Ensuring the Longevity of Digital Information

by

Jeff Rothenberg

RAND
1700 Main Street
Santa Monica, CA 90407
310/393-0411
e-mail (Internet): jeff@rand.org

Revision: February 22, 1999

Digital documents are replacing paper in the most dramatic record-keeping revolution since the invention of printing. Is the current generation of these documents doomed to be lost forever?

Note: this paper is an expanded version of the article “Ensuring the Longevity of Digital Documents” that appeared in the January 1995 edition of *Scientific American* (Vol. 272, Number 1, pp. 42-7).

As of the above date, this revision could be found at http://www.clir.org/programs/otheractiv/ensuring.pdf

Ensuring the Longevity of Digital Information

by

Jeff Rothenberg

Revision: February 22, 1999

Note: this paper is an expanded version of the article “Ensuring the Longevity of Digital Documents” that appeared in the January 1995 edition of *Scientific American* (Vol. 272, Number 1, pp. 42-7).

The year is 2045, and my grandchildren (as yet unborn) are exploring the attic of my house (as yet unbought). They find a letter dated 1995 and a CD-ROM (compact disk). The letter claims that the disk contains a document that provides the key to obtaining my fortune (as yet unearned). My grandchildren are understandably excited, but they have never seen a CD before—except in old movies—and even if they can somehow find a suitable disk drive, how will they run the software necessary to interpret the information on the disk? How can they read my obsolete digital document?

This scenario questions the future of our computer-based digital documents, which are rapidly replacing their paper counterparts. It is widely accepted that information technology is revolutionizing our concepts of documents and records in an upheaval at least as great as the introduction of printing, if not of writing itself. The current generation of digital records therefore has unique historical significance; yet our digital documents are far more fragile than paper. In fact, the record of the entire present period of history is in jeopardy. The content and historical value of many governmental, organizational, legal, financial, and technical records, scientific databases, and personal documents may be irretrievably lost to future generations if we do not take steps to preserve them.

What have we already lost?

Although there are few well-documented, undisputed cases of important digital documents or data that have been irretrievably lost, anecdotal evidence abounds. One of the best publicized cases concerns U.S. Census information for 1960. This was originally stored on digital tapes that became obsolete faster than expected. Although some information on these tapes was apparently unreadable, most of it was successfully copied onto newer media, and it appears that nothing irreplaceable was lost (since the raw census returns were saved on microfilm). Nevertheless, this case represents a narrow escape and is cited prominently in the 1990 House of Representatives Report *Taking a byte out of history: the archival preservation of federal computer records* (Nov. 6, 1990, House Report 101-978). Additional cases of possible loss noted in that report include hundreds of reels of tape from the Department of Health and Human Services; files from the National Commission on Marijuana and Drug Abuse, the Public Land Law Review Commission, the President’s Commission on School Finance, and the National Commission on Consumer Finance; the Combat Area Casualty file containing POW and MIA information for the Vietnam war; herbicide information needed to analyze the impact of Agent Orange; and many others. Other sources suggest that scientific data is in similar jeopardy, as old NASA tapes and irreplaceable records from numerous experiments age ungracefully in the absence of funding to

copy them to newer media. These cases exemplify all of the modes of loss discussed in this paper: physical decay of media, loss of information about the format, encoding, or compression of files, obsolescence of hardware, and unavailability of software.

To date there appear to be few documented cases of unequivocal loss, but this may simply reflect the fact that documents or data that are recognized as important while they are still retrievable are the ones most likely to be preserved. The historical significance of many of our digital documents—which we may not consider important enough to warrant saving—may become apparent only long after they have become unreadable.

Old bit streams never die—they just become unreadable

My grandchildren's dilemma reveals some fundamental problems concerning digital storage. First, without the explanatory letter, they would have no reason to think that the disk in my attic was worth deciphering. Despite the much-touted immortality of digital information (stemming from its ability to be copied perfectly), it is the *letter* that will be immediately intelligible fifty years from now, not the digital disk. The letter possesses the enviable quality of being readable with no machinery, tools, or special knowledge—other than that of English, which it seems safe to assume will remain understandable for hundreds of years [see Figure 1].

Ironically, although its reproducibility make digital information theoretically invulnerable to the ravages of time, the physical media on which it is stored are far from eternal. If the optical CD in my attic were instead a magnetic disk, attempting to read it would probably be a waste of time. The contents of most digital media evaporate long before words written on high quality paper, and they often become unusably obsolete much sooner, as they are superseded by new media or incompatible formats [see Figure 2 and Photo 1]. The past few decades have witnessed the demise of numerous forms of digital storage. This has prompted my observation that digital information lasts forever—or five years, whichever comes first.

Yet neither the physical fragility of digital media nor their lemming-like tendency toward obsolescence comprise the worst of my grandchildren's problems. They must not only extract the content of the disk—they must also interpret it correctly. To understand what this entails, we must examine the nature of digital storage. Digital information can be stored on any physical medium that can record digits (such as the 0s and 1s that we call "bits"). Different media may store a given sequence of bits differently, according to the physical properties of the media and various conventions. We will use the term "bit stream" to mean an intended, meaningful sequence of bits (which may not be the same as the sequence in which they appear on some storage medium). A bit stream is simply a stream of binary digits, strung together in sequence [see Figure 3].

A bit stream can be stored in many different ways on different media. Retrieving a bit stream from its physical representation on some medium requires a hardware device, such as a disk drive, on which to "mount" that medium, as well as special "controller" circuitry that can retrieve the information stored on the medium—whether magnetic, optical, or other. A special program (called a "device driver") is also required to make this device accessible by a given computer system. Yet even assuming that my grandchildren still recognize digital information that is encoded in binary form and that the intended bit stream can be retrieved from the medium, it

As Shakespeare so eloquently notes in the couplet of the famous 18th sonnet, the printed word has a kind of immortality that few other things can claim. The word "this" in the final line refers to the sonnet itself, thereby proving its own point.

