Document Formats for Archiving

James C. King, Leonard Rosenthol and Diana Helander; Adobe Systems Incorporated; San Jose, CA

Abstract

A continuing question that plagues the Archiving community is the choice of file format to use for archiving electronic documents. It is complicated by many factors including: the wide spectrum of archiving situations and specialized requirements, continuing evolution of existing document file formats, the invention of new kinds of documents and electronic content which must be archived, and the difficulty of planning to have all the appropriate hardware, software and media available in the future to process these documents. In this paper we focus primarily on the choice among existing document file formats and discuss their current suitability for various anticipated tasks.

Electronic Archiving Requirements

Before consideration of what file format is most appropriate for archiving we must first determine what objectives the archiving is intended to achieve and what kind of material we are being asked to archive. This seems rather simplistic but those considerations are often just assumed and one archivist may not realize that another archivist has different objectives. An important first question is: what kind of document is to be archived. For traditional documents there are two general categories of electronic documents.

ePaper Documents

For example, if the task is to preserve existing paper documents, unchanged but in electronic form, then some form of what we will call "electronic paper" or ePaper is called for. We expect it to simulate paper electronically, maintain appearance fidelity and not be easily changed.

This would include documents that are documents of record that preserve public documents (e.g., birth certificates, wedding licenses), business transactions (e.g., invoices, agreements), legal decisions (e.g., court records), or laws (e.g., records from legislative bodies). Generally the objective when archiving these ePaper documents is to preserve them for future reference much as we preserve paper documents emphasizing visual fidelity. They may have been derived by scanning paper documents. Providing the ability to edit and reuse these documents is *not* a requirement.

Editable Documents

On the other hand, if the need is to maintain some master document that can be edited to produce derivative documents and to do this over long periods of time, then a document format primarily designed to preserve the edit-ability of the contents is needed. If preserving the exact appearance of the master must be sacrificed in order to provide edit-ability then so be it. Since the master will be used to produce different derivative documents in the future, their appearance can be tailored to the needs at that time.

Other Document Types

Although we are going to focus primarily on more traditional document file formats, it is worth mentioning some of the other document types not necessarily aligned with traditional documents.

Live Documents

The most difficult category of document to preserve is the newer interactive and dynamic document types. As we try to make use of the ever more powerful computers and displays available to us, we strive to invent new document metaphors that move away from our more static paper-based ideas. The challenge in archiving these documents is two-fold: a wide variety of new document types which quickly may become obsolete as technology matures, and the technologies these documents depend upon are also still maturing, with the result that some of the base technologies may become obsolete.

Live documents include games and interactive web applications. One obvious archiving objective for this class of document is to preserve the ability to repeat the current experience at some future time. This is a very perplexing type of content since its value is in its active nature and that depends upon being able to provide a suitable "execution environment" in the future. Virtual machines that can simulate today's computer systems may be the best answer for experiencing these documents in the future.

Pictures

Pictures, usually referred to as images when in electronic form, also offer yet another kind of artifact that has its own unique requirements and objectives for archiving. Since images tend to consume large amounts of electronic storage space, compression techniques are a primary focus area for archiving. Color images compress much more if lossy compression is allowed but the debate continues about using lossy techniques. The best lossy color compression remove image features that are impossible for humans to physically see under normal viewing conditions. So, again, if we know the archiving requirements and they are confined to normal human viewing of the archived pictures then careful application of lossy color compression is acceptable.

Scanned Documents

Since most of our document archiving and preservation activities in the past have been centered upon paper documents, it is natural to want to convert some of our existing paper into electronic representations for electronic archiving. Scanning the paper pages into computer images is usually the starting point. TIFF [1] is commonly used for this. However, TIFF supports a suite of choices for image representation with a wide variety of technologies including RGB and CMYK color representations and JPEG [2], LZW [3], fax [4] and other compression technologies. TIFF is often used with no compression, based on a belief that the data may be more easily retrieved and not accidentally corrupted than when using a lossless compression technique. The price for this is larger files requiring more archive space. There is also some risk in using TIFF for archiving, as it is a proprietary standard and not an open one. In addition, it has been incompatibly fragmented since the last official publication (TIFF Version 6 in 1992).

Scanning page images is valued for being as close to saving the paper itself in electronic form but it also inherits a lack of more sophisticated electronic capabilities like content searching.

