73 equal signs which was used to separate the part she wrote from the part Professor Jones had sent.

16 The other two cases are both of two lines of 73 equal signs in a row, with nothing between. Whether these indicate that the preceding post ended with the same set of characters that is used as the start-post delimiter, or that there is a missing post, we do not know.

17 So the result of this analysis was that a line of 73 equal signs in a row could be used reliably to indicate the start of a new post, with one exception, which we simply changed by hand to a line of 72 equal signs. (This is visually the same for a human reader, but distinctly different to a computer looking for 73 of them.) We also had to be prepared for the possibility of two such lines in a row.

2.2.2 Illegal Characters (for XML)

18 Early email systems used ASCII (7-bit) characters only. Thus there were 128 available code points (0–2^7-1, or 0–127). Of those 128 available code points, only 99 of them are legal XML characters. The remaining 29 characters (for which see Appendix 1) are all non-printing control characters, intended for controlling the behavior of a peripheral device such as a card punch or printer, or (in a few cases) whitespace.[5] While neither of us is an expert on early email systems, we suspect that the designers never intended these characters to be used in email, but did not feel it necessary to do anything to prevent them from being used, as the average user of a computer had no way of typing them, and thus would find it very difficult to get them into an email.[6]

19 Nonetheless, of the 29 characters that are not legal in XML, 17 of them occur in the TEI-L archives:

- 64 occurrences in 34 posts before 2000 (which, on reflection, was not too surprising)
- 38 occurrences in 11 posts after 2008 (which we did not expect at all)

These characters were generally used not to represent the control characters to which they are mapped in the ASCII character set, but rather to represent various accented characters that were not part of the original (7-bit) ASCII set. Often these characters occurred within text that was likely copied-and-pasted from another source (e.g., from a conference announcement). The last use of a character that is illegal in XML was on 2016-03-13.

20 An input document that contains illegal characters *cannot* be processed by XSLT, whether it is in an XML file or a plain text file. And, of course, such characters cannot occur in an XML file whether that file is generated by XSLT, XQuery, Python, C, or by hand in a text editor. In some of those

languages, or using a text editor, you could *put* them into a file that has a ".xml" extension, but the file would not meet the definition of an XML file, and thus could not be processed by conforming XML software (other than to emit an error message).

21 These control characters are being used incorrectly, and are not allowed in XML, but they are not mal-formed or illegal Unicode characters—they are perfectly legitimate Unicode characters that are not allowed in XML. Thus oXygen's otherwise useful Encoding errors handling feature is not helpful here, as it relates to oXygen's processing of mal-formed Unicode characters. Likewise a utility like iconv, which converts text from one character encoding to another, is of little to no use because these control characters are problematic no matter what character encoding is used to represent them.

22 We immediately envisioned several possible solutions, each of which would have to be carried out on the plain text input files prior to processing into XML.

1. Delete the problematic characters, leaving users to figure out what was missing and where. This lossy methodology seemed like a bad idea.
2. Replace each of the 102 occurrences of the 17 problematic characters with a single "warning" character. This would allow users to immediately know where there was a problem, but would leave solving it (i.e., figuring out which character belonged there) to the user. It requires, of course, that we use a character that does not occur in the data. (One obvious choice, for example, would be "⚠" (U+26A0), but it already occurs three times in the input dataset.) This is obviously better than just deleting the characters, but still puts effort on the reader.
3. Replace the problematic characters with 17 different "warning" characters. We believe some users would find decoding this a lot easier, but others would find it more difficult (because it did not seem to be the case that there was always a 1:1 relationship between the control character used and the desired character). This also would mean that we would have to find 17 characters that would be clear to the user, but do not occur in the input dataset.

4. Replace the problematic characters with 17 different "warning" *strings*. After all, XML does have a replacement mechanism. So if we were to replace each of the six occurrences of, say, U+001A with "⊙SUB⊙" in the input document, we could either leave them that way for the user, or convert them to "&SUB;" and provide an appropriate declaration, for example to a PUA character: `<!ENTITY SUB "#">`. This would likely be easier for readers than using a single character for each of the 17 problematic characters, but as with that solution readers would be confused when there was not a 1:1 mapping from the strings we provide and the likely intended characters.
5. For each occurrence of a problematic character, try to figure out what character the author intended, and replace it with that (proper UTF-8) character. While this involves far more work than any of the above, we feel this option results in documents that are easier for users to process and read. We considered four variations on this theme, based on two binary features:

    - in each case, try contacting the author to ask what was the intended character, or not;
    - in each case, just replace the problematic character with our guess at the correct desired character, or replace it with a `<choice>` encoding which records both the original character (e.g., as `<orig><g type="CONTROL" subtype="FS"/></orig>`) and our regularization thereof (e.g., as `<reg>£</reg>`).

    While we think that contacting the original authors is, in some cases, beneficial, and believe that the `<choice>` encoding is by far the best way of representing the problematic characters, for our small experiment we chose the easier route for each of these features.[7]

23 After replacement of those characters that are illegal in XML by those characters we thought they should be, the input dataset was easily read by XSLT. Converting that to useful, readable XML presented some challenges, though.

2.2.3 Java, and Memory, and Streams, oh my!

24 We quickly discovered that processing the entire ~151 MiB of data at once requires a lot of RAM, more than Saxon gets from the Java virtual machine by default. There are three obvious solutions to this problem.

1. Chop up the data: Rather than reading in a single file of ~151 MiB of data, use perhaps 10 files of ~15 MiB each (or vice-versa); or even use the original 412 files (the largest of which is ~4 MiB) generated by Listserv.
2. Use streaming XSLT: The whole point of the "streamability" feature of XSLT 3.0 is to be able to process large, perhaps infinitely large, datasets in memory.
3. Give the Java virtual machine more heap space.

25 Using smaller chunks of input seems like a very good approach in the general case. We did not follow that route simply for the sake of expediency: we had already put in hours of effort on the combined ~151 MiB file of data (searching for and changing characters illegal for XML), and simply did not want to take the time to artificially divide the data into multiple files and then change our XSLT program so it would iterate over them. That said, this would be a reasonable approach were it not for the time constraints on our unfunded pilot project.

26 We *think*, but are not at all convinced, that the use of streaming XSLT would solve this problem. Although the data we are reading in is not XML, it is internally converted to XML and then processed in several stages. It is likely that if these stages were streamable the memory use would be dramatically decreased. However, neither of us is particularly well versed in writing streamable XSLT, and, perhaps more importantly, do not (yet?) consider it a viable solution for digital humanists, as the only XSLT processors in the world that have this feature are payware.

27 So while we admit it may be rude to require a user of our program to do so, we simply increased Java heap space to 5 GiB when running this transformation.

2.2.4 Date Inconsistencies

28 We think it is a given that having a machine-actionable date & timestamp for each posting is an important feature of a preservation format for an archive of email messages. In TEI such a timestamp is typically stored on a `@when` attribute in the W3C format (itself a subset of ISO 8601:2000 2nd edition format).

29 Early on in the development of our XSLT program we were quite worried about how to go about parsing the values of the `Date:` header fields. While the format of this field is reasonably strictly defined by rfc5322, that standard was not published until 2008, 18 years after the TEI-L mailing list was started. We expected the given dates to be in dozens of different, potentially incompatible, formats.

30 After some analysis we were pleasantly surprised to find that only 21 different formats were represented in this dataset. (The frequency of each format is listed in Appendix 2.) Interestingly, most of the variation was in the time zone. Discounting that variation, there were only four different formats. The variation in time zone indication, however, was at times ambiguous. For example, the time zone "BST" appears in the `Date:` field of over twenty posts. Given that the TEI was partially based in Great Britain, it is very likely that "BST" means "British Summer Time" (+01:00). But it is also used for "Brazilian Summer Time" (-02:00), "Bangladesh Standard Time" (+06:00), and "Bougainville Standard Time" (+11:00).