SHall I compare thee to a Summers day?
Thou art more lovely and more temperate:
Rough windes do shake the darling buds of Maie,
And Sommers lease hath all too short a date:
Sometime too hot the eye of heaven shines,
And often is his gold complexion dimm'd,
And every faire from faire some-time declines,
By chance, or natures changing course untrim'd:
But thy eternall Sommer shall not fade,
Nor loose possession of that faire thou ow'st,
Nor shall death brag thou wandr'st in his shade,
When in eternall lines to time thou grow'st,
So long as men can breath or eyes can see,
So long lives this, and this gives life to thee.

A modern, digital version of the couplet would have to be something like the following:

So long as the magnetic flux on this disk has not been disturbed,
and so long as humans retain the appropriate size and speed disk drives,
and so long as they have hardware controllers and software device drivers capable of reading the bits from this disk,
and so long as they have access to the software that encoded the file structure and character codes employed in the bit stream of this document,
and so long as they can still find or recreate the computing environment necessary to run that software,
and so long as they can still breathe or see,
So long lives this,...

Figure 1: Shakespeare's immortal Sonnet 18 and its digital equivalent

There is considerable controversy over the physical lifetimes of media: for example, some claim that tape will last for 200 years, whereas others report that it often fails in a year or two. However, physical lifetime is rarely the limiting factor, since at any given point in time, a particular format of a given medium can be expected to become obsolete within no more than 5 years.

Medium	practical physical lifetime	avg. time until obsolete
optical (CD)	5-59 years	5 years
digital tape	2-30 years	5 years
magnetic disk	5-10 years	5 years

Figure 2: The medium is a short-lived message

The centerpiece of this photo is a 1/3 scale replica of the Rosetta Stone. Discovered in Egypt in 1799 by a French military demolition squad, it contains equivalent renditions of a single text in three scripts. The first of these, hieroglyphic, had not been used since the 4th century AD, while the second, demotic, had last been used in the 5th century AD, making ancient Egyptian undecipherable for over 13 centuries. Since the third rendition is in Greek, the Rosetta Stone provided the key to interpreting the ancient Egyptian scripts. The original, which can be seen in the British Museum, dates from 196 BC and consists of a royal decree issued on the first anniversary of the coronation of Ptolemy V. (The rulers of Egypt during the Ptolemaic Period were Macedonian Greek, descended from Ptolemy, the son of one of Alexander's generals, which accounts for the use of Greek in this official document.) In addition to being quite legible after nearly 22 centuries, the Rosetta Stone's preservation is directly attributable to the fact that its import (i.e., that it consisted of three versions of the same text, one of which, being Greek, might provide the key to deciphering the lost Egyptian scripts) was visually apparent to the French lieutenant (Pierre Francois Xavier Bouchard) who was in charge of the squad that discovered the stone. The digital storage media shown surrounding the replica have already failed to remain readable for 1/100th as long as the Rosetta Stone.

Color photo by Jeff Rothenberg

Photo 1: The Rosetta Stone has far outlasted digital media

must still be interpreted. This is not straightforward, because a given bit stream can represent almost anything—from an integer to an array of dots in a pointillist-style image [see Figure 4]. How can future generations interpret our bit streams correctly?

Imagine that all the numbers in a monthly checking account statement were strung together with no punctuation or spacing to distinguish between check numbers, dates, or the dollar amounts of checks, deposits, or balances. The result would be a decimal "digit stream" containing all the important information in the statement—albeit in a decidedly unreadable form. In order to understand this stream, you would need to know its format, e.g., that it is a sequence of entries, each consisting of a number of pieces, such as a date, followed by a check number—with zero indicating a deposit—followed by a transaction amount, followed by an intermediate balance. You would also need to know where each piece starts and ends, i.e., how many digits comprise a date, a check number, and an amount. A bit stream is simply a digit stream in which each digit is binary (i.e., 0 or 1). Note that changing the length of the stream or rearranging it in any way wreaks havoc with its meaning.

date	chk/dep	amount	balance
4/5/94	deposit	$500.00	$500.00
4/26/94	chk# 314	$100.00	$400.00
4/27/94	deposit	$ 50.00	$450.00
11/3/94	chk# 315	$100.00	$350.00

Removing all spaces and punctuation and translating dates into 6 digits (mmddyy); check numbers into 4 digits; deposits into 0000; and amounts into 11-digits, the above entries become:

```
040594000000000500000000050000
042694031400000010000000040000
042794000000000050000000045000
110394031500000010000000035000
```

Concatenating these entries produces the following decimal digit stream:

```
040594000000000500000000050000042694031400000
010000000000400000427940000000000500000000450
001103940315000000100000000035000
```

A bit stream is simply a digit stream in which each digit is either 0 or 1.

Figure 3: What is a bit stream?

Compounding this problem, a bit stream has implicit structure that cannot be represented explicitly in the bit stream itself. For example, if a bit stream represents a sequence of alphabetic characters, it may consist of fixed-length chunks of information (called "bytes"), each of which represents a code for a single character [see Figure 5]. In current schemes, bytes are typically 7 or 8 bits long. But a bit stream cannot include enough information to describe how it should be interpreted. In order to extract fixed-length bytes from a bit stream (thereby "parsing" it into its constituent pieces) we must know the length of a byte. We could in principle encode a "key" integer at the beginning of the bit stream, representing the length of each byte [see Figure 6].

The 8 bits highlighted in the bit stream shown below can be interpreted in many ways, e.g., as an integer, a simple character code, a sound, a floating point number, an image, a logical bitmap, etc.