For those archivists who prefer to use open standards, scanned documents should be saved as PDF/A [6] for which several compression choices are provided including: none, fax, flate (PNG [7]), JBIG2 [8] and JPEG2000 [9]. There are also products that will do optical character recognition and document recognition on scanned images while turning them into non-image PDF/A files to which word searching may be applied [10].

Document Binding Stages

Our attention will now be focused more on compound documents and their binding to particular output devices. There is a spectrum of document formats determined by how tightly the contents of the document have been "bound" for presentation on a particular display or onto paper. At the right end of the spectrum shown in Figure 1, at binding stage 4, is the actual display or printed page that is typically produced from a device resolution bitmap as shown. This is the most tightly bound form for a document.

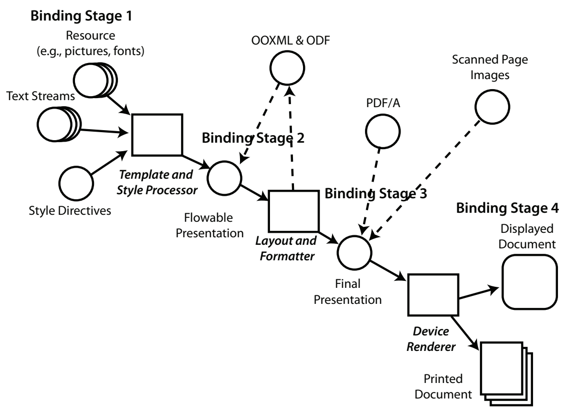

Figure 1. Document Binding Stages

At the other least bound end of the spectrum, at binding stage 1 in Figure 1, we have the raw materials from which a document is composed consisting of one or more streams of text, pictures, graphics and other resources such as fonts and color spaces. In between we have identified document representations that represent interesting points along the "binding" spectrum at binding stages 2 and 3 in Figure 1.

One way the spectrum can be explored is to imagine creating a document as we move its representation from least bound (stage 1) to most bound (stage 4). We begin with the raw material from which a compound document will be created including text, pictures, graphics, required resources and styling directives at stage 1. We combine all this raw material into a single document that represents a flowable presentation document at binding stage 2. The raw items could be used for some purpose other than a presentation, but in moving to stage 2 we begin to bind the material together with a presentation as the ultimate target. At this stage the exact layout has not been determined but font choices and placement constraints have been introduced. We have a flowable document marked up for presentation. We have not yet bound the information to a presentation size and shape. XHTML, ODF and OOXML are examples of stage 2 document formats.

To arrive at binding stage 3 we actually perform the layout and formatting determining exact placements, line breaks, hyphenations, pagination, etc. PDF/A is a good example of a format in which to capture binding stage 3. The document has not been committed to a particular screen or page but the general size and layout characteristics have been totally bound.

And finally the document can be further bound to a bitmap destined to drive a particular display or to be printed by a particular printer on particular paper at binding stage 4. Here we are safely assuming that nearly all of today's output devices are driven by bitmaps.

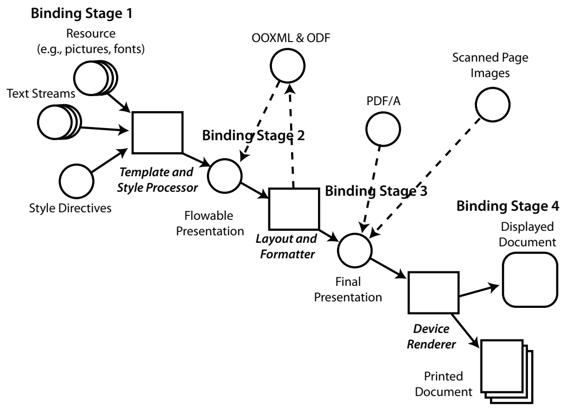

Figure 2. Positioning OOXML, ODF, PDF/A, and scans

In Figure 2 we have introduced some common document formats and related them to the binding stages. OOXML [11] and ODF [12] are examples of binding stage 2, PDF/A is an example of binding stage 3, and scanned images are a special case also at binding stage 3.

Scanned pages result in scan resolution bitmaps which are not tightly bound to a particular device since they may not match the device's resolution or bitmap characteristics (number of bits per pixel, number of color channels per pixel, if any, etc.). A further binding via the device rendering process is necessary to create a final presentation from this scanned material.

Note that for OOXML and ODF there is an arrow going from the layout/formatter to those files types. This is because they also serve as the native file format for the authoring applications, saving key information that can be restored to continue creating or editing the documents after an interruption.