31 In the end, we were able to parse all 38,787 `Date:` fields using the regular expression capability of XSLT 3.0, plus a pair of maps to convert month names to 2-digit numbers and the various time zones we had found in our analysis to the correct format. Some of those time zone mappings were guesses based on the top level domain of the email addresses that had sent mail with an ambiguous time zone.[8]

2.2.5 Unsolved Conversion Complexities: MIME

32 MIME stands for *Multipurpose Internet Mail Extensions*, a set of extensions to email that allows for

- representation of more than just 7-bit US-ASCII characters in header field values
- representation of more than just 8-bit US-ASCII characters in mail bodies
- representation of content using systems other than just plain text, possibly as attachments (e.g., text/html, application/msword)

These capabilities and the corresponding questions raised are discussed in the following subsections. But for us MIME may as well stand for *Major Incoherent Mess in Email*, because other than being able to parse out the various MIME parts of an email message body into separate constructs, we were not able to process MIME extensions at all in this limited pilot project.

33 When MIME is used,[9] (1) a `MIME-Version:` field is present (its value is always `"1.0"`); (2) the mail body is divided into 1 or more *parts*, each of which has some associated metadata of its own; and (3) if there is more than 1 part, the `Content-Type:` field will start with `"multipart/"` and have a `boundary=` parameter, whose value can be used by mail clients (or intrepid digital humanists) to parse out the different parts. Furthermore, each part may itself be multipart, in which case it also has a `Content-Type:` field that starts with `"multipart/"` and a `boundary=` parameter. Thus the parts are arranged as a standard hierarchical tree, although in practice the tree often has only leaf nodes (i.e., parts that are not themselves multipart).

2.2.5.1 ASCII Work-around: Encoded-word Syntax

34 As mentioned above, early email used only 7-bit ASCII characters for the entire message. Later updates to the specifications allowed for the body of the email message to use a different character set, but the email headers (that is, the lines of metadata at the top) are still to this day limited to 7-bit ASCII. Most of the world's printable characters, of course, are not available in that set of 96 printable characters. To allow for characters outside the 7-bit ASCII set to be used in email headers (e.g., in the name of the sender or recipient of an email), an escape system called *encoded-word syntax* may be employed. (See rfc2047.)

35 Using this system, a word that contains a character outside of 7-bit ASCII is encoded as `=?charset?encoding?encoded text?=`, where *charset* is the character encoding in which the encoded text should be looked up, *encoding* is either "Q" (for a quoted-printable style escape mechanism) or "B" (for a base64 style escape mechanism), and *encoded text* is the string encoded using the escape mechanism indicated. The character sets employed in this dataset were as follows, ignoring case distinctions. (For example, UTF-8 was often spelled "utf-8.")

character set (case regularized)	count	%
UTF-8	2300	55.6%
ISO-8859-1	1043	25.2%
ISO-8859-2	383	09.3%
Windows-1252	236	05.7%

ISO-8859-15	132	03.2%
ISO-8859-5	16	00.4%
KOI8-R	5	00.1%
big5	4	00.1%
ISO-2022-jp	3	< 1 ‰
Windows-1250	3	< 1 ‰
ISO-8859-4	2	< 1 ‰
Windows-1256	2	< 1 ‰
Windows-1251	2	< 1 ‰
Windows-1257	1	< 1 ‰
X-UNKNOWN	1	< 1 ‰
gb2312	1	< 1 ‰
total	4134	

Dramatically more word-encoded passages in this dataset employ “Q” encoding over the “B” method: 3,397 (82.2%) and 737 (17.8%), respectively.

36 We recognized and considered the problem of passages represented with word-encoded syntax. Our instinct is that most digital humanists with an interest in this data would be better served by the results of our conversion process if we decoded these strings (for example, converted “=?UTF-8?Q?Piotr_Ba=C5=84ski?=” to “Piotr Bański”),[10] and that those performing digital archaeology about how various characters were represented by various historical email systems would just as soon use the source data rather than the TEI-encoded output of our process. We are not convinced of this position, though, and did not have anywhere near enough time in this pilot project to attempt to write a word-encoded syntax decoder in XSLT.

2.2.5.2 Separating MIME Parts

37 Email that uses MIME is divided into one or more *parts*, each of which contains a singe type of data. The most common types in our dataset are text/plain and text/html; much less frequent are text/enriched, text/x-vcard, text/xml, and application/*.

38 It would not be unreasonable to put all the different MIME parts of a post to the Listserv as raw data into a single TEI `<div>` or `<post>` element. It would certainly be easier than trying to parse the parts into separate TEI elements. But this *caveat emptor* approach, while possibly useful for preservation of the data, would yield data that was not at all amenable to scholarly processing.

39 It would also be reasonable to ignore all MIME parts other than text/plain — after all, this is the *Text* Encoding Initiative version of the data we are creating. (In fact, this is what we did for the "scrape + craft encoding" approach described in section 2.1, except there all parts were ignored except the one the Listserv web interface chose to display.) But it would also be quite reasonable, and quite possibly helpful to future scholars and interested parties, to retain the information in other parts of postings to the mailing list. But doing so presents interesting practical and theoretical challenges.

40 The practical challenges can be divided into two broad categories: parsing the content of the email into its separate MIME parts, and processing the content of each MIME part according to its type. We were able to partially solve the first challenge within the scope of this small pilot project. Our XSLT will recognize MIME multipart content, and separate out each of the highest-level parts into their own constructs. However, it does not yet handle multipart-inside-multipart MIME content.

41 The second practical challenge, processing the content of a MIME part according to its type, is reasonably easy to contemplate for XSLT when the content type is text/* or application/xml. But in almost all other cases conversion to something tractable by XSLT is at best difficult.

42 The theoretical challenges strike us as very interesting, and they boil down very simply to a question of what to do with data other than plain text. Some datatypes easily lend themselves to TEI encoding; counter-intuitively, these are the datatypes for which there is really no reasonable XML representation. Imagine, for example, that one of the MIME parts is of type image/jpeg or audio/ogg. (None of them are, as the rules of this particular mailing list disallowed most binary attachments to help prevent the spread of malware.) The only real possibilities are to ignore it, to put the binary data in a separate .jpg or .ogg file and point to it, or to encode the data in a `<binaryObject>` element.

43 But what of data typed as some form of text other than text/plain? In many cases, the MIME part is another representation of the same content as a text/plain MIME part. (This allows a client that knows how to display, say, text/html to do so, but a client that does not have that capability to display the text/plain content.) In which case, should it be included in the TEI version of the archive at all? It would certainly be easier to simply ignore the HTML version. But the author of the email may well have used an editor that allowed certain styling (like italics, change in font size, change in font color, etc.), and that styling is likely available in the HTML but not the plain text. Moreover, that styling may be of interest to future scholars. But what TEI constructs should the HTML be converted into? HTML to TEI conversion is not at all a simple task. (For starters, HTML permits headings in the middle of divisions; and the table models, while similar, are not the same.) One possibility, of course, is to move the HTML into a separate file, and point to it from the TEI. (This also has the advantage of avoiding the problem of what to do with ill-formed HTML.)

2.2.5.3 Content Encoding

2.2.5.3.1 Why Content Encoding is Required

44 Email servers, etc., use at most 8 bits per character. (And, as pointed out in section 2.2.5.1, only 7 bits per character in the header.) It is possible to represent 256 characters in 8 bits. Thus any message body that contains any of the other 154,807 characters needs to be down-translated into only those 256 characters. MIME permits two different methods for performing this down-translation: *quoted-printable* and *base64*. For the actual definitions of these formats see rfc2045 sections 6.7 & 6.8, respectively.