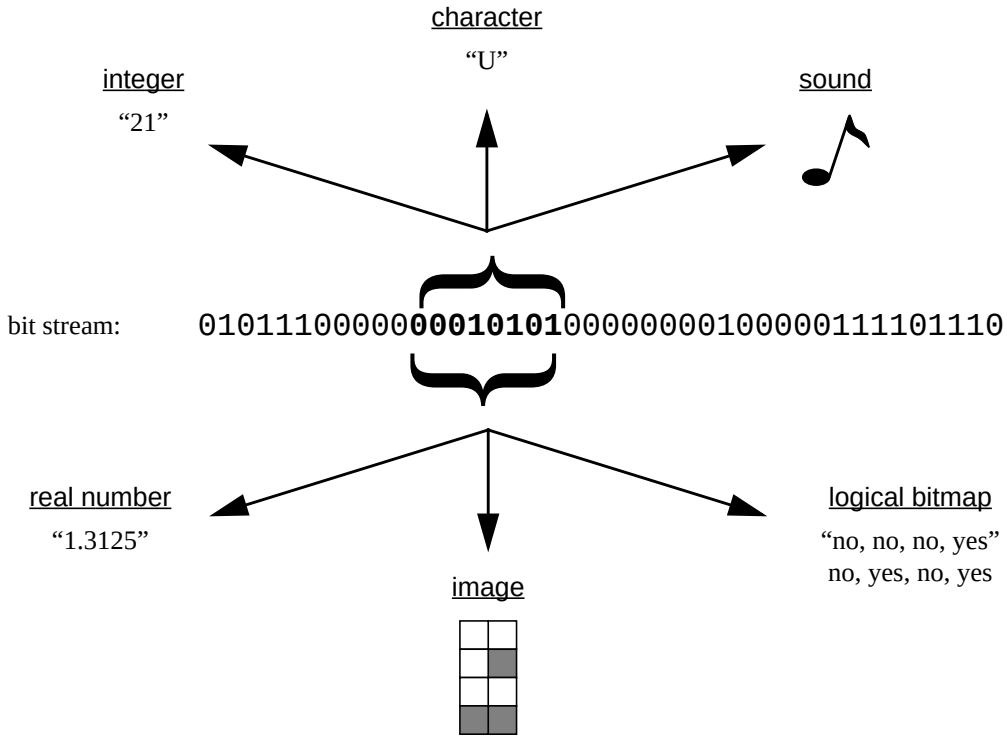


Figure 4: A bit stream can represent anything at all

Different length bytes allow different ranges of codes, which can in turn represent different numbers of characters. For example, 6-bit bytes provide barely enough codes to represent unadorned, uppercase text, whereas 8-bit bytes provide considerably more freedom.

byte-length	sample byte	code range	representable characters
6 bits	**000101**	0-63	{uppercase letters + digits + some punctuation}
8 bits	**00000101**	0-255	{upper + lowercase letters + digits + punctuation + "control" characters + graphical elements}

Figure 5: Chunks (or "bytes") can be of any length

The 4 bits at the start of this bit stream are intended to be read as the "key" integer 7, meaning that the remaining bytes in the bit stream are each 7 bits long. However, there is no way to tell from the bit stream itself how long the key integer is; if we were to erroneously read the first 5 bits of the bit stream as the key (instead of the first 4), we would erroneously conclude that the remaining bytes were each 15 bits long.

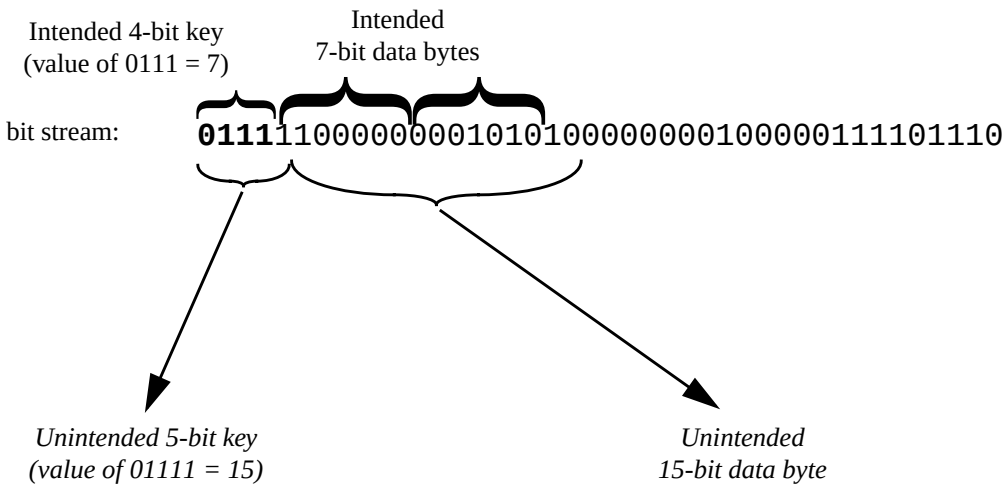


Figure 6: Bit streams cannot be made self-explanatory

However, this key integer must itself be represented by a byte of some length. How can a reader interpret the key without knowing how long it is? We need another key to explain how to interpret the first key! Computer scientists describe such recursive problems as requiring a "bootstrap" (i.e., a way of doing something without help from any external source, as in pulling oneself up by one's own bootstraps). In order to provide such a bootstrap, we must annotate our digital storage medium with easily-readable information that explains how to read it. In our scenario, the letter accompanying the disk must fulfill this role.

In addition, compression schemes (which reduce the length of bit streams, to lower the cost of storing and transmitting them) and encryption schemes (which encode them for privacy) make bit streams quite difficult to parse [see Figure 7]. And even after a bit stream is correctly parsed, we face another problem: If the resulting stream of bytes represents a sequence of numbers or alphabetic characters, decoding it seems straightforward: we simply interpret each byte according to the appropriate code. Yet this leads to a problem similar to that of encoding a key to specify the length of each byte in a bit stream. To interpret each byte, we need to know what coding scheme it uses; but if we attempt to identify the coding scheme by encoding a "code-identifier" in the bit stream itself, we need another code-identifier to tell us how to read the first code-identifier! Again we must bootstrap this process by providing easily-readable annotations.