Note also that PDF/A is not a stage 2 representation but is a stage 3 format that is more tightly bound to the final output. PDF/A-1a does support including enough stage 2 information so that searching and reading aloud to the blind can be accomplished. PDF/A-1b is strictly stage 3.

Exploring the Binding Spectrum

Moving from less bound to more bound along the binding spectrum involves removing degrees of freedom from the representation. For example, at stage 2 the styling information may bind the document to two column formatting. All points to the right must conform to that choice. In order to reverse that choice we have to return to a less bound point on the spectrum. Tighter binding also may lose information no longer essential for just presenting the content. For example, at stage 2 we may know that a text string is a "heading" but at the more bound stage 3 that information is no longer needed, since it already has been used to bind the string to a large font and a particular page position.

Information may be dropped as the document moves to more bound representations, which means that it may be difficult to move a document in the opposite direction, from more bound to less bound. For example, one can only guess that certain spacing and font choices mean that a text string is a heading. There could have been other high level semantics other than that of "heading" that determined the same font and positioning.

This is an important observation. Documents scanned into image formats like TIFF fall to the more bound end of the spectrum and it is difficult to move them toward the less bound end because we have to deduce information not readily available in the image representation. For example, optical character recognition makes informed guesses that certain pixel clusters in the image data are really letters of the alphabet. A level of uncertainty accompanies those determinations. Going further toward stage 1 involves guessing which strings form paragraphs, which strings are headings, which numbers are page numbers and so on.

When starting with scanned pages one must determine if simply being able to present the highly bound pages for viewing or printing is sufficient or if other requirements will dictate that we do optical character recognition or document recognition.

File Size

Each of the document representations has a cost in terms of file storage size. For example, full page scans are usually much larger than more compact "coded" representations using text strings and graphics commands to describe the pages (e.g., PDF/A). The most efficient representations with respect to size seem to lie in the middle of the spectrum where enough is known about the document contents to compress or otherwise reduce the size but not so bound that it is using an image representation. At the most unbound end of the spectrum we may have more information than is actually needed to produce the required presentation.

Of course, the most flexible and reusable document representations are those that are least bound. This is almost a tautology. The less bound the material is, the more binding choices remain to be made and hence the more flexible that representation is. But if a document owner or author wants to control the presentation of a document then we need to remove the flexibility and bind the document to reflect the author's wishes.

An archivist needs to determine the correct trade-off between flexibility and authentic appearance and choose one or more document formats accordingly.

Augmenting Stage 3 Documents

There is a file format attribute that runs orthogonal to the binding axis and that is the search-ability of the document content. We may have no requirements for editing or modifying the documents but we may still need to see if given words, phrases or concepts can be found in the text of the document. We may also want to search non-textual material although the techniques for that are still under development.

Text streams (e.g., text articles) are the key component of documents needing to be searched. Special efforts are made in PDF/A-1a documents to retain or re-introduce the higher level information to be able to extract the text streams and search them. This is also necessary for reading aloud for the blind. So in this case we have a hybrid representation that is primarily tightly bound but contains additional information allowing reconstruction of a less bound form for parts of the document. A conforming PDF/A-1a file is required to have this additional information. Sometimes this kind of extra information is difficult or impossible to obtain, and hence the need for the more relaxed PDF/A-1b.

In the other direction, given that we have enough styling and other formatting constraints in a loosely bound representation, it may be only a matter of computer power to turn that representation quickly into a bound displayable or printable representation. The trick that seems not to have been accomplished adequately to date, is to be able to carry out this formatting to produce 100% consistent output from user to user, computer system to computer system, from product revision to product revision and to do it very quickly even for large documents. In many cases perfect instantaneous reproduction across display environments is an essential requirement.

A Closer Look

We want to examine the less bound end of the spectrum in a little more detail because it may expose some interesting and commonly held misconceptions. In Figure 1 stage 1 we indicate the use of typical file formats used to contain the raw document components in their most unbound form. These include common image formats like JPEG, PNG and TIFF, graphic formats like EPS [13] and WMF [14], and text stream formats. By text stream we mean the raw stream of words that make up an articles contents. Various XML [15] markup languages are effective for marking up these text streams with basic semantic information, like delineating paragraphs and identifying chapters, headings, and titles (e.g., DocBook [16], XHTML [17]). These are, at times, called "structured text streams."