45 Note that using the specified down-translation method to recover the correct character codepoints is not sufficient to decode the message body. The client must also know which character set to use to map each codepoint to a character. For example, in the ISO 8859-2 character set the codepoint 241 (hexadecimal F1) represents “ń”; in UTF-8 it represents “ñ.” Which character set to use is specified on the Content-Type header field. For example, `Content-Type: text/plain; charset=ISO-8859-1` specifies the ISO 8859-1 character set.

2.2.5.3.2 Quoted-printable

46 The quoted-printable method is, at its core, quite simple. Any sequence of 8 bits (an octet, the base 2 version of a number from 0 to 127, and on most systems a byte), except for the octets reserved for signaling end-of-line, is represented instead by an equal sign (U+3D, “=”) followed by two (upper

case) hexadecimal digits. Thus when US-ASCII, UTF-8, or any of the ISO-8859 character sets are in use, the letter "n" may be represented by "=6E," because the codepoint for "n" is 110 (equal to hexadecimal 6E) in each of those character sets. When the UTF-8 character set is in use, "=C5=84" can be used to represent "ń," because the codepoint 324 (hexadecimal 0144) is mapped to "ń" in Unicode; 324 is too large to be represented in a single octet, so it is broken up into two.[11]

47 This system, if used as described above, would result in an encoding that is three times larger than the unencoded information (because every character is represented by an equal sign followed by two characters, the hexadecimal digits) and impossible to read. So there is an important exception: any character from 33 (hexadecimal 21, "!") through 126 (hexadecimal 7E, "~"), inclusive, except for 61 (hexadecimal 3D, "=," the equal sign) may be represented by itself. That includes the 26 letters of the US-ASCII alphabet in both upper & lower cases. Thus "Espa=C3=B1ola" is normally used for the string "Española" rather than "=45=73=70=61=C3=B1=6F=6C=61."[12]

48 There are several other details to quoted-printable encoding (whitespace, end of line, and soft line breaks to be precise) but this is the main gist of it.

2.2.5.3.3 Base64

49 The base64 method of encoding content is also not particularly complex at its core. It has the advantages (over quoted-printable) of expanding input by a factor of ~133% rather than a factor of up to 300%, and of being easily applicable to arbitrary binary input (e.g., JPEGs or OGGs); it has the disadvantages of being somewhat more complex, and is completely unreadable before decoding.

50 In base64 the input is broken up into sets of 3 octets (24 bits). Each set of 24 bits is then divided into 4 sets of 6 bits. A sextet gives you a range of 64 numbers, 0–63. Each of those numbers is mapped onto a character that will survive transfer through an email system because it is the same in US-ASCII, ISO 646, and ISO 10646 (i.e., Unicode), etc.[13] Each of those characters, of course, is represented by an octet in the encoded version of the data. So every 24 bits of unencoded data takes 32 bits to represent in the encoded form, hence the 33% increase in size.

51 So the string "Española" is encoded in base64 as "RXNwYcOxb2xhCg==." The equal signs are used at the end for padding, as the input could not be divided into an integral number of 3-octet sets. In addition to padding there are special rules for line length and line ends. But the above is the important basis of the system.

2.2.5.3.4 Content Encoding, Lineation

52 Whether quoted-printable or base64 encoding is used, each line of the body of the email can be at most 76 characters long. Because base64 makes use of only 64 characters (or 65 if you count trailing equal signs), none of which are whitespace, base64 encoded content often looks like a large block of undecipherable letters and numbers.

53 Quoted-printable encoded content typically looks much more like the original content (because the whitespace and many letters are the same), but because the line length is limited to 76 characters, a special flag has to be used to signal when a single line of text has been encoded into two (or more) lines of encoded text. This flag is an equal sign (U+003D, "=") at the end of a line. (It is usually, but not always, the 76th character.)

2.2.5.3.5 Content Encoding Examples

54 In order to exemplify these content encoding methods, here we take a short snippet of sample text taken from a posting by Lionel Clement on Saturday, 18 December 1999 in which the second call for papers for the the upcoming TAG+5 conference is presented. The sample is shown here three times:

1. Using multibyte characters (in UTF-8, as is this entire article).
2. Using only 7-bit US-ASCII characters (i.e., every character is represented by an octet, of which the first bit is 0) with characters outside of US-ASCII represented using the quoted-printable algorithm.
3. Also using only 7-bit US-ASCII characters, but the entire passage has been encoded using the base64 algorithm.

Example 3. Snippet of plain text — 287 characters in 7 lines.

```
      DEUXIÈME APPEL À COMMUNICATIONS

   La cinquième conférence sur les grammaires d'arbres adjoints (TAG) et

autres formalismes proches (d'où le +) aura lieu à l'Université Paris
7 du 25 au 27 mai 2000 sous l'égide de l'ATALA (Association pour le
Traitement Automatique des Langues).
```

Example 4. Snippet in quoted-printable — 307 characters in 9 lines.

```
      DEUXI=C8ME APPEL =C0 COMMUNICATIONS

   La cinqui=E8me conf=E9rence sur les grammaires d'arbres adjoints (TAG)=
 et

autres formalismes proches (d'o=F9 le +) aura lieu =E0 l'Universit=E9 Par=
is
7 du 25 au 27 mai 2000 sous l'=E9gide de l'ATALA (Association pour le
Traitement Automatique des Langues).
```

Example 5. Snippet in quoted-printable — 389 characters in 6 lines.

```
ICAgICAgREVVWEnITUUgQVBQRUwgwCBDT01NVU5JQ0FUSU9OUwoKICAgTGEgY2lucXVp6G1lIGNv
bmbpcmVuY2Ugc3VyIGxlcyBncmFtbWFpcmVzIGQnYXJicmVzIGFkam9pbnRzIChUQUcpIGV0Cgph
dXRyZXMgZm9ybWFsaXNtZXMgcHJvY2hlcyAoZCdv+SBsZSArKSBhdXJhIGxpZXUg4CBsJ1VuaXZl
cnNpdOkgUGFyaXMKNyBkdSAyNSBhdSAyNyBtYWkgMjAwMCBzb3VzIGwn6WdpZGUgZGUgbCdBVEFM
QSAoQXNzb2NpYXRpb24gcG91ciBsZQpUcmFpdGVtZW50IEF1dG9tYXRpcXVlIGRlcyBMYW5ndWVz
KS4K
```

2.2.5.3.6 Content Encoding—NOT!

55 It is trivially easy to perform encoding or decoding of a quoted-printable or base64 encoded file because other people have written programs to do the work. In GNU/Linux, for example, the commands `qprint` and `base64` will perform either encoding to or decoding from quoted-printable and base64 respectively.[14]

56 Furthermore, many modern programming languages have libraries with functions that perform quoted-printable or base64 encoding and decoding. But, as far as we can tell, XSLT is not one of them. Our reading of the XPath 3.1 specification says that it provides the capability to encode a string of text *into* base64, but not to decode base64 data into a string. There is support for base64 using the EXPath extensions to XPath.

57 We did not attempt to decode quoted-printable or base64 encoded content mostly due to a lack of time in this small pilot project. That said, we are hesitant to employ EXPath for a tool intended for digital humanists, as the only EXPath-capable processors we are aware of are payware.

3. Mapping to TEI

58 In this section we represent some experimental modelings of email Listserv data in TEI encoding. We experimented with a combination of encoding from the newly introduced chapter on Computer Mediated Communication (CMC) and from TEI encoding for correspondence. In each of the methods shared in this section, we followed the guidance of the new Computer Mediated Communication chapter to store email postings at the same hierarchical level, rather than permitting them to nest when posts reply to original posts. This approach represents the email archive simply according to chronological sequencing of posts, without attempting to cluster posts with their replies. It was easy to prepare the archive in this way working from our source documents, because the TEI-L log files that we worked with to prepare these examples are arranged by chronological sequence and without topical nesting. In the interests of facilitating comparison of our encoding decisions, we have truncated the body of each message in the examples to show only the first and last few lines.

