Provenance, Journal of the Society of Georgia Archivists

Volume 26 | Number 1 Article 4

January 2008

The Elusive Simplicity of Container-Level Encoded Archival Description: Some Considerations

Leah Broaddus
Southern Illinois University Carbondale

The Elusive Simplicity of Container-Level Encoded Archival Description: Some Considerations

Leah Broaddus

INTRODUCTION

Web-managed finding aids require streamlined, efficient intellectual organization of materials. It is not just a question of aesthetics, but of pragmatics. A more consistent, generalizable system of organization aids institutions in adopting, migrating, and building on the structure. The generalizable elements of a solution can be repeated, predicted, explained, taught, and further developed.[1] They also lend the skeletal structure necessary to support unique elements.

Pinning down the "unique" and "non-unique" elements of archival finding aids has been a long and complex process. Part of the early impetus for doing so cooperatively was the push toward the creation of an Encoded Archival Description (EAD) Document Type Definition (DTD). This was to be a scripted language, much like the more commonly known HyperText Markup Language (HTML), for describing and posting the standardized elements of finding-aid documents to the World

[1] Conversation with University of Illinois math graduate student Dan Lior, October 30, 2007, Champaign, Illinois.

Wide Web, allowing for some higher interactive Web functions. According to Dennis Meissner, leading up to the release of version 1.0, many of the archivists involved in the push seemed to nurse some small, defiant hope that for their institutions the smallest number of collection-description revisions possible would be required in order to bring them into compliance with new Web structures.[2] Even the smallest changes to any of the thousands of local finding-aid structures would require human resources that few archives had available or could afford. Making changes to physical labels on thousands of boxes was so impracticable that the very idea was understandably offensive to contemplate.

Choosing their battles, the creators of EAD, according to Janice E. Ruth, focused on creating a standard hierarchical structure for collection-level data.[3] To their immense credit, it is now a relatively simple process to transfer collection-level data between institutions and software platforms. The quest that archives have not yet followed to its labyrinthine conclusion, however, is that of creating a software-compatible, peer-institution-transferable, standardized container-level Extensible Markup Language (XML) hierarchy. In the interim, EAD is very cleverly structured to accommodate a near-infinite system of possible data-hierarchies and arrangements at the container level, and no single piece of collection-administration software can or could ever navigate all of them. Hence, every archive's container-list structure is local or nearly local.

The purpose of this article is to advocate the development of a structural goal towards which container-level data standards might evolve over time, and to contribute to the needed corpus of hypotheses in order to arrive at a solution to the problem of universal transfer. To this end, a hypothesis is posited which points to a possible standardized solution. Illustrative examples are then presented.

[2] Dennis Meissner. "First Things First: Reengineering Finding Aids for Implementation of EAD," *American Archivist* 60, n. 4 (2007): 373.

[3] Janice E. Ruth. "Encoded Archival Description: A Structural Overview," *American Archivist* 60, n. 3 (1997): 316.

BACKGROUND

One frequent conception among newcomers to EAD and Web-database-driven administrative software for the management of finding aids is that such programs and DTDs were written so that archivists would be able to put finding aids on the Web. This is not really an adequate summary of the goal, however. Finding aids had been put on the Web by many simpler and more widely supported full-text searchable methods. Even Gopher, as described by Michael Holland and Elizabeth Nielsen in 1995, supported full-text searching via the Internet.[4] HTML documents were all as full-text searchable as a specialized archival XML document later would be. But Holland and Nielsen also believed that full-text was not enough; it did not "relieve one of the responsibility of following established professional guidelines for arrangement and description, including rigorous subject analysis and vocabulary control."[5] According to Daniel Pitti and Wendy M. Duff six years later, "during the early stages of EAD, many asked why it was necessary, arguing that HTML appeared to be 'good enough' to do the job."[6] This is probably because there was, and still is, some lack of universal clarity as to what that "job" was to be.