Of course, each particular type of text stream may require its own special markup language. In our model, these markup languages do not contain styling information, font choices, or any other information that determines the *form* a presentation might take. We are embracing the classic separation of "form" and "content" and here we are talking strictly about *content*. The form gets added as we take steps to bind the document into a stage 2 representation.

These text streams are fed together with pictures, graphics, and resources, such as fonts and color spaces, into a template completion and styling step where additional styling information is supplied. The output of this stage is then a document that is bound to styling and formatting constraints although they have not been "executed" yet – stage 2. Typical of this level of binding are OOXML, ODF and XHTML as shown in Figure 2. They are sufficiently unbound that the presentation geometry may still be changed and a complete re-layout and re-formatting of the document is possible. We might consider this the best definition of a "revisable" document and the first point in our binding spectrum where all the required components are bound together with choices that define the presentation document. Before stage 2 we only have a collection of raw materials that have not yet been shaped into a document.

Stage 2 documents are input to a layout and formatting process that binds the document further to a class of devices, primarily based upon presentation size and shape, by performing pagination, layout, hyphenation and justification, figure placement and all those functions we have come to expect from InDesign, FrameMaker, Microsoft Word and Open Office; a stage 3 document is the result.

Since one may spend considerable time and effort working at this stage, the document formats are created such that the work can be saved in a file (e.g., OOXML, ODF, FM), that can subsequently be opened and read to restore the application to the point it was at when the file was saved. These can be characterized as revisable flowable document formats.

These formatting applications can also move the document along the binding axis by generating PDF [5], PDF/A, PostScript [18] or other similarly more-bound representation that captures all the layout, styling and formatting decisions into a concise document format.

Common Questions

There are a couple of typical questions that arise when choosing a document archiving format and we address these now since we have laid a foundation upon which they can be addressed.

ODF versus PDF/A

A common question is: should I use ODF or PDF/A to archive my documents? This is one of the easiest to answer because it depends upon whether one wants to do further editing and revision of the document (e.g., ODF) or whether one wants a relatively frozen presentation format (e.g., PDF/A). PDF/A will guarantee that everyone subsequently viewing or printing that document will obtain results identical to all others doing the same thing and they will all see what the original author intended and/or what the original paper document looked like. If one wants to archive documents that will be further edited or used as templates for future work then a revisable format is more advised (e.g., ODF). Both stage 2 and stage 3 documents can be searchable.

What about XML documents

Another common question is: shouldn't we use XML as our archiving format? In actual fact, that is not a very well formed question. Although the long name of XML is the Extensible Markup Language, we choose to think of it as a method for creating markup languages rather than a markup language itself. The term *markup* derives from the early publishing practice of marking up paper drafts with editing, styling and formatting comments to guide the creation of subsequent drafts. This moved into computer markup, primarily on text streams to delineate logical structure and semantic components within the text (e.g., paragraphs, emphasized text, chapter headings, etc.).

Using the XML notation we may create markup languages for a wide variety of uses. What often comes to mind for many people when one just says "XML" is an XML markup language for general documents that includes tags for paragraphs, list of various kinds, heading, chapters, foreword, preface, and other common document textual components (e.g., DocBook). However, the XML notation has been used to markup thousands of other kinds of documents ranging from business cards, invoices, health records, vector graphics, and database entries to programming languages. See [24] for thousands of examples.

Given the diversity of specialized markup languages, we encourage people to talk about XML ***for*** business cards, or XML ***for*** general documents, or XML ***for*** hypertext (XHTML) rather than just use the term "XML." It needs to be "XML ***for*** X." To say that one has created an XML Document is akin to saying you have written some text using the Roman alphabet. It is much more informative and useful to say that you have used an XML ***for*** technical reports analogous to saying that I have written a poem in the German language using the Roman alphabet.

Compound Documents

In our case we are most interested in XML used for some form of a general document. However, we are interested in compound documents that include pictures, graphics and other non-textual elements. The XML notation has a heavy bias toward marking up text, in fact, Unicode [19] encoded text. Only clumsy provisions are made for representing binary data or anything not easily or compactly coded using Unicode.

Another property of the XML notation is that it is a stream notation, making the most sense when read/written sequentially from start to end. Compound documents by their nature are composed from a variety of distinct components and it is useful to maintain their distinctiveness in the overall document format. In some situations it is also necessary to read one of the components independently from the whole document or independently from any of the other components. The nature of XML markup languages forces a processor to treat it more as one whole, perhaps reading everything into random access storage, usually into a Document Object Model or DOM, for more flexible accessing. In all cases in order to find any particular element in an XML marked up file one has to scan through the file looking for it.