59 The email headers that were present in the entire dataset include the following information fields:

- Date
- From
- Reply-To
- Sender
- Subject
- MIME-Version
- Content-Type
- In-Reply-To
- Content-Transfer-Encoding
- Comments
- Message-ID
- Organization
- Content-Disposition
- X-cc (carbon copy)

In our efforts to model these data, we initially found the following fields could neatly map to TEI constructions associated with computer-mediated communication and correspondence encoding. However, they are not the only possible ways to apply the TEI to email data, and our examples in the following sections will explore some variations, especially in developing the `<teiHeader>`.

email header field	TEI location (in XPath)
Date	`post/@when`
From	`post/@who`
Reply-To	`correspDesc/correspContext/ptr[@type="reply-to"]`
Sender	`correspDesc/correspAction[@type="relayed"]/email`
Subject	`post/head`
In-Reply-To	`correspDesc/correspContext/ref[@type="in-reply-to"]`
Comments	`correspDesc/correspAction[@type="orig-(to|cc)"]/email`
Message-ID	`idno[@type="message-id"]`
Organization	[not yet represented]
X-cc	`correspDesc/correspAction[@type="orig-cc"]/email`

Note the vast majority of the `Comments:` fields (~93.7%) reproduce the original `To:` field; in addition, all of the `X-cc:` fields and most of the remaining `Comments:` fields (~5.6%) reproduce the original `CC:` field. The original `To:` and `CC:` fields are otherwise not available in the archive.

60 It is perhaps a little ironic that we prepared many of our examples for this paper from a 1990 conversation on TEI-L about encoding compound documents, given the compound nature of an email Listserv archive. In preparing distinct modelings of a TEI archive, we followed the method

of the Listserv monthly log documents that store the messages as a chronological sequence, and we needed to make decisions about how to associate related messages that reply or respond to one another.

3.1 One TEI Document for a Collection of Posts

61 In one approach to mapping the metadata for a collection, we envisioned a single `<TEI>` element holding a collection of `<post>` elements, each of which contains an email message. For this approach, we applied the `<teiHeader>` to store metadata about the original server location of the TEI Listserv and its log files in the `fileDesc/sourceDesc`. We encoded the method of our extraction of data from the TEI-L archive in the `encodingDesc/samplingDecl`, as shown in this example:

```
  <teiHeader>
   <fileDesc>
    <titleStmt>
     <title>Text Encoding Initiative public discussion list: January and February
1990</title>
    </titleStmt>
    <editionStmt>
     <edition>Second experimental (“alpha”) edition</edition>
    </editionStmt>
    <publicationStmt>
     <authority>
      <persName>Elisa Beshero-Bondar</persName>
      <persName>Syd Bauman</persName>
     </authority>
     <date when="2024-09-22"/>
     <distributor><orgName>GitHub</orgName></distributor>
     <address>
      <addrLine>88 Colin P. Kelly Junior Street</addrLine>
      <addrLine>San Francisco, CA 94107</addrLine>
      <addrLine>United States</addrLine>
     </address>
    </publicationStmt>
    <sourceDesc>
     <bibl>
      <title level="j">TEI-L Listserv</title>
```

```
      <title level="s">LOG9001</title>
      <title level="s">LOG9002</title>
      <publisher>University of Illinois Chicago</publisher>
      <distributor>TEI-L@UICVM</distributor>
      <date>1990</date>
      <relatedItem type="archive">
       <bibl>
        <publisher>The Pennsylvania State University</publisher>
        <distributor>LISTS.PSU.EDU Listserv Server (17.0)</distributor>
        <date>2024</date>
       </bibl>
      </relatedItem>
     </bibl>
    </sourceDesc>
   </fileDesc>
   <encodingDesc>
    <samplingDecl>
     <p>Sampled by requesting monthly logs from <name type="API">LISTS.PSU.EDU</
name> by
      email with GET commands: <code>GET TEI-L LOG 9001</code> (for January 1990).
One log
      command was issued for each month. See <ptr type="APIdoc" target="https://
www.lsoft.com/manuals/17.0/commands/14File-serverandwebfunctioncomma.html"/>.
      Received by email between <date from="2024-09-21" to="2024-09-22">September
21
      and 22, 2024</date>.</p>
    </samplingDecl>
   </encodingDesc>
  </teiHeader>
```

62 In the `<text>` of the TEI document, we encoded information about the collection of posts and the posts themselves. Since Listserv delivers the logs as files representing an entire month of posts, we supplied a unique identifier to each monthly log collection on `text/body/div[@type="log"]/@xml:id`. Within this `<div type="log">`, we encoded each email message inside a `<post>` element, drawn from the new CMC chapter.

63 To store the metadata for individual email messages, in this method we encoded a `<dateline>` element, which contained `<ref>` elements to store information about the transmission of the message within its parent `<post>`, thus:

Date	`date/@when` or `date/text()`
Reply-To	`ref[@type="reply-to"][@target="email:___]`
Sender	`ref[@generatedBy="system"][@type="sender"][@target="email:TEI-L@__]`
From	`ref[@generatedBy="template"][@type="from"][@target="email:___"]`
Subject	`title[@level="a"][@generatedBy="human"]`
In-Reply-To	`ref[@target="#id-of-earlier-posting]`

Notice that in this encoding we are experimenting with setting the Subject field in a `<title>` element, supplying a `@level` with a value of `"a"`, which seems fitting for a portion of an assembled collection. Here is an example encoding of a single email message encoded as a `<post>` following this method:

```
  <post xml:id="Web-1990-01-31-0933">
   <dateline>
    <date when="1990-01-31">Wed, 31 Jan 90 09:33:26 CST</date>
    <ref generatedBy="system" type="reply-to" target="email:TEI-L@UICVM">Text
Encoding
    Initative public discussion list </ref>
    <ref generatedBy="system" type="sender" target="email:TEI-L@UICVM">Text
Encoding
    Initative public discussion list</ref>
    <ref generatedBy="template" type="from"
target="email:WEBER@HARVARDA.BITNET">Robert
    Philip Weber</ref>
```

```
    <title level="a" type="subject" generatedBy="human">compound documents and
    images</title>
   </dateline>
   <p>could someone please explain the <name type="ML">TEI</name> approach to
compound
    documents and images? WIll <name type="ML">SGML</name> be used here, and if
so,
   how? I've just joined the list. sorry if this has been asked before.</p>
   <p>Many Thanks</p>
   <signed generatedBy="human">Bob Weber</signed>
   <signed generatedBy="template"> Robert Philip Weber, Ph.D. | Phone: (617)
495-3744
    <lb/>Senior Consultant | Fax: (617) 495-0750 <lb/>Academic and Planning
Services |
    <lb/>Division | <lb/>Office For Information Technology| Internet:
    weber@popvax.harvard.edu <lb/>Harvard University | Bitnet: Weber@Harvarda <lb/
>50
    Church Street | <lb/>Cambridge MA 02138 | </signed>
  </post>
```

Note that in this example original lineation is not preserved (except in the signature template), but said lineation was used as evidence for establishing paragraph boundaries.

3.2 One TEI Document for a Single Post

64 A collection of posts could conceivably be encoded as a collection of TEI documents, sharing the same root of `<TEI>` or `<teiCorpus>`. For this method, we would apply the same outer `<teiHeader>` as shown in previous examples, but then supply metadata about each post in the `<teiHeader>` elements of the internal `<TEI>` elements. This approach treats each email message directed to the Listserv more as we would encode written and mailed correspondence.