EAD documents provide a large hierarchical template for a collection, and to represent a collection in the hierarchy an archivist must first shred a finding aid into standardized pieces and group them into levels. The point of the shredding and the standardized groupings and hierarchies is to lend machine-readable meaning to the archival information elements that underlie the visual display. The computer needs to be able to use the arrangement to translate the content according to an XML DTD that tells it what to expect to find, and where. As Stephen J. DeRose phrased it in 1997, "Structured information is information that is analyzed. [O]nly when information has been

---

[4] Michael Holland and Elizabeth Nielsen. "Gophers in the Archives: Planning and Implementing an archives and Records Management Gopher" *Provenance* XIII (1995): 27.

[5] Ibid., 44-45.

[6] Daniel V. Pitti and Wendy M. Duff, "Introduction," *Encoded Archival Description on the Internet*, Pitti and Duff, eds. (New York: The Haworth Press, Inc., 2001), 3.

divided up by such an analysis and the parts and relationships have been identified, can computers process it in useful ways."[7]

Because EAD XML limits what tags can be used inside of other tags, the computer can discern infinitely recurring hierarchical relationships. For computers, "navigation requires naming."[8] The nature of the data is recognizable by looking at where the data is filed. The location serves as a structurally defined "name" for the piece of data. When EAD was created, the idea was that if every institution used a standard EAD tag-system to store its data, not only would any institution be able to take in foreign EAD trees from any other and display them using a local stylesheet, but it would be possible to do other things, like create a stylesheet modeled to look like a Swiss cheese version of a library catalog entry to create a draft of a MAchine-Readable Cataloging (MARC) record. The designers of EAD intended that eventually such use of the structure would be possible, though they did not include it in the primary EAD development project.

As Janice Ruth has written, "The group … felt that it would be burdensome and unwieldy for EAD to be structured so that a complete MARC record could be harvested automatically from the SGML markup," but "for those MARC-like elements already represented in EAD, the team added an optional ENCODING ANALOG attribute, which permits the designation of the applicable MARC field or subfield together with the authoritative form of the data."[9]

A person does not need to have an EAD tag hierarchy in place to put a finding-aid display on the Web, and someone visiting a Web site can successfully use a non-EAD finding aid, but without the hierarchies underneath the display, or an administrative software program with spreadsheet hierarchies that tell what is grouped with—and ranked under—what, meta searches cannot recognize the nature of the pieces of data in the finding aid, down to the granular level required for a successful federated archival reference-search.

[7] Steven J. DeRose, "Navigation, Access, and Control Using Structured Information," *American Archivist* 60, n. 3 (1997): 299.

[8] Ibid., 301.

[9] Ruth, *Encoded Archival Description*, 316.

EAD was meant to allow a researcher to search the archives of the entire world all at once, by typing in a question that could be interpreted and answered by all the many and different worldwide machines. Daniel Pitti and Wendy M. Duff called this ideal “union access” and they predicted that users would “be able to discover or locate archival materials no matter where they are located in the world” and that “Libraries and archives will be able to easily share information about complementary records and collections and to ‘virtually’ integrate collections related by provenance, but dispersed geographically or administratively.”[10] This was to be accomplished by convincing everyone to use the same EAD structure and applying tags in a software-generalizable manner. It was also meant to ensure that if one university sent another a file containing one of their collections’ EAD documents, the new institution could download it straight into their EAD reader and have no trouble whatsoever digitally storing it and “parsing” or parceling out the data into local hierarchies. The goal was that the local program should be able to pack a foreign finding aid away with the rest of the native finding aids just as if it had been created locally. This ideal, however, has not yet come to fruition.

