# Notes about Nelson's File Structure for The Complex, The Changing, & The Indeterminate

Gene Michael Stover

created Thursday, 2005 May 5
updated Saturday, 2005 May 7

## Contents

## 1 What is this?

These are my notes about "A File Structure for The Complex, The Changing, and The Indeterminate", by Theodor H. Nelson [1].

He calls his file format the Evolutionary List File (ELF). Readers in 2005 should note that this file format has nothing to do with the ELF executable file format of our current operating systems.

## 2 Brief observations

Nelson takes the time in [1] to define the term "hypertext". I have read elsewhere that he created the term, so maybe [1] was the first time the word "hypertext" was ued in print.

Nelson estimates that one half of time spent writing is specifically spent in cut-&-paste activities (cutting real paper & using real paste). Combine that estimate with some of Nelson's functional requirements for an ELF/memex system, & see that Nelson was sort of predicting modern word processors.

I think I can see the seeds of Nelson's Xanadu in [1]. Nelson's ELF/memex system is sort of a Xanadu without networking, a personal Xanadu.

## 3 How to surf the web

In an article written fully 40 years ago, I learned a better way to surf the web.

From Nelson's quotation of Vannevar Bush & also from things Nelson writes directly, I now see the importance of saving the URLs that you follow. The trail should be persistent & annotated, so the "back" button & history lists of web browsers is inadequate.

I can see it now. In Nelson & Bush's world, while reading a document, you might realize that it belongs in your list of political events, so you'd insert it there.

You continue reading the same document, but now you want to research something it says about a particular politician. So that you can remember to finish reading this file, you insert it in your list of things to finish, & then you follow a link to a page about that politician.

You notice that you have already seen this new page on some other day because your web browser shows you that it's already in two of your list of politics, & it's also linked to your entry about blue whales in your "environmental issues" list. You skip to the blue whales document & see that the politician has proved to be pro-environment in his votes for several marine life issues.

Okay, so I'm not good at writing use cases. But notice what's happening in the Nelson/Bush world. From one web page, you can jump to a related page, possibly through a link provided in the first page. We have this feature on the real world's web. What's better in the Nelson/Bush world is that each page is (or can be) grouped in a list of related items, & it can be related to pages through links which you (or maybe some other reader, but not the page's author) create. From the second page, you could follow a link to another page inserted by the author (again, just like the real world's web), but you can also take a step outside of the page to see the lists which contain it & to see other pages to which you have linked it. So a page can be linked to other pages through connections that you, the reader, notice.

The links themselves can form threads through pages which are grouped into lists. You might arrive at a page through author-provided links or through one thread of links. You might leave that page & go to another through more author-provided links, through links which are part of the original thread, or through another thread of links.

To me, it sounds more exciting & useful than what we have in the world wide web now.

## 4 Types of objects

The types of objects in Doctor Nelson's *elf* files are:

- entries,
- lists, &
- links.

## 5 ELF file format

I think the ELF file format is a special case of what used to be called a *network database*, so ELF might be implemented in a network database.

*Relational databases* are at least as general as network databases, so ELF could be implemented in a relational database.

I wonder if these observations were apparent to programmers of the 1960s. I know they had Algol, Lisp, FORTRAN, & COBOL, but from other things I've read from that time[1], I am under the impression that they were more concerned with CPU registers & memory locations than we are now. Nelson wrote his ELF article [1] at a high-level of abstraction, so I would not be surprised if he did make these same observations at the time.

Network databases never became popular. Or maybe they did, but for a short time which is long gone because relational databases have been king since the middle of the 1980s, if not before.

The reason relational databases superseded network databases (if network databases ever had a foothold at all) is that relational databases are as general as network databases but also more manageable. If ELF files are network databases with a little structure added, does ELF's lack of popularity have roots in network databases's lack of popularity? Is there some technical reason, which I don't see, that ELF is impractical?

## 6 Viewing the ELF file

Notice that ELF files place the burden of presentation on a *memex*-like application. Maybe that was a good idea.

There might be some standard presentation forms that would be appropriate for any memex user. Some of them might include:

[1] And I wish I could cite a few now.

**entry** For a given entry, show all the lists it is in.

**list** For a given list, show all the entries.

**two-level entry** For a given entry, show the lists it is in, & show abbreviated data (such as title) for the entries in the lists.

**two-level list** For a given list, show all the entries, & for each entry, show the names of the lists it is in.

Those views rely on the (very little) structure which is native to ELF files, but views which make use of higher-level structures might be possible.

**Higher-level View Example**: Consider a view which shows the entries in a list, sorted by date. The dates of entries could be available to this viewer as entries in a list of dates to which this list links.

To use this view, the user would select a list & tell memex to display it with this view. The list of dates could be known to the view by a particular name, & its entries could be in a format known to the view.

What if some entries in the list did not link to the list of dates? What if some entries in the list of dates were not well-formed dates?

This Higher-level View Example also shows the need of standards. The view might assume that some data it needed was in entries of a particular format or in lists of a particular name. The view is surely a program. What if it's manufactured from a software company that makes many different views? That might be cool because the views from that company might have similar & compatible requirements for lists & entry formats in the user's ELF file. How would those views operate with views from a competing manufacturer?

What if someone manufactured a malicious view, which we might call a virus view? That view might destroy important entries or lists in a user's ELF file. Could information be irrevocably lost? Would other views become unusable?