3.2.1 One TEI Document per Post with Subject in Title Statement

65 In this example, we supply the email message header information in a `<correspDesc>` rather than in the `<dateline>`, and we ventured to develop the `teiHeader/titleStmt` to deliver the subject line of a list message as its `<title>`, with the writer's name given as the author, and also the person responsible for the `correspAction/@type="sent"` (usually the same as the writer).

```
<TEI>
 <teiHeader>
  <fileDesc>
   <!-- same as previous example … -->
  </fileDesc>
  <encodingDesc>
   <!-- same as previous example … -->
  </encodingDesc>
 </teiHeader>
 <TEI xml:id="Web-1990-01-31-0933">
  <teiHeader>
   <fileDesc>
    <titleStmt>
     <title type="subjectLine">compound documents and images</title>
     <author>Robert Philip Weber</author>
    </titleStmt>
    <publicationStmt>
     <p>Text Encoding Initative public discussion list</p>
    </publicationStmt>
    <sourceDesc>
     <bibl><title level="s">LOG9001</title></bibl>
    </sourceDesc>
   </fileDesc>
   <profileDesc>
    <langUsage>
     <language ident="en">English</language>
    </langUsage>
    <correspDesc>
     <correspAction type="sent">
      <persName>Robert Philip Weber</persName>
      <email>WEBER@HARVARDA.BITNET</email>
      <date when="1990-01-31">Wed, 31 Jan 90 09:33:26 CST</date>
     </correspAction>
     <correspAction type="relayed">
      <orgName><ref target="mailto:TEI-L@UICVM">Text Encoding
       Initative public discussion list</ref></orgName>
     </correspAction>
    </correspDesc>
```

```
     </profileDesc>
    </teiHeader>
    <text>
     <body>
      <post>
       <p>could someone please explain the <name type="ML">TEI</name> approach to
compound
        documents and images? WIll <name type="ML">SGML</name> be used here, and
if so,
       how? I've just joined the list. sorry if this has been asked before.</p>
       <p>Many Thanks</p>
       <signed generatedBy="human">Bob Weber</signed>
       <signed generatedBy="template"> Robert Philip Weber, Ph.D. | Phone: (617)
495-3744
        <lb/>Senior Consultant | Fax: (617) 495-0750 <lb/>Academic and Planning
Services |
        <lb/>Division | <lb/>Office For Information Technology| Internet:
        weber@popvax.harvard.edu <lb/>Harvard University | Bitnet: Weber@Harvarda
<lb/>50
        Church Street | <lb/>Cambridge MA 02138 | </signed>
      </post>
     </body>
    </text>
   </TEI>
   <TEI xml:id="Spe-1990-02-06-1054">
    <teiHeader>
     <fileDesc>
      <titleStmt>
       <title type="subjectLine">compound documents</title>
       <author>Michael Sperberg-McQueen</author>
      </titleStmt>
      <publicationStmt>
       <p>Text Encoding Initative public discussion list</p>
      </publicationStmt>
      <sourceDesc>
       <bibl><title level="a">LOG9002</title></bibl>
      </sourceDesc>
     </fileDesc>
```

```
     <profileDesc>
      <langUsage>
       <language ident="en">English</language>
      </langUsage>
      <correspDesc>
       <correspAction type="sent">
        <persName>Michael Sperberg-McQueen</persName>
        <email>U35395@UICVM.BITNET</email>
        <date when="1990-02-06">Tue, 6 Feb 90 10:54:04 CST</date>
       </correspAction>
       <correspAction type="relayed">
        <orgName><ref target="TEI-L@UICVM">Text Encoding
         Initative public discussion list</ref></orgName>
       </correspAction>
       <correspContext>
        <ref type="in-response-to" target="#Web-1990-01-31-0933">previous message
of
         <persName>Robert Philip Weber</persName> to the TEI-L Listserv.
         <date when="1990-01-31"/>
        </ref>
       </correspContext>
      </correspDesc>
     </profileDesc>
    </teiHeader>
    <text>
     <body>
      <post>
       <p>About compound documents in <name type="ML">SGML</name> and in the <name
type="ML">TEI</name>. <ref target="#Web-1990-01-31-0933"><persName>R.P. Weber</
persName>
         asked a week ago <q>could someone please explain the <name
type="ML">TEI</name>
          approach to compound documents and images? WIll <name type="ML">SGML</
name>
         be used here, and if so, how?</q></ref></p>
       <p>Apologies for my delay in answering. I was hoping one of our hypertext
sages might
        weigh in with a reply. (But he appears to have been in the
```

```
        <placeName>Caribbean</placeName>, and may not have received the query.)</
p>
       <!-- 8 paragraphs omitted … -->
       <p>Perhaps those subscribers to this list who actually work with compound
documents
        and <name type="ML">SGML</name> will be willing to say how they make
things work
       now, and how they would like to see things developing in the future.</p>
       <p> All this is, I repeat, just personal opinion and shouldn't be taken as
defining
        <soCalled>the</soCalled> position of the <orgName>TEI</orgName>. (Unless,
of
        course, taking as <soCalled>the</soCalled> position will help get a
discussion
       started.)</p>
       <signed generatedBy="human">-Michael Sperberg-McQueen<lb/> University of
Illinois at
       Chicago </signed>
      </post>
     </body>
    </text>
   </TEI>
  </TEI>
```

3.2.2 One TEI document per Post with Subject in its <head>

66 In the process of automating the conversion of Listserv data to XML, we prepared a rather different version of a <TEI> element that stores a single email message. Like the previous example, each email message is stored in a TEI child of a TEI document, but in this approach, the email message metadata has been stored in different locations:

- in a <correspDesc> within the <teiHeader>,
- in the attributes of each <post>,
- in a <head> first child of each <post>, and, in addition,
- a copy of the original field values in a <xenoData> within the <teiHeader>

```
<TEI>
```

```
   <teiHeader>
    <fileDesc>
     <titleStmt>
      <title><!-- basename of input document --></title>
      <title type="sub">a corpus of 38787 postings</title>
     </titleStmt>
     <publicationStmt>
      <ab>Currently an experimental document. The original source was
      published as part of a public mailing list, so this
      document similarly is publicly available.</ab>
     </publicationStmt>
     <sourceDesc>
      <ab>Derived from source file /path/to/INPUT.txt, which
      should be a log file from a Listserv mailing list.</ab>
     </sourceDesc>
    </fileDesc>
    <encodingDesc>
     <appInfo>
      <application ident="listserv_log2cmc.xslt" version="0.1">
      This file generated 2025-02-25T14:11:16.933643722-05:00 by file:/path/to/
listserv_log2cmc.xslt
      using /path/to/INPUT.txt as input.
      </application>
     </appInfo>
    </encodingDesc>
    <particDesc>
     <!-- <listPerson> here, see discussion below … -->
    </particDesc>
   </teiHeader>
   <!-- 4 <TEI> elements missing here … -->
   <TEI xmlns:tmp="http://www.wwp.neu.edu/temp/ns" n="00005" xml:id="TEI-
L.txt_msg_00005">
    <teiHeader>
     <fileDesc>
      <titleStmt>
       <title>TEI-L posting from INPUT.txt, #5</title>
      </titleStmt>
      <!-- <publicationStmt> from corpus header repeated here … -->
```

```
      <!-- <sourceDesc> from corpus header essentially repeated here … -->
     </fileDesc>
     <!-- <encodingDesc> from corpus header essentially repeated here … -->
     <profileDesc>
      <correspDesc>
       <correspContext>
        <ptr type="reply-to" target="mailto:TEI-L@UICVM"/>
       </correspContext>
       <correspAction type="relayed">
        <email>TEI-L@UICVM</email>
        <date>Tue, 6 Feb 90 10:54:04 CST</date>
       </correspAction>
       <note type="Comments">"ACH / ACL / ALLC Text Encoding Initiative"</note>
      </correspDesc>
     </profileDesc>
     <xenoData>
      <tmp:Date>Tue, 6 Feb 90 10:54:04 CST</tmp:Date>
      <tmp:Reply-To>Text Encoding Initative public discussion list &lt;TEI-
L@UICVM&gt;</tmp:Reply-To>
      <tmp:Sender>Text Encoding Initative public discussion list &lt;TEI-
L@UICVM&gt;</tmp:Sender>
      <tmp:Comments>"ACH / ACL / ALLC Text Encoding Initiative"</tmp:Comments>
      <tmp:From>Michael Sperberg-McQueen 312 996-2477 -2981
&lt;U35395@UICVM.BITNET&gt;</tmp:From>
      <tmp:Subject>compound documents</tmp:Subject>
     </xenoData>
    </teiHeader>
    <text>
     <body>
      <post who="#u35395·#·uicvm.bitnet" when="1990-02-06T10:54:04-06:00">
       <head type="subject">compound documents</head>
       About compound documents in SGML and in the TEI. R.P. Weber asked a
       <lb n="1"/>week ago "could someone please explain the TEI approach to
compound
       <lb n="2"/>documents and images? WIll SGML be used here, and if so, how?"
       <lb n="3"/>
       <lb n="4"/>Apologies for my delay in answering. I was hoping one of our
hypertext
```

```
       <lb n="5"/>sages might weigh in with a reply. (But he appears to have been
in the
       <lb n="6"/>Caribbean, and may not have received the query.)
       <!-- 87 lines deleted here to reduce size of example … -->
       <lb n="88"/>All this is, I repeat, just personal opinion and shouldn't be
taken as
       <lb n="89"/>defining "the" position of the TEI. (Unless, of course, taking
as "the"
       <lb n="90"/>position will help get a discussion started.)
       <lb n="91"/>
       <lb n="92"/>-Michael Sperberg-McQueen
       <lb n="93"/> University of Illinois at Chicago</post>
     </body>
    </text>
   </TEI>
  </TEI>
```