FROM EAD TO COLLECTION ADMINISTRATIVE SOFTWARE

According to a Web survey of fifty-four institutions done by Xiaomu Zhou in 2006, database-driven structures are one of the more popular solutions for Web delivery.[11] These special complex table systems allow an archivist to list the data from each of the XML finding aids one after another, as one would enter multiple line-entries in a flat spreadsheet like Excel, yet still keep track of all of the complex hierarchies and relationship groupings. The most common of these table-management systems that lets an archivist list multiple XML documents-worth of information inside a single traditional table-structure is called MySQL. “My” is an adornment, but SQL means “Structured Query Language.” It is called “query language” because it allows for lots of advanced search capabilities by standardizing, or structuring, the layers of hierarchy inside of which unique data

[10] Pitti and Duff, 3.

[11] Xiaomu Zhou, “Examining Search Functions of EAD Finding Aids Web Sites,” *Journal of Archival Organization* 4, n. 3/4 (2006): 106 (table).

are described. By using a MySQL table to store the data, all kinds of programs, not just those used in the library-archives industry, can reach in, interpret data relationships, and pull out whatever pieces of the data they desire to display or use at the time.

Administrative software designed to input and extract data to and from these hierarchical spreadsheets, or MySQL-managed tables, allows archivists to manipulate data using customized interfaces. For example, one administrative software component might be fill-in-the-blank forms and menu selections for new collection data entry, rather than requiring raw-encoded EAD. An early example of this would be the University of Illinois's Archon (Archives-Online) software-development project co-authored by Chris Prom and Scott Schwartz. Another emerging example is the Archivists' Toolkit Project, an ongoing project of the University of California San Diego Libraries, New York University Libraries, and Five Colleges, Inc. Libraries. Archivists enter collection information into programs like Archon and Archivists' Toolkit using fill-in-the-blank online form interfaces. Ideally, the software takes the information out of the forms, stores it in tables, and then uses it to create as many formats as desired, such as an online finding aid that can be displayed in a standardized EAD tag-code, or even a MARC record draft.[12] If any of the early examples of this kind of administrative software system were to become fully functional, it would no longer be essential for an archivist to be able to encode raw EAD or program and customize a delivery system in order to display EAD XML documents, though he might still choose to do so, working from raw output options. With that in mind, some archivists are already making the move to focusing now on user studies and home-grown programming to help archives collaborate to develop non-commercial, local delivery systems that utilize these untapped functionalities.[13]

[12] Chris Prom and Scott Schwartz, Archon Web site, <http://www.archon.org/> (accessed October 15, 2007); University of California San Diego Libraries, New York University Libraries and Five Colleges, Inc. Libraries, Archivists' Toolkit Project Web site <http://archiviststoolkit.org/> (accessed October 28, 2008).

[13] Zhou, "Examining Search Functions," 103.

Though it is possible that these local systems might one day compete and eventually merge into one world-system, for the moment the "union access" proposal simply becomes more and more encumbered as each institution or region strikes out on its own.

Software programmers generally attempt to write collection-administration programs so broad and open as to accommodate multi-institutions' local container-level structurings. That way the software can be marketed and sold broadly. The software, once installed, however, requires that the local institution hire its own programmer to "finish off" and customize the functionality so that it will accommodate the locally chosen hierarchical structures for the container-list, and the end result is that inevitably the software becomes locally distinct again, incompatible with other offshoots of the same original marketed package. Because many archives are still trading individual data sets between these systems using EAD documents as the "Esperanto" of the digital finding-aid lexicon, it might be efficient to consider that some further standardization of the underlying hierarchical structure of EAD, even within single institutions, would simplify the process of delivery-system development and EAD markup, to the benefit of many.

PROBLEM

Structured database software systems like Archon, Archivists' Toolkit, and other homegrown local and regional systems which import or read EAD-structured XML documents can be programmed to import collection-level data from other managed databases with relatively few problems. A moderately experienced programmer can steer the collection-level fields from one EAD XML-generating program into any other, writing a script with instructions that allow the computer to carry out the transfer automatically. However, when it comes to the container-level data, much of this potential for clean exchange falls apart. It is rare and perhaps unheard of for one archive's local EAD-compatible administration-software platform to trade container-level data smoothly with another's, or for a program that searches multiple institutions' data with any search method other than full-text keyword searching to read and negotiate in a fully functional manner among all of what are fundamentally dissonant EAD container-level management systems.