## 7 Analyses

Nelson was careful to state that he did not imagine intensive processing on ELF files, but let's think about that possibility anyway.

The views I mentioned in Section 6 wouldn't do much computation. They would mostly display structure that is explicit in the ELF file. What of views whose output contained less obvious information?

**Computationally Intensive View Example 1**: Given two entries (which may or may not be related via links), a view might show entries or lists which were related to the first two entries.

**Computationally Intensive View Example 2**: In Section 6, I mention the possibility of a view which displayed the entires in a list, ordered by the "time entries" to which those first entries link. What if the entries in a list were not linked to a list of times? There might be a time-ordered view, like the one in Section 6, except that it extracted times from hints in the entries themselves.

The previous example begs the question “Would we need a quick version & a computationally intensive version of every view?” The answer is that we would not. Instead of making more complicated views, we could instead of analysis programs which create lists & entries for use by views. For example, if a user had not already linked her entries to date objects, she might run a “time extraction” analysis program on a list of entries. The time extraction analysis program would try to figure out the times that should be associated with each entry, insert those times into a specific list, & then link each entry to the times in the list of times. Then the user could apply the “list ordered by time” view.

So analysis programs could create entries & lists for use by views & by other analysis programs. This would be another way of achieving the same functionality we have with unix-style “pipes & filters”[2].

## 8 Really Big Analyses

Could there be programs to analyze an ELF file & make deductions? I don’t mean trivial little deductions such as the date an entry was authored; I mean *big* deductions, which we would consider artificial intelligence? Would it be useful for an analysis program to tell you how closely related two entries were? How about an analysis which reported the path (which entries on which lists) from one entry to another?

Those analyses would tell you about connections one user has made in her own ELF file, but what if you ran analyses on the ELF files of many users?

## 9 Implementing ELF memex

In Section 5, I suggest that ELF files could be implemented on network or relational databases. Here are other ideas about implementation, & these ideas are in more detail.

In a hypothetical implementation, entries could be any file with an URL. That would make every page on the world wide web a potential entry. It would allow for many files available view FTP, plus local files.

In Nelson’s ELF system, all entries are editable, but for a given user, most web documents are not editable. Instead of rebelling against that reality, but a hypothetical memex system could allow the user to edit entries which are local files.

If we implemented our hypothetical ELF memex in Lisp, lists could be . . . lists! No surprise there. Links would be `cons` cells, which are usually implicit in Lisp.

In Nelsons ELF memex, lists must be accessible to the user so she can add entries to them. Our Lisp-based ELF memex will need to associate lists with

[2]It’s a functionality which is criminally under appreciated in the computing world, by the way.

names. The easiest way to do that might be binding them to symbols (possibly with SETQ.

There should be a well-known list of all entries. Maybe it could be bound to the symbol `*all-entries*`. Similarly, there should be a Lisp list of all memex lists, & it could be called `*all-lists*`.

If a user is viewing a web page that she wants to add to her ELF memex system, she might give the page's URL to a function called INSERT-ENTRY, which might be defined like this:

```
(defvar *all-entries* () "The list of all entries")

(defun insert-entry (url)
  "Ensure that URL is in the list of all entries.
Modifies & rebints *ALL-ENTRIES*."
  (declare (type string url))
  (unless (member url *all-entries*)
    (push url *all-entries*)))
```

After adding an entry, the user will probably want to insert it into a list other than the *ALL-ENTRIES* list. Assuming the new entry relates to programming, & she has already created a *PROGRAMMING* list, she just needs to PUSH the new URL onto the *PROGRAMMING* list.

Links are more complicated. To show that *entry A* in *list A* links to *entry B* in *list B*, we might use a 4-tuple: `(entry-a list-a entry-b list-b)`. In such a 4-tuple, the *entry-a* & *entry-b* would be the URL strings that we gave to the INSERT-ENTRY function. *List-a* & *list-b* are something of a problem. If they are the FIRSTs of two lists, then we can't PUSH things onto those lists. If *list-a* & *list-b* are the symbols to which the lists are bound, then we must eval the symbols to obtain the lists, & modern Lisp programmers avoid eval. So maybe *list-a* & *list-b* should be the *names* of lists, & those names should be stored in an ELF memex symbol table of some sort.

In a quick hack version of ELF memex, the user could call a DUMP-HTML function after making changes to the memex system. DUMP-HTML would write HTML files that show the entries & lists in various views. I called this a quick hack, but it would in fact be really useful as long as the user wasn't making lots of little changes & having to call DUMP-HTML after each one.

For a version of ELF memex which was more friendly to refreshing the views after each change, we might use CGI programs or scripts inside a web browser. Whichever technique we used, it would generate a view from the lists & links in real time.

There could also be a web interface that allowed the user to perform CRUD[3] on entries, lists, & links.

[3] CRUD is Create Report Update Delete.

## A Other File Formats

- This document is available in multi-file HTML format at http://lisp-p.org/nelf/.
- This document is available in Pointless Document Format (PDF) at http://lisp-p.org/nelf/nelf.pdf.

## References

[1] T. H. Nelson. Complex information processing: a file structure for the complex, the changing and the indeterminate. In *Proceedings of the 1965 20th national conference*, pages 84–100, New York, NY, USA, 1965. ACM Press. http://doi.acm.org/10.1145/800197.806036 (Must be ACM member or Digital Library subscriber to access.).