Because this encoding was generated by an (XSLT) application, information about that application is stored in `<appInfo>` in the `<encodingDesc>`. And although the metadata expressed in the email header fields is mapped to various TEI constructs, a verbatim copy of the value of each field is retained (in `<xenoData>`).[15] The values of the field names are used as the local names of elements within `<xenoData>`, but are not used verbatim. Since email header fields are case insensitive, we have removed case distinctions when translating field names to the local names of elements.

67 The repetition of metadata in `<xenoData>` serves two purposes. The first (and for now, primary) purpose is to make debugging our program easier. But we anticipate that reporting the raw data will be useful to those researchers who are well versed in XPath and interested in details of email communication. For example, determining exactly how the `From:` field was punctuated or what capitalization was used for the "quoted-printable" transfer encoding is not otherwise possible using the output of our program.

68 Similar to our previous encoding, the `<correspDesc>` also stores the distinct identifier of the message to which this is a reply. Perhaps the most significant difference is moving the information about the sender of the message to the `<post>` element, where the sender is designated on the `@who` with a distinct identifier, mapped from their email address, and the date and time of initial sending supplied on `@when`.

69 One field that gave us significant pause is the `Subject:` field. Although such fields are ubiquitous in office memos and in email headers, they are not a normal part of either physical correspondence or the sorts of media CMC encoding was intended for. We have experimented with both encoding the subject line as the heading of the `<post>` (using the `<head>` element) and as the `<title>` in the TEI header. Neither seems entirely satisfactory.

70 Encoding the `Subject:` as the heading of the `<post>` results in a a very usable encoding that is easy to understand and process. But it leaves us unsettled because it implies that the subject information was originally part of the email body (as is the rest of the content of `<post>`), which it was not—it was metadata, albeit directly associated with the body.

71 While encoding the `Subject:` in the `<title>` inside the `<titleStmt>` correctly puts it in a metadata container, it is not the correct container. This `<title>` is supposed to be the title of the TEI encoded file, not of that which is encoded. Besides, we are uncomfortable equating a subject and a title.

72 One possible future encoding of this information would be to store both the title of the TEI document and the subject of the encoded email as separate `<title>` elements in the `<titleStmt>`, differentiated by `@type` or nesting. But we have not attempted to implement this yet.

73 In this example, the human authorship of the message and the designation of its subject is entirely designated within the `<post>`, in a manner more like the way CMC encoding approaches discussion board or wiki posts, and less like the way we would handle the encoding of a letter as part of a collection of correspondence. While letters do not usually have subject lines, the personal actions of transmission and receipt are typically addressed in the `<teiHeader>`, wherein the `<titleStmt>` often provides information about the exchange, and the `<correspDesc>` is used to provide detailed information about how messages literally change hands, from the writer to the postal agent to the recipient.

3.3 Emailography?

74 An archive of an email list is likely to, and in our case does, contain posts from many members of its community. And information about members of the community, particularly about historic or inaugural figures, is likely to be of interest to the community. Thus it seemed prudent for our XSLT process to develop a set of structured information about participants in the email exchanges

recorded in the source archive. If nothing else, TEI CMC encoding of a `<post>` uses such a structure to indicate who is the author of the posting (using `@who`, a pointer to the structured information about the author of the posting).

75 However, the only information about individual participants in the Listserv archives that is available in highly structured form are their email addresses and names, and even the names are sometimes absent. To make matters worse, a name, when present, is that which the author's email client uses, which may not be entirely under the author's control.[16]

76 Other information about posters to the list may be available (e.g., from their signatures, or by looking at the dates on which they posted), but only names and email addresses are readily available in dedicated metadata fields. Nonetheless, an automatically generated structured "'ography" of those who post seems like a good idea, both as a starting point for those wishing to add further information about parties involved, and as a place for the `@who` attribute of `<post>` to point. But how should we structure this set information? Eschewing complex calculations or heuristics for now, we only have two relevant pieces of information about the sender of any given piece of email: a name and an email address. And even though there is a strict one-to-one relationship between these pieces of information *for a particular post*, many people post multiple times to the list.

77 It is often the case that for a given name multiple email addresses were used. For example, the email address emcaulay@library.ucla.edu is associated with three different names: "McAulay, Elizabeth," "Elizabeth McAulay," and "McAulay, Lisa." (It is easy for a human to see these are probably the same person. It would be far more difficult, although not impossible, for a computer to make that inference.) In another case the email address gusfer@gmail.com is associated with two distinctly different names (for the same person): "Gus Riva" and "Gustavo Fern=?UTF-8?Q?=C3=A1ndez_Riva?=" (which, according to the rules sketched out in section 2.2.5.3.2, could be expressed as "Gustavo Fernández Riva").

78 Conversely, for any given name, several email addresses may have been used in different postings. For example, the name "C. M. Sperberg-McQueen" is associated with U35395@UICVM.BITNET, U59467@UICVM.BITNET, and cmsmcq@acm.org.

79 Something like a TEI "personography" data structure (a <listPerson> with a series of <person> elements) is called for. Our first attempted program actually generates exactly this personography structure, but we are unsatisfied with this encoding. Before discussing why we are unsatisfied with our own encoding, it is worth pointing out the two approaches that can be taken when generating the bare bones personography. Given that the only types of information we have are names and email addresses, we could create either a structure of entries identified by email address, each of which lists the various names associated with that address; or a structure of entries identified by name, each of which lists the various email addresses associated with that name.

80 We chose the former (a structure based on email addresses, each of which lists one or more names that are associated with it), primarily because it is a lot easier to convert the email addresses to XML IDs than it is to convert the names to XML IDs. There are only seven characters that occur in our set of email addresses that cannot occur in an XML ID: @, :, !, +, %, /, and ~. Conversely, in the set of names extracted from the From: fields of this dataset, there are over a dozen characters not allowed in an XML ID including some which make manipulating them in XSLT annoying: ', ", and &. Others include ©, ¶,), (, ¡, Ð, and ¼.

81 So for each unique email address in the From: fields of our dataset we generated a <person> element that had an @xml:id that was generated from the given email address, and from which the email address could be reproduced.[17] Inside that <person> element we list each unique name associated with it, and then reproduce the email address.