When an archivist makes the decision to start entering finding aids into a table-driven piece of software instead of hand-coding them, he or she faces several hurdles. If previous archivists have already implemented one of many arbitrary systems for hand-coding EAD documents one-by-one, it is unlikely that the box lists will upload correctly into any new commercial collection-management program. The collection-level data will fare better, generally, but collection-level data are usually just a few pages long at most, whereas container-level data may go on for thirty or forty pages. With that in mind, the archivists who previously have been hand-coding EAD documents for the institution will, quite understandably, want to stick with their current non-database-structured process. If they are in compliance with EAD display standards, they will see no advantage to re-coding or migrating hundreds or thousands of lines of data, just so that it can be uploaded and stored in a particular piece of software that allows for the same sort of controlled searching, particularly if that software, unlike the perceived-EAD, is not standard to all institutions. But again, though by hand-coding they are complying with allowable structures of EAD, all they may have accomplished in hand-coding the hundreds of finding aids is little more than if they had coded them in HTML so far as compatibility with other institutions and software goes. Yet compatibility was a primary purpose for EAD and all of the recent collection-administrative software. Looking ahead a little, even if the legacy finding aids must remain unchanged, surely at least the newly digitized finding aids could be brought into compliance with some agreed-upon standard.

Many institutions that produce articles and have sought a voice in EAD development naturally also have a large legacy of encoded finding aids. On the other hand, many of the institutions concerned with reading the literature and using the standards may not yet have implemented EAD, or may have been hand-coding a very small, limited set of finding aids. Some archives are still trying to evaluate their first software solutions. As Zhou points out, “Although a variety of archival institutions are considering joining the EAD community, it is primarily college and university archives and special collections that have adopted

EAD to encode their finding aids."[14] It would seem, therefore, still useful to establish a current recommendation for optimal EAD-encoding structure down to the container list, such that any unencumbered institution could be invited to adhere, if interested in achieving the most seamless EAD field-mapping for exchange of finding aids with future peer institutions and administration software platforms, realizing the fullest potential of having a specialized XML DTD. If an institution chooses not to follow the optimal-structure recommendation, they could, of course, still code a document in an acceptable, locally administrated form of EAD that would function as a freestanding document on the World Wide Web, even if the container-level data would not be available for interchange between institutions. But this is not an optimal level of cooperation for an academic and professional field in the digital age. Working together, as with the collection-level data, it would seem possible for archivists to unite and determine an optimal, software-interpretable, generalizable skeleton upon which to model new container lists.

The most frequent explanations given for the lack of standardization at the container level are usually one or both of these two arguments:

1. Archival collections are unique; and
2. We cannot relabel boxes, so physical order has to trump intellectual coherence in the digital realm.

These arguments are based in part on a lack of understanding of the term "standardization" in the context of information technology. Standardization in a searchable database is an attempt to define what is new or unique about an element by building on what is known and non-unique about it. Take library cataloging for an example. Library of Congress subject headings form a standardized lexicon which effectively serves two purposes:

1. It provides an established vocabulary for describing materials in consistent manner across institutions; and
2. It demonstrates by rules and by consistency the manner by which further unique words may be added to that vocabulary.

[14] Ibid., 100.

The cataloger places the new, unique heading in a meaningful non-unique position within the existing body of vocabulary so that others can understand it, as well as locate it for later applications. The system of using headings and the process for creating new headings are standardized, whereas the headings themselves remain unique.