As a simple example of compressing a bit stream without loss, “run-length encoding” replaces each sequence of 0s (000...0) by a count, indicating how many 0 bits were present in the given “run” (similarly for 1s). This can reduce the size of a bit stream without losing any information. For example, each run in the original bit stream shown can be represented by a 5-bit byte whose first bit specifies whether the run is of 0s or 1s and whose remaining 4 bits specify the length of a run (of up to 15 bits). This scheme is most appropriate for data that contains long sequences of 0s or 1s, such as digital imagery.

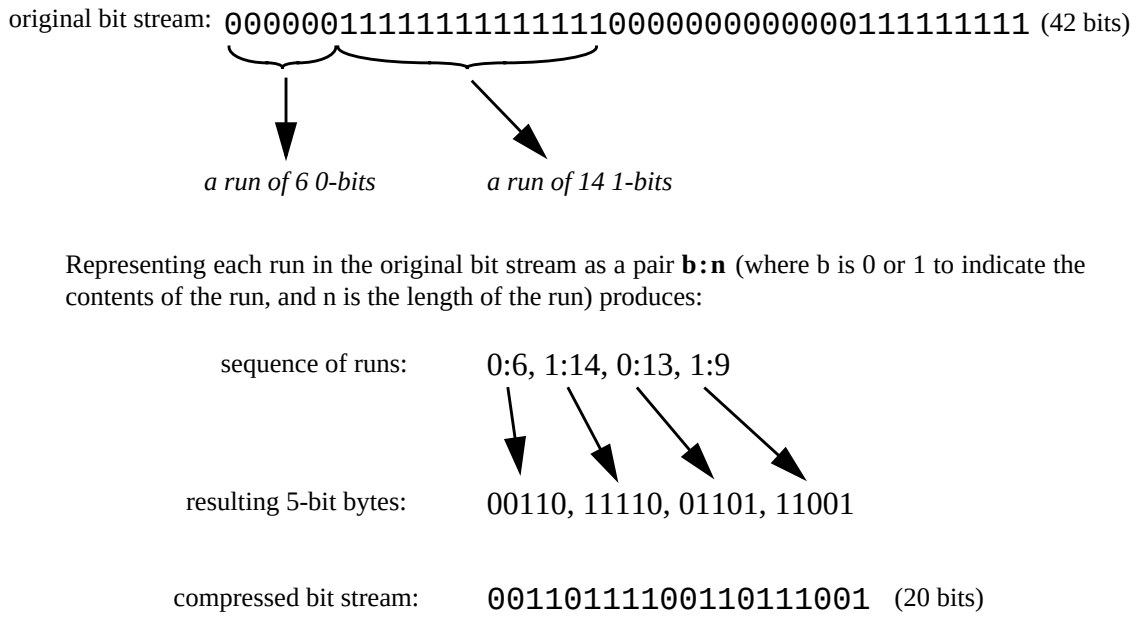


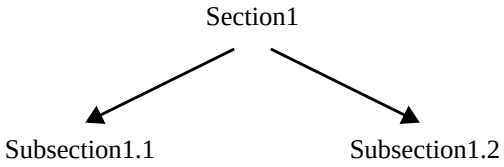
Figure 7: Compressing a bit stream

It’s all in the program

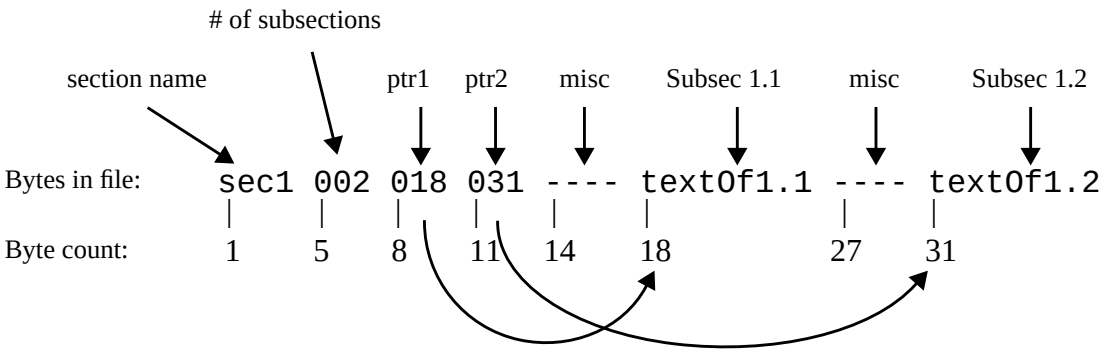
Yet the problem is deeper than this. Digital documents are typically saved as “files” of information: collections of bits corresponding to the bit streams representing specific documents. (Multiple documents are stored as separate files on a single digital medium; for simplicity we can assume a one-to-one correspondence between files and documents.) The bit stream in a document file may represent structures much more complex than sequences of fixed-length bytes. In particular, files often contain logically related but physically separate elements that are linked to each other by internal cross-references, consisting of pointers to other places within the bit stream or patterns to be matched. (Printed documents exhibit similar structure and cross-referencing, in which page numbers are used as pointers, while section names or other content references require the reader to search for specified text.) [see Figure 8].