82 We find it dissatisfying that the only way to include an <email> inside a <person> is to place it inside a <p>, <ab>, or <note> that is itself inside the <person>. This feels clumsy to us. After all, an email address is a key piece of information many of us wish to record about people. That said, the correspondence between people and email addresses is not at all a simple one, so a simple child relationship may be insufficient. Furthermore, email addresses come and go, so at the very least <email> would have to be added to the att.datable class in order to make it usable as a child of <person>. It might be reasonable instead to make <email> a child of <state> (which, in turn, is available as a child of <person> and a member of att.datable and att.typed).

83 We can imagine further processing that might enhance this data. For example, a routine could check to see the chronologically earliest and latest use of each name and record that information. It might be reasonable to look for each name in *other* entries and generate links between entries

that contain the same name. One might group entries by the email address domain name. For each entry, minimum, average, and maximum lengths of posts might be included. (Although it is an interesting question which MIME part one uses to calculate length, and how the calculation is performed.)

84 So while we are not entirely satisfied with our current "personography" structure, we are not entirely *dissatisfied* with it, either. But either way, we are uncomfortable with our data structure in principle, because it is *not really a personography*. It is not a set of data structures about people, but rather it is a set of data structures about email addresses. We could, of course, use generic structures instead of abusing the special purpose elements. An example follows.

```
<list type="emailography">
 <item xml:id="nsmith·#·email.unc.edu">
  <name>Natalia Smith</name>
  <name>Natasha Smith</name>
  <email>nsmith@email.unc.edu</email>
 </item>
 <!-- hundreds of other <item>s here -->
</list>
```

Note the use of the character sequence `·#·` (MIDDLE DOT, FULLWIDTH COMMERCIAL AT, MIDDLE DOT; U+00B7, U+FF20, U+00B7) instead of just "@" (COMMERCIAL AT; U+0040). This is because an "@" is not permitted in an XML Name, but "·" and "#" are, per XML.

85 While a generic representation is possible, an emailography might be better encoded with the proposed `<listEntity>` element as discussed on TEI issue #2341. An `<entity>` is more precise than a generic `<item>`, and could be (depending on the definition used for `<entity>`) far more applicable than `<person>`. In any case, the limited locations permitting the already established `<email>` element seem short-sighted as the TEI contemplates the archiving of born-digital texts.

4. Conclusion: Improving the TEI for the Encoding of Born-digital Resources?

86 This paper has explored several challenges and possibilities for the TEI to encounter the born-digital resource of the email list, and presented quite a few potentially interesting statistics along the way. Since the digital text is so ephemeral and contingent, yet so full of information about the computer age, our methods in archiving such work should help to sustain it as a resource for scholarly research, rescuing it from loss from changes in technology providers that take little care for maintaining access to or preserving heritage. In our specific efforts to archive Listserv data, we find that the TEI can benefit from some improvements.

87 The limited valid positions for `<email>` in TEI documents are a concern as we begin to develop emailographies, and also as we need to reference email addresses as primary resource of information in born-digital documents. That `<email>` cannot be a child of `<person>` (or the basis for its own list) raised problems for us. More robust encoding should be developed.

88 The `<post>` element, as a descendant of `<text>`, contains "normal" transcriptional elements for encoding the logical structure of a document. But given that at its core an email message is definitionally a series of characters, and (because indication of line breaks is well defined) they can easily be considered as a sequence of lines, in some situations it makes sense to use an embedded transcription approach.

> An *embedded transcription* is one in which [the characters] … are encoded as subcomponents of elements representing the physical surfaces carrying them rather than independently of them.
>
> (P5 § 12.2.2)

In TEI this approach makes use of `<line>` elements within a `<sourceDoc>`. However, in the case of an email message, there are no physical surfaces, so the intermediate elements (`<surfaceGrp>`, `<surface>`, and `<zone>`) do not seem applicable. So while `<line>` within `<sourceDoc>` may not be the right mechanism, some method for representing email as a relatively uninterpreted sequence of lines of characters should be available; or at the very least, we need to be able to preserve the ASCII art of yesteryear.

89 The TEI provides no good way to encode an email Subject line in either <correspDesc> or <dateline>, and our attempts to place the Subject varied a little too widely. We do not find the use of the <head> element in <post> satisfactory to encode an email Subject because it is not really part of the email body, but part of the heading metadata stored with From, To, CC, etc. Setting it in the <titleStmt> certainly presents the Subject line as metadata, but it is then metadata about the encoded TEI document, not necessarily the source. This could be overcome by consistent use of a system that included the latter in the former. (E.g., if each <TEI> that represented a posting to the list had a <title> of `A TEI encoding of TEI-L post #04570 of <date when="2002-11-04T11:54:15">Mon, 4 Nov 2002</date> by <persName>Seal, Jill</persName> with subject <title level="a" type="subject">Re: multiple page range in master</title>`. Verbose, but it works.)

90 Contemplating the use of the <title> element raises questions about the available values for @level. What @level values are appropriate for the <title> of the TEI-L Listserv as a whole, for the log files it curates by the month, or for the individual email message if we use a <title> element to store its Subject line? Perhaps we could apply a @level of "s" (for series) for the title of the collected archives of a Listserv mailing list, a @level of "m" (for monographic) for the title of a single monthly log file, and a @level of "a" (for analytic) for the title of a single post to the list. But monographic, usually associated with bound books or works concentrating purposefully on a single topic, does not seem at all appropriate for a log file of posts sent conversationally to an email Listserv. Perhaps @level of "j" would be marginally more appropriate for the log files, but they do not bear much resemblance to journal or magazine publications with planned contents. Do we need new values of @level for born-digital compound artifacts?

91 Identifying these issues is the first step to introducing new encoding models. We hope that our work in progress provides a basis for further discussion, debate over our experimental data modeling of the TEI Listserv, one of our oldest and most important resources in the TEI community.

APPENDIXES

Appendix 1. The 29 7-bit ASCII Characters that are not Legal in XML

base 10	base 16	base 02	symbol	description
0	00	0 000 000	NUL	Null character
1	01	0 000 001	SOH	Start of Heading
2	02	0 000 010	STX	Start of Text
3	03	0 000 011	ETX	End of Text
4	04	0 000 100	EOT	End of Transmission
5	05	0 000 101	ENQ	Enquiry
6	06	0 000 110	ACK	Acknowledge
7	07	0 000 111	BEL	Bell, Alert
8	08	0 001 000	BS	Backspace
11	0B	0 001 011	VT	Vertical Tabulation
12	0C	0 001 100	FF	Form Feed
14	0E	0 001 110	SO	Shift Out
15	0F	0 001 111	SI	Shift In
16	10	0 010 000	DLE	Data Link Escape
17	11	0 010 001	DC1	Device Control One (XON)
18	12	0 010 010	DC2	Device Control Two

19	13	0 010 011	DC3	Device Control Three (XOFF)
20	14	0 010 100	DC4	Device Control Four
21	15	0 010 101	NAK	Negative Acknowledge
22	16	0 010 110	SYN	Synchronous Idle
23	17	0 010 111	ETB	End of Transmission Block
24	18	0 011 000	CAN	Cancel
25	19	0 011 001	EM	End of medium
26	1A	0 011 010	SUB	Substitute
27	1B	0 011 011	ESC	Escape
28	1C	0 011 100	FS	File Separator
29	1D	0 011 101	GS	Group Separator
30	1E	0 011 110	RS	Record Separator
31	1F	0 011 111	US	Unit Separator

Regular Expressions to Match these Characters.

PCRE, Java, or oXygen:

```
[\u0000-\u0008\u000C\u000E-\u0019]
```

W3C:

```
[#x00-#x08#x0C#x0E-#x19]
```

Emacs:

```
[^#[:print:]]   <!-- where "#" is typed CTL-J -->
```

grep or egrep:

```
[^[:print:]]
```

Appendix 2. Date Format Frequency Chart

See below the chart for a glossary of the format components.