The second argument is a symptom of under-utilization of information technology, whether EAD or spreadsheet table-based collection-administrative software programs in general. It is possible to represent illogical physical orderings with very logical Web-accessible intellectual descriptive documentation. Historically, users have not "browsed" archival shelves, and boxes from a single collection have not had to sit next to each other on the shelf. Now, however, it has become possible to create virtual, browseable electronic shelves by presenting a falsely organized view of a collection that can quite easily refer back to a disordered physical reality. EAD and collection-administrative programs can impose some useful regulation on this wide-open descriptive situation so that researchers, as well as archivists, can make informed assumptions about where to look electronically for descriptive data even if the physical arrangement of the materials is unique. Many of the scenarios that archivists think of as being a part of the "uniqueness" of collections are actually the result of physical happenstance, and are furthermore quite commonplace among repositories, even though they may disobey the current descriptive practices. For instance:

- A series extends over three boxes with nonconsecutive numbers.
- A series ends mid-box and another begins.
- A new box needs to be inserted between two old boxes intellectually, even though its box number is much higher.
- The collection is too small for series, but there are five distinct intellectual themes inside each of the two boxes.

Collections may sometimes be old, and they may have been processed before certain descriptive practices were put in place, or perhaps the current descriptive practice seems unclear. EAD, for its part, allows for a plethora of solutions, without making it clear which one will result in the most frequently applied structure for each case. If archivists could agree upon a standardized, optimal hierarchical container-level shell schema for newly encoded finding aids that directed structuring of these common scenarios, then even if archives keep legacy templates intact, looking to a more collective future, it might enable commercial programmers to create programs with higher delivery functions for a larger, more viable customer base, rather than having to spend their energies creating one-by-one compatibility patches for isolated customer systems. One common illustration of a container-level element that has eluded much-needed standardization is the concept of the box.

Hierarchically, in an XML document, depending on one's local system setup, a box tag might not be able to be opened and closed as a subcomponent within a series because it might also contain folders of another series. Concurrently, a single box may, in some institutions, be listed in a single EAD document twice, but it risks confusing those other institutions' brands of EAD administrative software that either disallow repetition, or interpret it as an order to overwrite on import. Within a single institution, some of the finding aids for collections treat boxes as intellectual sub-sub-series bearing scope notes and dates, and others treat boxes as strictly physical locations whereas folders bear scope notes and dates. Sometimes within a single finding aid it is possible to find examples of both intellectual and strictly physical treatments of "box." In the context of prose and individual free-standing EAD documents, such variety is permissible. For a programmer or a database, each of these forks in the road of local treatment requires an entirely separate customized programming path and an increasingly sophisticated understanding on the part of a non-cognitive machine in order to carry out each small function across institutions.

According to the creators of EAD, "It was agreed that the intellectual arrangement of the archival materials was more important and more permanent than the physical order, and the DTD was designed accordingly."[15] It may be impossible to settle on one single standardized physical structure that would meet all collection-descriptive needs. But on the other hand, it might be possible for intellectual structure to ascend still further and form a more restrictive, standardized tag structure for marked-

[15] Ruth, *Encoded Archival Description*, 315.

up EAD container lists. If physical elements could be exclusively relegated to serving an attribute-function within intellectual structure, it might in fact grant archivists more freedom of physical description without disrupting software-compatible container-level arrangements.

For optimal software and peer compatibility, tag hierarchy must be consistent, even if attributes are flexible. On a family tree, for instance, the grandmother must always be the mother's mother—she cannot sometimes be the sister of the grandchild, but she still may have any physical attributes she likes. For optimal software-compatibility, EAD XML structure could prohibit physical containers, such as a box, from bearing any intellectual sub-elements such as titles and dates. Any physical item such as a box or folder entered in EAD could be required to have some level of intellectual structure surmounting and anchoring it, from which it would consistently inherit its descriptive traits.