In addition to having complex structure, many documents embed special information that is meaningful only to the software that created them. Word processing programs embed special format information in their documents to describe typography, layout, and structure (identifying titles, sections, chapter headings, etc.). Spreadsheet programs embed formulas specifying relationships among the cells in their documents. “Hypermedia” programs use embedded

Digital documents may encode structure as well as text. For example a document may consist of multiple subsections within sections:



This can be represented by a bit stream that contains pointers (ptr1 and ptr2) that give the byte-count at which each subsection begins:



"Hypertext" documents may consist of elements that are linked together to form multiple, alternative sequences, no one of which is necessarily any more "correct" than any other. In a document of this kind, a given element may appear as a subsection of several different sections (making the pointers in its bit stream even more essential for understanding its structure):

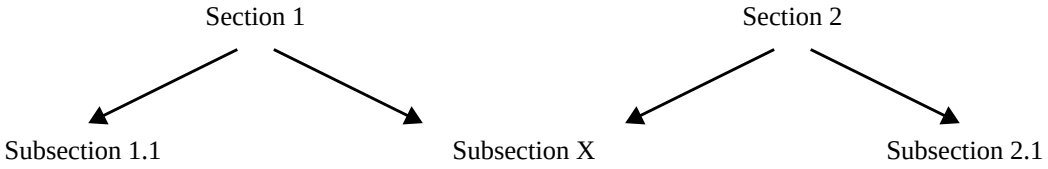


Figure 8: Document structure

information to identify and link text, graphics, imagery, sound, and temporal information in arbitrarily complex ways. For convenience, we will refer to all embedded information of this kind—including all aspects of a bit stream's representation, such as its byte lengths, character codes, multi-media information, and structure—as "coding" (though this term is often used more narrowly).

As documents become more complex than simple streams of alphabetic characters, it becomes increasingly meaningless to think of them as existing at all except when they are interpreted by the software that created them. The bits in each document file are meaningful only to the program

that created that file. In effect, document files are programs, consisting of instructions and data that can be interpreted only by the appropriate software. That is, a document file is not a document in its own right: it merely *describes* a document that comes into existence only when the file is "run" by the program that created it. Without this authoring program—or some equivalent viewing software—the document is held cryptic hostage to its own encoding.

As we discover the advantages of digital documents, we are coming to rely more and more heavily on those capabilities of the digital medium that transcend the limitations of the printed page. This may be partly a result of our infatuation with the novelty of information technology, but it nevertheless implies that nonlinear, multimedia documents will become increasingly prevalent, at least while our infatuation lasts. To the extent that we create digital documents of this kind, they will be impossible to access without appropriate software.

Suppose that my grandchildren manage to read the intended bit stream from the CD-ROM. Only then will they face their real problem: without further help from my accompanying letter, how can they interpret the coding of the document file on the disk? If the document is a simple sequence of fixed-length bytes representing alphabetic characters, then trial-and-error experimentation might decode the document as a stream of text. But if the document is more complex than this, a brute-force approach—trying to "decrypt" the structure and meaning of an arbitrary document file—is unlikely to succeed. The meaning of a file is not inherent in the file itself, any more than the meaning of this sentence is inherent in its characters or words. In order to understand a file, we must know what its content *signifies*, i.e., what meaning it has in the language of its intended reader. Unfortunately, the intended reader of a digital file is a computer program, not a human.

Digital documents therefore have the discouraging characteristic of being software-dependent. They cannot be "held up to the light" but must be viewed by using appropriate software. Is it necessary to run the specific software that created a document, or is it enough to run some similar program that can at least partially interpret the document file's encoding? In some cases the latter may be sufficient, but it is naive to believe that any document encoding—however natural it seems to us today—will continue to be readable by future software for very long. The information technology revolution continually creates new paradigms, which often abandon their predecessors instead of subsuming them. Collaborating authors and publishers are already confounded by a bewildering and ever-changing collection of incompatible document file formats that must be translated back and forth, often with annoying losses of format, structure, and even content.

If "reading" a document means simply being able to extract its content—whether or not it is in its original form—then we may be able to avoid running the original software that created the document. But content can be subtle: translating from one word processing format to another, for example, often displaces headings or captions or eliminates them entirely. Is this merely a loss of structure, or does it impinge on content as well? If we transform a spreadsheet into a table, thereby deleting the formulas that relate the cells of the table to each other, have we retained its content?

As an extreme example, suppose that the document in my attic explains that my fortune can be found from a treasure map represented by visual patterns of inter-word and inter-line spacing in

the digital version of this paper, stored on the CD. Since these patterns are artifacts of the formatting algorithm of my word-processing software, they do not appear in a printed or posted version of this paper: they will be visible only when the original digital version is viewed using the software that created it. Ultimately, if one needs to view a complex document as its author viewed it, one may have little choice but to run the software that created it.

In order to read the document file stored on the CD-ROM in my attic, my grandchildren must therefore know what program created the file; but what are their chances of finding that program fifty years from now? If I include a copy of the program on the CD-ROM itself, they must still find the operating system software that allows the program to run on an appropriate computer. Including a copy of this operating system on the CD-ROM may help, but the computer hardware required by that operating system will have long since become obsolete. A digital document depends not just on the specific program that created it but on the entire suite of hardware and software that allowed that program to run. How much of this can I store on the CD-ROM, to make it as self-sufficient as possible? What kind of digital Rosetta Stone can I leave to provide the key to understanding the contents of my disk? What can we do to ensure that the digital documents we are creating will not be lost to the future?

What's an author to do?

As a first step, we must preserve the bit streams of digital documents. This requires copying the bits onto fresh media to preserve their physical existence and copying them to new forms of media to ensure their accessibility. The need to refresh digital information by copying it onto new media (and possibly translating it into new formats, sometimes called "migration") has been recognized in the library sciences and archives literature, as well as in a number of scientific and commercial fields. This requires an ongoing effort: future access depends on an unbroken chain of migrations with a cycle time short enough to prevent media from becoming physically unreadable or obsolete before they are copied. A single break in this chain can render digital information inaccessible—short of heroic effort. Given the current lack of robustness and rate of evolution of media, migration cycles may need to be as frequent as every few years, requiring a significant commitment. Moreover, copying the bit streams of digital documents in this way is necessary but not sufficient. Like an illiterate monk dutifully copying text in a lost language, migration may save the bits but lose their meaning.