# occurrences	format
32632	Day, #D Mon YYYY HH:MM:SS ±HHMM
3136	Day, #D Month YYYY HH:MM:SS ±HHMM
2347	Day, #D Mon YYYY HH:MM:SS TZA
281	Day, #D Mon YY HH:MM:SS TZA
245	Day, #D Month YYYY HH:MM:SS TZA
73	Day, #D Mon YY HH:MM:SS ±HHMM
22	Day, #D Mon YY HH:MM:SS LCL
20	Day, #D Month YY HH:MM:SS LCL
8	Day, #D Mon YYYY HH:MM:SS LCL
7	Day, #D Month YY HH:MM:SS TZA
3	Day, #D Mon YY HH:MM:SS bst
2	Day, #D Mon YY HH:MM:SS MSZ
2	Day, #D Mon YY HH:MM:SS TZA
2	Day, #D Mon YYYY HH:MM:SS MSZ
1	Day, #D Mon YY HH:MM:SS pst
1	Day, #D Mon YYYY HH:MM:SS 0000
1	Day, #D Mon YYYY HH:MM:SS GMT0BST
1	Day, #D Mon YYYY HH:MM:SS METDST
1	Day, #D Mon YYYY HH:MM:SS TZONE

1	Day, #D Month YY HH:MM:SS METDST
1	Day, #D Month YY HH:MM:SS ±HHMM

Where:

Day

Weekday name expressed as 3 letters, capitalized (e.g., “Sun.”)

#D

One or two digit day-of-the month (e.g., “4” or “26.”)

Month

The full month name (e.g., “August.”)

Mon

The month in 3 letters (e.g., “Aug.”)

YYYY

A 4-digit year.

YY

A 2-digit year.

HH

A 2-digit hour.

MM

A 2-digit minute.

SS

A 2-digit second.

±

Either “+” or “-.”

TZA

A valid time zone symbol.

Anything else as last token

An invalid time zone symbol.

BIBLIOGRAPHY

Kosek, Jirka and Lumley, John, *Binary Module 1.0* EXPath Module 3, December 2013; https://expath.org/spec/binary.

Freed, N. and Borenstein N., Eds., *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, RFC 2045, November 1996; 10.17487/RFC2045.

Moore, K., Ed., *MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text*, RFC 2822, April 2001; 10.17487/RFC2047.

Resnick, P., Ed., *Internet Message Format*, RFC 2822, April 2001; 10.17487/RFC2822.

———. *Internet Message Format*, RFC 5322, October 2008; 10.17487/RFC5322.

TEI Consortium, eds. *TEI P5: Guidelines for Electronic Text Encoding and Interchange*. 4.9.0. 2025-01-24. TEI Consortium. https://www.tei-c.org/Vault/P5/4.9.0/doc/tei-p5-doc/en/html/index.html 2025-03-02.

Bray, Tim; Paoli, Jean; Sperberg-McQueen, C. M.; Maler, Eve; and Yergeau, François; Eds., *Extensible Markup Language (XML) 1.0 (Fifth Edition)*, November 2008; https://www.w3.org/TR/REC-xml/.

NOTES

1 ~41.2% of email messages in the entire dataset have a `Message-ID` header field which provides a unique identifier for the message. But this is not the same identifier Listserv uses for its display, and over half of them are missing, making this field not useful for some purposes.

2 This was done quite easily in Emacs using dired-mode. But in fact, it is not very hard to do directly in the shell (the single command

```
for f in tei-l.log* ; do
   echo "---------${f}:" ;
   if [[ ${f} =~ tei-l.log(9[0-9])([0-9][0-9]) ]] ; then
      mv ${f} tei-l.log_19${BASH_REMATCH[1]}-${BASH_REMATCH[2]} ;
   elif [[${f} =~ tei-l.log([012][0-9])([0-9][0-9]) ]] ; then
      mv ${f} tei-l.log_20${BASH_REMATCH[1]}-${BASH_REMATCH[2]} ;
   fi ;
done
```

does a nice job), and there are utilities available for Mac OS that could do this.

3 In fact, in a vain effort to avoid the character encoding problems described below, we actually used `iconv -f ISO-8859-1 -t UTF-8 -c tei-l.log* > TEI-L.txt`, but we do not think it made any significant difference.

4 That is, we searched for the regular expression `^={73}\n[^D]`.

5 It is quite reasonable, in our opinion, for an astute reader of the XML specification to think that production 14 implies that *any* character (other than < and &) is allowed as data content. But production 14 needs to be read within the limits already established by production 2 — which prohibits these control characters.

6 In fact, early definitions of email messages assert they are just a sequence of ASCII characters in the range of 1–127; see rfc2822. Later RFCs updated this to say "the use of US-ASCII control characters (values 1 through 8, 11, 12, and 14 through 31) is discouraged since their interpretation by receivers for display is not guaranteed."

7 As it is, finding the problematic characters, figuring out what each was likely supposed to be (often by multiple visits to the Internet Archive's Wayback Machine), and replacing them took multiple hours.

8 Thanks to our transformation to TEI, it took us only minutes to discover that of the 38,787 timestamps for all posts, 38,716 (99.8%) have an on-the-hour time zone indication, 51 (0.1%) have no time zone indication, 19 have an on-the-half-hour time zone indication, and 1 has a clearly erroneous time zone indication of `-00:08`. We expect that it would be easy to ascertain this using any of a variety of programming languages reading either the original dataset or our TEI encoded version. But given that we are experts in XPath and XSLT, having this information in TEI made this discovery very easy *for us*.

9 Which these days is essentially always. In our dataset ~92.6% of posts use MIME. The first 1841 posts occurred over the first 2531 days (6.93 years) and did not use MIME; the first use of MIME was on Friday, 13 December 1996. Over the next 2531 days 3327 of 3707 posts (89.75%) used MIME. Over the last 2531 days in the dataset 4854 of 4861 posts (99.86%) used MIME.

10 This name also appears as "=?ISO-8859-2?Q?Piotr_Ba=F1ski?=" and "=?UTF-8?B?UGlvdHIgQmHFhHNraQ==?=."

11 The octets are 11000101 10000100. In the first octet the first 3 bits (110) assert that this number is expressed in 2 octets, the last 5 bits (00101) are the first 5 bits of the codepoint number. In the second octet the first 2 bits (10) are the signal that this is a continuation of the previous octet, and the last 6 bits are the last 6 bits of the codepoint number. Thus we can read this UTF-8 character expressed using two bytes as the 11-bit binary codepoint 00101000100, which is 324 in decimal or 144 in hexadecimal.

12 This means that when used on Western texts that have a high percentage of characters in the standard US-ASCII set, quoted-printable results in relatively easy-to-read, compact encodings. Sadly, however, when used on Cyrillic, Greek, or CJK text, or arbitrary binary data, the encoded version is both large and completely unreadable.

13 The "alphabet" used to represent the 64 possible values is `A`–`Z`, `a`–`z`, `0`–`9`, `+`, and `/`.

14 In fact, these are the commands we used to generate the examples in the previous subsection.

15 The fields are currently stored using elements from a temporary namespace. We are considering using a namespace URI that refers to RFC 5322 (perhaps `https://datatracker.ietf.org/doc/html/rfc5322`), and using a prefix of `rfc5322:`.

16 For example, the Northeastern University mail system puts a user's name in the `From:` field in "Surname, Forename" order.

17 That is, the conversion process from email address to XML ID is deliberately round-tripable. We have not, however, yet written code to convert an ID back into an email address, and likely never will, given that the email address is stored elsewhere in the `<person>` entry.

AUTHORS

SYD BAUMAN

Senior XML Programmer/Analyst Northeastern University member TEI Technical Council

ELISA BESHERO-BONDAR

Professor of Digital Humanities Director of the Digital Humanities Lab Penn State Erie, The Behrend College chair TEI Technical Council