In XML markup terms, this would mean something like displacing all of the <container> tags and attributes and assigning them as attributes within intellectual tags such as the <c> tags. The "box" might not sometimes be hierarchically above a series and at other times below it, but rather always above. Alternately, in order for "container" to be used as a hierarchical indicator within EAD tag structure, it could be made to suffer a concrete hierarchical boundary. All of the optional container attributes, like "type," would need to be physical descriptions that corresponded to the hierarchical station of that box or its sub-elements. Some elements of physical structure in a finding aid happen to sync up consistently with intellectual structure. One such element is "folder," or "file." No two series or subseries need ever be housed within a single folder in any archive. For that reason, "file" is clearly always arranged hierarchically below the series and subseries, never above. "File" is thus already hierarchically stable as a part of the intellectual <c> tag structure, and the <container> tag's attribute-destination "folder" should conceivably be able to cede to "file." "Folder" is consistently intellectual, as well as consistently physical, whereas "box" is only consistently, reliably physical.

ILLUSTRATIONS

For those already using XML, or for those planning to design customized collection administrative software, one of the best ways to explicate this type of suggestion is through the use of illustrations. As explained in section 7.2.5 of the EAD Application Guidelines, version 1.0, one XML tag can only inherit an attribute from another if it falls within the family of that tag, after the opening parent-tag and before the closing one.[16] Similarly, in a normal XML structure designed for an archive, if there were a series that consisted mostly of boxes, an XML document could assign the default container-type "box" at the series level. This is not to say that the series would be one box, but rather that the attribute "container," if used by any tag within this series would always be of the type "box." All the tags that were listed under the jurisdiction of that series, if they invoked the attribute "container" by assigning a container number, would inherit the container-type "box" attribute, without having to say so each time, unless another were specified locally to override it.

If an archivist had a software program for administrating collection data, he could input a complex legacy container list such as the one below, exactly in the progression it is written here:

```
Series 1: Correspondence, packaged awards, and standing volume
    Box 1
        Folder 44 — Correspondence with Jim and Ralph, 1920-1940
            Item 1 — Letter from Jim
            Item 2 — Letter from Ralph
    Box 2
        Folder 1 — Correspondence, 2004-2006
    Package 1
        Item 1 — Framed Award
        Item 2 — Framed Award
    Item 1 (a free-standing unboxed item) — Book
```

Behind the scenes, meanwhile, the administration software program could, among other things, format this list into

[16] Society of American Archivists, Encoded Archival Description Working Group, *Encoded Archival Description Application Guidelines: Version 1.0* (Chicago: Society of American Archivists, 1999), 200-203.

software and database-friendly, consistently hierarchical XML code similar to that shown in **Example A**:

```
<c01 level=“series” container-type=“box”>1
    <did>
    <unittitle >
         Correspondence, packaged awards, and standing volume
    </unittitle>
    <c02 level=“file” container=1>44
         <did>
             <unittitle>
             Correspondence with Jim and Ralph
                  <unitdate type=“inclusive”>
                  1920-1940
                  </unitdate>
             </unittitle>

             <c03 level=“item”>1
                  <did>
                      <unittitle >Letter from Jim
                      </unittitle>
                  </did>
             </c03>

             <c03 level=“item”>2
                  <did>
                      <unittitle > Letter from Ralph
                      </unittitle>
                  </did>
             </c03>
         </did>
    </c02>

    <c02 level=“file” container=2>1
         <did>
             <unittitle>
                  Correspondence
                  <unitdate type=“inclusive”>
                  2004-2006
                  </unitdate>
             </unittitle>
         </did>
    </c02>

    <c02 level=“item” container-type=“package” container=1>1
         <did>
             <unittitle>
             Framed award
             </unittitle>
```

```
        </did>
    </c02>
    <c02 level="item" container-type="package" container=1>2
        <did>
            <unittitle>
            Framed award
            </unittitle>

        </did>
    </c02>
    <c02 level="item">3
        <did>
            <unittitle>
            Book
            </unittitle>

        </did>
    </c02>
    </did>
</c01>
```

If a series were composed of two boxes and each box held a different kind of content that required titling, rather than assigning titles to the boxes themselves in XML, the archivist would need to impose an extra level of "subseries" structure within the code (not on the box-labels of the actual boxes—just electronically within EAD) using unnumbered subseries. Unnumbered <c> tags might, for example, always indicate that a level existed only in XML hierarchical structure, not in the physical world. The two unnumbered subseries could be assigned the container-type "box" and a container number (box number) which would indicate the existence of a physical box. As before, one might also here assign the container-type "box" at the series level, so that it could be left out of all the subsequent "subseries" level tags that fell hierarchically within the parent series.