Preserving digital documents is analogous to preserving ancient written texts. Just as with digital documents, it is sometimes necessary to refresh an ancient text by transcribing it, since the medium on which it is recorded has a limited lifetime—though parchment or stone tablets last noticeably longer than magnetic disks. An ancient text can be preserved in one of two ways: either by copying it in its original language or by translating it into whatever language is current at the time of transcription.

Translation is attractive because it avoids the need to retain knowledge of the text's original language, yet few scholars would praise their ancestors for taking this approach. Not only does each translation lose information, but translation makes it impossible to *determine* whether information has been lost, because the original is discarded. (In extreme cases, translation can completely destroy content, as when translating a dictionary. Imagine some misguided

archaeologist having blindly translated all three copies of the text on the Rosetta Stone into English the moment it was discovered and discarding the original: an invaluable correspondence between languages would thereby have been translated into a trivial repetition of the same text.)

Copying text in its original language, on the other hand, guarantees that nothing will be lost—assuming that knowledge of the original language is retained along with the text. This amounts to saving the "bit stream" of the original text.

Similarly, there are two strategies for dealing with digital documents, both of which have received attention by archivists, library scientists, and others concerned with the preservation of records. The first attempts to translate documents into standard, system-independent forms, while the second attempts to extend the longevity of systems so that documents will remain readable using their original software. Unfortunately, neither approach promises a complete solution without considerable additional effort.

The illusion that standards provide an answer

On the surface, it may appear preferable to translate digital documents into standard forms that can be guaranteed to be readable in the future. This would circumvent the need to retain the ability to run the original software that created a document. Proponents of this approach cite the relational database model (developed by E.F. Codd in the 1970s) as a paradigmatic example. Since all relational database management systems (RDBMSs) implement this same underlying model, any relational database produced by any RDBMS can in principle be translated without loss into a form acceptable to any other RDBMS. A standard relational form could therefore be established, and all relational databases could be translated into this form. Files represented using this standard could be copied to new media as necessary, and the standard would provide readability for all time. This sounds tempting, but it is flawed in two fundamental ways.

First, although the formal mathematical definition of the relational database model leads all RDBMSs to provide equivalent baseline capabilities, most commercial RDBMSs distinguish themselves from each other precisely by offering features that extend the standard relational model in non-standard ways. Therefore relational databases are less amenable to standardization than they appear. If a Procrustean standard relational database form were imposed on existing relational databases, many of them would lose much of their richness.

Moreover, the relational model is rapidly giving way to an object-oriented database model (which represents entities as structured, composite objects), as the limitations of the relational approach become apparent. This evolution is neither accidental nor undesirable: it is the hallmark of information technology that it evolves at a rapid rate. Data saved in relational form may well become inaccessible as relational database systems give way to object-oriented systems.

Furthermore, the relational database model does not constitute a paradigmatic example, because it is practically unique. No other type of digital document comes close to having as formal a basis on which to erect a standard. Word processors, graphics programs, spreadsheets, and hypermedia programs each create documents with far greater variance in expressive capability and format than relational databases. The incompatibility of word processing file formats is a notorious

example—nor is this simply an artifact of market differentiation or competition among proprietary products. Rather it is a direct outgrowth of the natural evolution of information technology as it adapts itself to the emerging needs of users. No common application other than relational database management is yet a suitable candidate for long-term standardization.

The false promise of "migration"

In the absence of long-term standards for each type of digital document, it might still be possible to translate a document into successive standards, each having a relatively short life-span (on the order of one or two migration cycles). It is sometimes suggested that a variation of this approach occurs naturally, since documents that are in continuous use within organizations are translated into new formats as needed. However, this breaks down when a document ceases to be used in the ongoing business of the organization that owns it, since few organizations can justify the cost of translating documents that they no longer use.

The successive translation approach avoids the need for ultimate standards, but it compounds the problem of losing information, since each translation may introduce new losses. In theory, translating a document into a standard (or sequence of standards) retains a path back to the original. By keeping descriptions of each standard used in the sequence of translations (where these descriptions themselves would have to be translated into successive standards in order to remain readable), future scholars might reconstruct the original document. Unfortunately, this requires that each translation be reversible without loss, which is rarely the case. If all early versions of Homer had been discarded after translating them, there would be little hope of reconstructing them by translating backwards again.

Finally, the translation approach suffers from a fatal flaw. Unlike ancient Greek and English, which have roughly equivalent expressive power and semantics, digital documents are still evolving so rapidly that periodic paradigm shifts are inevitable. And new paradigms do not always subsume their predecessors: they represent revolutionary changes in what we mean by documents. By definition, paradigm shifts do not necessarily provide upward compatibility. Old documents cannot always be translated into new paradigms in meaningful ways, and translating backward is frequently impossible. The relational database model provides a case in point. Many earlier, "hierarchical" databases had to be completely redesigned to fit the relational model, just as relational databases are now being drastically restructured to make use of newer object-oriented models. Paradigm shifts of this kind can make it extremely difficult, if not meaningless, to translate old documents into new standard forms.

Although defining ultimate standards for digital documents may be an admirable goal, it is premature. Information technology is still on the steepest slope of its learning curve. The field is too new to have developed an accepted, formal understanding of the ways that humans manipulate information. It would be presumptuous to imagine that we are ready to enumerate the most important kinds of digital applications, let alone to propose that we are ready to circumscribe their capabilities by standardizing them. Any attempt to force users to settle for artificial limitations imposed by such standards would be futile, since the momentum of the information technology revolution derives directly from the attraction of new capabilities. It may become feasible to define long-term standards for digital documents sometime in the future, when

information science rests on a more secure, formal foundation, but standards do not yet offer a solution to our problem.