ZIP Archive Packaging

Because of the limitations of XML markup languages, most of the compound document formats that use XML actually do so with the assistance of a file packaging strategy, typically ZIP archives [20]. Most of us are familiar with ZIP archives as a compression and packaging technology that can turn a directory structure, including any files therein, into a single file for more convenient transport or archiving. What is less well known is that we do not have to "unpack" the single ZIP file in order to read its component files. Software has been written that can read any given component individually from the ZIP file without reading any of the other components and especially without reading the whole archive. When writing, ZIP has the option to use lossless compression (e.g., Flate [21]) on the sub files so it has the additional benefit of making the overall document files smaller. The sub files are easily decompressed upon reading. And perhaps most importantly, the sub files within a ZIP archive can be in any encoding not just XML so binary data can be kept in its own efficient binary representation compressed or not. Sub files encoded in an XML markup language can be compressed to great advantage.

So the use of ZIP archives provide us with optional compression to make the overall file size smaller, independent random access to the components of the document, and the ability to store special and binary formatted material unchanged.

For these reasons OOXML, ODF and many other "XML" document types are actually *not* XML files but ZIP archive files. Reference to these formats as "XML Document" formats might be justified because the primary sub files within the archives are using XML markup languages but it *is* a misuse of terminology.

The XML Myths

As with many powerful and general technologies, we tend to attribute to the XML markup notation more positive attributes than it actually earns, and we tend to use it in cases that are much better served by other technologies. The marriage of XML markup languages with an encompassing file packaging technology (e.g., ZIP) is a good one and many, if not all, of the XML markup weaknesses can be compensated for when used in this way.

A reputation that XML markup languages enjoy is that they are easier to access and process and would likely be more accessible in the distant future – thus a good archiving technology. But we challenge the idea that XML files are more human readable and that it matters for archival purposes.

Since XML is really a notation and toolset for creating unique and specialized markup languages, the ability to understand any given markup language is dependent upon the complexity of the markup language not on the complexity of the XML notation. Consider Figure 3, which shows a fragment of an XSLT language program [22]. This is extremely difficult to decipher without an XSLT manual.

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
    <xsl:template match="/">
        <xsl:for-each select="./body/p">
            <p>
                <style>
                    bchar { font-size:"xxlarge";}
                </style>
                <xsl:value-of select="."/>
            </p>
        </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
```

Figure 3. Fragment of an XSLT "program"

Or consider the simple Scalable Vector Graphic (SVG) [23] file shown in Figure 4. Will these really be easier to understand in 50 years than say PDF/A? In 50 years will we be so limited that text base information will be easier to digest and understand?

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 12August
1999//EN" "http://www.w3.org/Graphics/SVG/SVG-
19990812.dtd">
<svg xml:space="preserve" width="612" height="792">
    <path style="fill-rule:nonzero;"
        d="M297,264V64.5H64.5V264H297"/>
    <path style="fill-rule:nonzero;fill:#FF0000;"
        d="M241.5,320.5c55.228,0,100-44.772,100-100c0-
        55.228-44.772-100-100-100c-55.228,0-100,44.772-
        100,100c0,55.228,44.772,100,100,100"/>
    <path style="fill-rule:nonzero;fill:#FFFF00;"
        d="M260.406,204.138l-
        97.894,175.38l232.207,0.087L260.406,204.138"/>
</svg>
```

Figure 4. Sample Scalable Vector Graphics (SVG) file

The complexity of the document format is a function of the complexity of the document not of the notation in which the contents and their relationships are recorded. Note: this SVG file defines a simple drawing of a square (in black), triangle (in yellow), and circle (in red) and overlapping.

Conclusion

One must first determine the use to which the archived documents will be put in the future, the intention of the archiving activity. Then one can chose one or more file formats for archiving that class of documents. In particular, a choice between preserving the exact appearance and preserving the full edit-ability of our document determines whether to use ODF or PDF/A.

We should always consider the particular XML markup language being used. XML, itself, is really just notations, rules and tools for defining text markup languages and the complexity for the files is in the complexity of the markup language not in the complexity of XML. Just as the Roman alphabet is relatively simple to understand it is quite different from being able to read and understand a poem in the German Language written using the Roman alphabet.