The archivist would enter the collection into an administrative software database in the following structural order:

```
Series 1
    Subseries (unnumbered) — Correspondence with Mr. Smith,
            1940-1943
        Description: This subseries contains correspondence with
```

```
          Mr. Smith.
      Box 34
          Folder 1 — Letters about floorboards
          Folder 2 — Letters about curtains
  Subseries (unnumbered) — Correspondence with Mr. Jones,
          1940-1942
      Description: This subseries contains correspondence with
          Mr. Jones.
      Box 35
          Folder 1 — Letters about light fixtures
          Folder 2 — Letters about carpeting
```

The software would then generate roughly the XML code of **Example B**:

```
<c01 level="series"; container-type="box">1
    <did>
    <c02 level="subseries"; container=34>
        <did>
            <unittitle >
            Correspondence with Mr. Smith
                <unitdate type="inclusive">
                1940-1943
                </unitdate>
            </unittitle>
            <scopecontent>
            This subseries contains correspondence with Mr. Smith
            </scopecontent>

            <c03 level="file">1
                <did>
                <unittitle> Letters about floorboards
                </unittitle>
                </did>
            </c03>
            <c03 level="file">2
                <did>
                <unittitle> Letters about curtains
                </unittitle>
                </did>
            </c03>
        </did>

    </c02>
    <c02 level="subseries"; container=35>
        <did>
            <unittitle >
            Correspondence with Mr. Jones
```

```
                <unitdate type="inclusive">
                        1940-1942
                        </unitdate>
                </unittitle>
                <scopecontent>
                This subseries contains correspondence with  Mr. Jones
                </scopecontent>
                <c03 level="file">1
                    <did>
                    <unittitle> Letters about light fixtures
                    </unittitle>
                    </did>
                </c03>
                <c03 level="file">2
                    <did>
                    <unittitle>Letters about carpeting
                    </unittitle>
                    </did>
                </c03>

            </did>
        </c02>
        </did>
</c01>
```

If a collection were too small traditionally to have had series, and was, for example, housed within a single box, one would, for the sake of optimal XML software-usable structure, impose an unnumbered (again, electronic-only) series upon the entire collection, assign a container type "box" and box number to indicate an actual physical box within that series, continuing by adding all of the files within it. Administrative software data entry would be something like the following, where the unnumbered series bears the descriptive data that would have belonged to the box:

```
Series (unnumbered) — Collection of correspondence with everyone,
    1920-1963
    Box 1
        Folder 1 — Letters about floorboards
        Folder 2 — Letters about light fixtures
        Folder 3 — Letters about rats
```

XML output would look similar to **Example C**:

```
<c01 level="series"; container-type="box"; container=1>
      <did>
      <unittitle>Collection of correspondence with everybody
            <unitdate type="inclusive">1920-1963</unitdate>
      <unittitle>
      <scopecontent>This series contains correspondence with Misters
            Yardley, Smith, and Jones
      </scopecontent>
      <c02 level="file"> 1
            <unittitle>Letters about floorboards
            </unittitle>
      </c02>
      <c02 level="file">2
            <did>
            <unittitle>Letters about light fixtures
            </unittitle>
            </did>
      </c02>
      <c02 level="file">3
            <did>
            <unittitle>Letters about rats
            </unittitle>
            </did>
      </c02>
      </did>
</c01>
```