Byting the bullet

The alternative to translating a digital document is to view it by using the software that created it. In theory, this need not require that we actually *run* the software. Suppose we could describe its behavior in some system-independent way and save that description instead of the software itself. Future generations could interpret the saved description of the software to recreate its behavior, thereby reading the document. While this sounds promising, information science has not yet produced methods to describe the behavior of software in the depth required for this approach. High-level behavioral formalisms of this kind—that describe programs in terms of their interactions with humans in performing information processing tasks—may eventually emerge, but they are not yet on the horizon. In their absence, the only meaningful description of the detailed behavior of a program (in most cases) is the program itself. In order to recreate the behavior of an arbitrary program, there is currently little choice but to run it.

This requires saving digital document files and the programs that created them, as well as all system software required to run those programs. Though this is a daunting task, it is theoretically feasible. It is not uncommon to save and distribute digital documents along with appropriate viewing software—and sometimes even a copy of the appropriate version of the operating system needed to run that software. This is often the only way to ensure that a recipient will be able to read a document (assuming the required hardware is available). Furthermore, in many cases, it is necessary only to *refer* to the appropriate application and system software, since these programs are ubiquitous. Archives of free (public-domain) software are already proliferating on the Internet, and with luck, copyright and royalty restrictions for proprietary programs may expire when those programs become obsolete, making them available for future access to historical documents.

If digital documents and their programs are to be saved, their migration must be carried out with extreme care to ensure that their bit streams are not modified in any way that affects their interpretation, since programs and their data files can be corrupted by the slightest change. This is a reappearance of the translation problem discussed above. Copying bit streams must not inadvertently change byte size, introduce additional bits, reverse the order of bits, compress or encrypt data, or in any way modify the encoding of the bit stream. If such changes are unavoidable, it is not enough to record information sufficient to interpret the final encoding: one must also record sufficient detail about each such transformation to allow reconstructing the original encoding of the bit stream, on which its semantics may have relied. For example, if internal pointers in a document consist of bit counts, they would be invalidated by any transformation that changed the number of bits in the stream. (Finding all such pointers in a document and adjusting them to account for the changed bit count is analogous to—but much harder than—the “Year 2000” problem of finding all two-digit numbers in a program that represent years.) Although it is possible to design bit streams whose semantics are immune to any expected transformations of this sort, future migration cycles may introduce unexpected transformations, which the designer of the bit stream could not have foreseen.

Ideally, bit streams should be considered inviolable entities, sealed in virtual "envelopes" whose contents are preserved verbatim. If transformation is inescapable, it must be reversible without loss; easily-readable information associated with each envelope must describe its contents and its transformation history, if any. It will be a serious challenge to encapsulate bit streams in this way and ensure that they retain the necessary contextual information in a form that remains easily readable well into the future.

How can we run obsolete hardware?

We must still show how we can preserve the hardware needed to run the software to view a digital document. The migration process eliminates the need to preserve storage devices such as disk drives, but systems and application software still depend on hardware, both for computation and for input and output. One obvious approach is to try to maintain computers in working condition long after they become obsolete. Indeed a number of specialized museums and informal "retro-computing" clubs are attempting to do this. However, despite a certain undeniable charm—attributable to its technological bravado—this strategy is futile in the long run: mechanical components fail, and even electronic circuits age as the "dopants" that make silicon into a useful semi-conductor and the metal traces that connect components on each chip diffuse and "migrate" within their substrate. As components wear out, the cost of repairing or replacing them (and of retaining the expertise required to do so) will quickly outweigh the demand for maintaining any given computer. How long can we hope to keep current systems in operational condition? Twenty years? Fifty?

Moreover, since records cannot be expected to survive on their original media but will only be readable if they have migrated to new media, using old computers to read old records would require building special-purpose hardware interfaces between each old computer and each new generation of storage media, for example to allow a vintage-1960 computer to read data from a CD-ROM. The effort of designing, building, and maintaining such interfaces would rapidly become prohibitive.

Fortunately, it is not necessary to preserve physical hardware to be able to run obsolete software. Emulators—programs that mimic the behavior of hardware—can be created to take the place of obsolete hardware as needed. Assuming that future computers will be orders of magnitude more powerful than ours, future users should be able to ask their computers to generate emulators for obsolete systems on demand. This may require accessing saved specifications for the desired hardware, but this hardware could not have existed in the first place if detailed specifications for its design and construction had not also existed. These specifications must be saved in a digital form that will be readable by future emulator generators (whether human or machine). Alternatively, specifications for emulators could be saved directly; since most new computers are emulated prior to being produced (as part of the design and evaluation process) machine-independent specifications for emulators could be derived from these existing machine-specific emulators. In the six years since I first suggested this approach, it has begun to occur spontaneously in an unlikely context. Special interest network groups are currently creating and sharing emulators for obsolete video game processors and early personal computers. Obsolete programs for these processors are being copied to current media by ingenious methods, such as

digitizing audio signals generated by makeshift devices that read obsolete media, and these programs are then run under emulation on modern computers.

Putting it all together

As we have seen, interpreting a bit stream depends on knowing how it has been encoded, and a bit stream cannot be fully self-describing, since any description that we encode in the bit stream must itself be interpreted. The only way to bootstrap this process is to include easily-readable annotation with every digital document, explaining how to interpret its bits. The letter accompanying the disk in my attic serves this purpose; but if the disk had been copied onto newer media, how would the information in the letter have been preserved? A unit of storage in the year 2045 may hold the bit streams of thousands of CDs. Even if each disk had an associated letter supplying the necessary context, how could this be carried along with the bit stream from each disk?

Clearly, any annotation must itself be stored digitally, along with its associated bit stream; but it must be encoded in a digital form that is more readable than the bit stream itself, in order to serve as a bootstrap. This is an ideal role for standards: a simple text-based standard should be able to encode sufficient explanatory information to allow interpreting an encapsulated bit stream. Whenever bit streams migrate to newer media, their annotations must be translated from their previous form to whatever standard is current at the time of the migration.