Recent trends to team up XML markup languages with a file packaging technology like ZIP archives seem to be able to make the best use of XML markup languages and complement their weaknesses.

It is also important to remember that the format alone does not make the archive. There are so many other considerations that go into a well-defined archival process. The document file format is just one piece of an archiving strategy.

References

[1] TIFF Revision 6.0, Final, Adobe Systems Incorporated, 1992.
[2] JPEG, ISO/IEC 10918-1:1994, Digital Compression and Coding of Continuous-Tone Still Images, 1994.
[3] LZW Compression, Lempel, Ziv, Welch compression, http://en.wikipedia.org/wiki/LZ77_and_LZ78.
[4] ITU-T (CCITT) T.6. Facsimile Coding Schemes and Coding Control Functions for Group 4 Facsimile Apparatus, 2005.
[5] PDF Reference, Version 1.7, – 5th ed., (ISBN 0-321-30474-8), Adobe Systems Incorporated (ISO 32000-1:2008).
[6] ISO 19005-1:2005, Document management -- Electronic document file format for long-term preservation -- Part 1: Use of PDF 1.4 (PDF/A-1), 2005.
[7] RFC 2083, PNG (Portable Network Graphics) Specification, Version 1.0, 1997.
[8] JBIG2, ISO/IEC 11544:1993/Cor 2:2001, Information technology—Coded representation of picture and audio information—Progressive bi-level image compression, 1993.
[9] ISO/IEC 15444-2:2004, Information Technology—JPEG 2000 Image Coding System: Extensions.
[10] Adobe Capture 3, Adobe Systems Incorporated, 2007.
[11] OOXML, Open Office XML, ISO/IEC 29500, 2008.
[12] ODF, ISO/IEC 26300:2006 Open Document Format for Office Applications, 2005.
[13] EPS, Encapsulated PostScript, Adobe Systems Incorporated, 1992.
[14] Windows Metafile Format (WMF), Microsoft Corporation, 2007.
[15] *Extensible Markup Language (XML) 1.1*, World Wide Web Consortium (W3C), 1998.
[16] DocBook, DocBook.org, http://www.docbook.org/, 2007.
[17] XHTML 1.0: The Extensible HyperText Markup Language, http://www.w3.org/TR/xhtml1/, 2000.
[18] PostScript Language Reference, Third Edition, Addison-Wesley, Reading, MA, 1999.
[19] The Unicode Standard, Version 4.0, Addison-Wesley, Boston, MA, Unicode Consortium, 2003.
[20] ZIP Archives, PKWare, http://www.pkware.com/documents/casestudies/APPNOTE.TXT, 2007.
[21] RFC 1951, DEFLATE Compressed Data Format Specification, Version 1.3, Internet Engineering Task Force (IETF), 1996.
[22] Extensible Stylesheet Language (XSL) 1.0, http://www.w3.org/TR/xsl/.
[23] Scalable Vector Graphics (SVG) 1.0 Specification, http://www.w3.org/TR/2001/REC-SVG-20010904/.
[24] XML Applications and Initiatives, http://xml.coverpages.org/xmlApplications.html.

Author Biographies

James C. King received his Ph.D. from Carnegie Mellon University (1969). Since then he has held three jobs, one with IBM Research at the T. J. Watson Research Center, the second with IBM at the Almaden Research Center and the third with Adobe Systems Incorporated where he is now a Senior Principal Scientist and PDF Architect. He is a Past President of the IS&T and a member of ACM and IEEE-CS.

Leonard Rosenthol serves as the Technical Standards Evangelist focusing on PDF standardization for Adobe Systems having been involved with PDF technology for more than 10 years. Prior to joining Adobe, Leonard worked as the Director of Software Development for Appligent, and the Chief Innovation Officer for Apago, while also running the successful consulting business of PDF Sages. Before becoming involved in PDF, Leonard was the Director of Advanced Technology for Aladdin Systems and responsible for the development of the StuffIt line of products.

Diana C. Helander is currently the Group Manager for Worldwide Standards for Adobe Systems. Her team represents Adobe in global standards organizations and industry associations responsible for setting standards, including PDF-based and XML data standards for government as well as the financial services, manufacturing, and life sciences industries. Ms. Helander was previously responsible for managing and growing Adobe Acrobat's presence in the manufacturing and AEC (building) markets. Ms. Helander graduated from Amherst College and is a member of the Board of Trustees of St. Timothy's School.