If parts of a single series appeared in multiple boxes that also contained parts of other series, the container attribute's destination number (the box number) could be repeated as an attribute within multiple file-level or other series-level tags, and software programmers would need to know that they should consistently treat multiple-mention of any container number as an "add-to" command rather than an "overwrite" command or a data entry error. Data entry example:

```
Series 1
      Subseries (unnumbered) — Correspondence with Mr. Smith
            Box 2
                  Folder 30 — Letters about floorboards
                  Folder 31 — Letters about curtains
```

```
Series 2
    Subseries (unnumbered) — Correspondence with Mr. Jones
        Box 2
            Folder 32 — Letters about light fixtures
            Folder 33 — Letters about carpeting
Series 3
    Subseries (unnumbered) — Correspondence with Mr. Yardley
        Box 3
            Folder 1 — Letters about rats
```

The XML output might look something like **Example D**:

```
<c01 level="series"; container-type="box">1
    <did>
    <c02 level="subseries"; container=2>
        <did>
            <unittitle >
            Correspondence with Mr. Smith
            </unittitle>

            <c03 level="file">30
                <did>
                <unittitle> Letters about floorboards
                </unittitle>
                </did>
            </c03>
            <c03 level="file">31
                <did>
                <unittitle> Letters about curtains
                </unittitle>
                </did>
            </c03>
        </did>

    </c02>
    </did>
</c01>
<c01 level="series"; container-type="box">2
    <did>
    <c02 level="subseries"; container=2>
        <did>
            <unittitle >
            Correspondence with Mr. Jones
            </unittitle>

            <c03 level="file">32
                <did>
                <unittitle> Letters about light fixtures
```

```
                    </unittitle>
                    </did>
               </c03>
               <c03 level="file">33
                    <did>
                    <unittitle>Letters about carpeting
                    </unittitle>
                    </did>
               </c03>

          </did>
     </c02>
     </did>
</c01>
<c01 level="series"; container-type="box">3
     <c02 level="subseries"; container=3>
          <did>
               <unittitle >
               Correspondence with Mr. Yardley
               </unittitle>

               <c03 level="file">1
                    <did>
                    <unittitle> Letters about rats
                    </unittitle>
                    </did>
               </c03>
          </did>
     </c02>
     </did>
</c01>
```

CONCLUSION

EAD in its current version requires that archivists impose one of many possible intellectual structures upon a box list, and simply by applying one of any number of possible structures, EAD serves to enable advanced-search functionalities locally. EAD markup tags can serve as markers/anchors for local programs and search engines, regardless of where they are or how they are arranged at a single institution. However, without consistency across collections, it is difficult to find administrative software that can work for all the disparately structured EAD documents. The problem is compounded when archivists try to create cooperative finding-aid databases across institutions. If a functional solution could lead to the standardized treatment of the container list across archives, then that alone might greatly reduce the amount of time programming-code software designers must currently invest in composing compatible import and export protocols. An optimal standard for software consciously structuring EAD container-level data as a whole would be an asset for both collection-administration system programmers and archivists at institutions who just want to know "the best" software solution for managing and encoding the finding aids for the Web. The axiom of Occam's Razor, that "the simplest solution is probably the best one," when it is used as a limit on creativity and exploration, is probably disputed for good reason in many scenarios, but once the rules of a solution are fully explored and understood, simplicity has its structural benefits. An optimized standard may not preclude the usefulness of other local or legacy solutions, yet it is certainly at least an asset that archivists might want to have in-pocket, for application where there is a choice.

Leah Broaddus is the university archivist at Southern Illinois University Carbondale's Morris Library Special Collections Research Center. She is a graduate of the MLS special collections program at Indiana University Bloomington and has a background in Perl programming and library instruction. With the assistance of SIUC information services staff programmer Mickey Soltys, the mentorship of Special Collections director Pam Hackbart-Dean, and the support of University of Illinois archivists Chris Prom and Scott Schwartz, she recently led the implementation of online-finding-aid management at her institution. Research support was provided by SIUC Morris Library.