Reading current digital documents in the future will not be easy. Eventually, information science may develop models of human information processing and computation that will allow digital documents to be stored in system-independent form, but this cannot happen in time to save the current generation of these documents. Similarly, long-lived storage media may ultimately make migration less urgent (the cost of migration may motivate the acceptance of such media, overriding our appetite for improved performance), but there is no sign of this happening yet. In the meantime, we must act quickly and decisively if we are to help our descendants read our documents.

We must develop evolving standards for encoding explanatory annotations to bootstrap the interpretation of digital documents that are saved in nonstandard forms. We must develop techniques for saving the bit streams of software-dependent documents and their associated systems and application software. We must ensure that the hardware environments necessary to run this software are described in sufficient detail to allow their future emulation. We must save these specifications as digital documents, encoded using the bootstrap standards developed for saving annotations so that they can be read without special software (lest we be recursively forced to emulate one system in order to learn how to emulate another). We must associate contextual information with our digital documents to provide provenance as well as explanatory annotations in a form that can be translated into successive standards so as to remain easily readable. Finally, we must ensure the systematic and continual migration of digital documents onto new media, preserving document and program bit streams verbatim, while translating their contextual information as necessary. If all of these factors come together, they should enable obsolete digital documents to be read as illustrated in Figure 9.

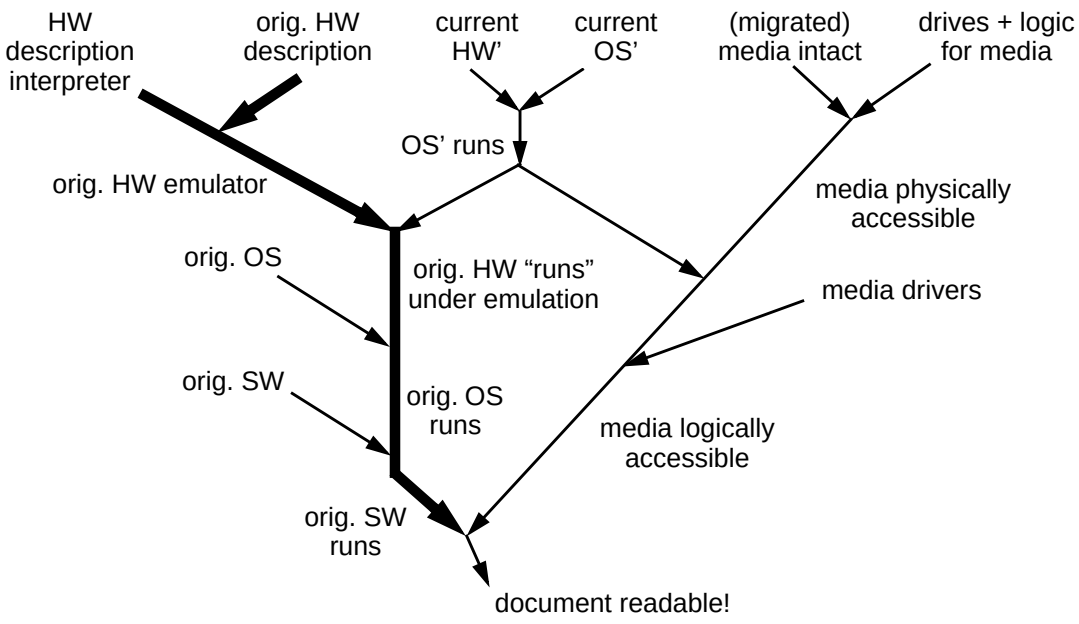


Figure 9: Using emulation to read an obsolete digital document

Conclusion

Beyond having obvious pragmatic value, the digital documents we are currently creating are the first generation of a radically new form of record-keeping. As such, they are likely to be viewed by our descendants as valuable artifacts from the dawn of the information age. Yet we are in imminent danger of losing them even as we create them. We must invest careful thought and significant effort if we are to preserve these documents for the future. If we are unwilling to make this investment, we risk substantial practical loss, as well as the condemnation of our progeny for thoughtlessly consigning to oblivion a unique historical legacy.

Where does this leave my grandchildren? By assumption, the information on their fifty-year-old CD has not migrated to newer media; but if they are fortunate, it may still be readable by some existing disk drive, or they may be resourceful enough to construct one, based on instructions in my accompanying letter. If I include all necessary system and application software on the disk, along with a complete and easily decoded specification of the hardware environment required to run it, they should be able to generate an emulator that will display my document by running its original software. I wish them luck.

Short bibliography

Archival Management of Electronic Records, Archives and Museum Informatics Technical Report no. 13, David Bearman, ed., Archives and Museum Informatics, Pittsburgh, 1991 (ISSN 1042-1459).

"Text and Technology: Reading and Writing in the Electronic Age," Jay David Bolter, *Library Resources and Technical Services*, 31 (January/March 1987), pp. 12-23.

"Understanding Electronic Incunabula: A Framework for Research on Electronic Records," Margaret Hedstrom, *The American Archivist*, 54:3 (Summer 1991), pp. 334-54.

"Scholarly Communication and Information Technology: Exploring the Impact of Changes in the Research Process on Archives," Avra Michelson and Jeff Rothenberg, *The American Archivist*, 55:2 (Spring 1992), pp. 236-315 (ISSN 0360-9081).

Research Issues in Electronic Records, published for the National Historical Publications and Records Commission, Washington, D.C., by the Minnesota Historical Society, St. Paul, Minn. 1991.

"Ensuring the Longevity of Digital Documents," Jeff Rothenberg, *Scientific American*, January 1995 (Vol. 272, Number 1), pp. 24-29.

"Metadata to Support Data Quality and Longevity," *The First IEEE Metadata Conference*, April 16-18, 1996, NOAA Auditorium, NOAA Complex, Silver Spring, MD, available only online at http://www.computer.org/conferen/meta96/rothenberg_paper/ieee.data-quality.html

Taking a byte out of history: the archival preservation of federal computer records, Report of the U.S. House of Representatives Committee on Government Operations, Nov. 6. 1990 (House Report 101-978).