A bold new approach to storing data could sweep away many difficulties users face.

THE TYRANNY OF THE FILE

by Theodor H. Nelson

The fish does not see the water. And we computer people, even the savants, whizzes, and frontiersmen among us, generally fail to see the most oppressive and devastating aspect of our working lives. I refer to the file, and the conceptual structure of the storage methods it forces us to deal with. Chaotic and fragmented, the world of computer files presents an enormous barrier to the creation of the clean systems of tomorrow.

We tell beginners, "It must be this way." But I believe there has to be a fundamental redesign of the way we use computers, undoing this harm and difficulty. A clean new interface to our worlds of information could sweep away a vast amount of the difficulty that people have with computers.

Indeed, what the world needs is a generalized form of storage that will grow and adapt and hold data for every type of application; that may be shared among all applications, yet does not slant or gerrymander those data in any particular way; that creates a common, clarifying, and universal system of storage.

Computer use is principally concerned with maintaining order in a growing jumble of files. Some files stand alone, some must be managed in clusters, but typically they are all piled together on disk with archival copies and backups in an ever more confusing tangle.

Having to keep track of files is an endlessly complex and exasperating chore. You have to name each one. You have to rename them. You have to keep copies. You have to move them by copying and then deleting one of the survivors. (Deleting the correct file in a jungle of similar names is roughly like holding your baby and your garbage over the incinerator and letting go of one, 10 times a day.) Then there are the endless problems of making backups and finding space for them (often in dark crannies of the disk), the escalating annoyance of disk volumes filling to capacity, and the emergency transfers and deletions that must be made.

Typically, from time to time, somebody just throws it all away and starts afresh, or just puts the whole mess in storage somewhere and starts over, or does a sweeping reorganization of files during which a great deal of data is lost.

Attempts have been made to organize these chores according to various schemes, the most popular of which is the hierarchical file structure. With catalogs and directories, this structure originally seemed an improvement because it provided more places to put things and because it seemed to match well the structure of certain applications.

Some people think hierarchically and that's fine, but those who don't should not be forced to do so. There are those who imagine that forcing a problem into a hierarchical structure promotes clear and rigorous thinking. But that is, to use the politest possible term, malarkey; mapping any set of ideas to any other may present interesting exercises to the mind, but there is little point if the two sets do not match well.

Unfortunately, the hierarchical model imposes intricate, fixed pathways that we must commit to and memorize and which are quite hard to change. Yet the way we think of our work is constantly changing at the highest level. (Lucky are they whose ideas are fixed and unchanging.) Unfortunately, existing file methods stick us forever with the groupings that we start with—those divisions and hierarchies that seemed appropriate at the beginning—unless we do the elaborate reorganizations that nobody has time for anyway.

Many sets of data need to be divided into separate files even though they are related to each other, but the connections, not being represented, tend to evaporate. Cross-references between them, the interconnections of shared material, commonality and parallels—all these become easily lost because they are not easily represented.

There are ad hoc programs we can build to cope with what is not built into the storage system. These include programs for comparing files, indexing programs that mark points in text, delta list programs to manage the history of changes, and software configuration management programs, which produce different versions of the same programs from a common library. But I insist that if we had proper storage, all these functions would be handled in the data structure as a common reference system, available to all software.

SOFTWARE AFFRONTS USER

Most software based on this file model affronts the user with tedious formalities of opening and closing. Since files, their names, and versions constitute the surface structure of this universe, selecting and opening them is a level of annoyance that makes firing up an application as tedious as opening a bank account. Such unnecessary distraction and formalism wear you out before you even start and they forbid acting on inspiration. The situation is comparable to having to deal with a desk clerk in order to make love. (Programs like Sidekick from Borland International help in this regard, but the elaborate formalities are still necessary to put files away when the user is done.)

We should be able to work on numerous things at once, Ping-Pong style, never having to deal consciously with the formalisms of opening and closing files and applications. Many would say you could escape some of these annoyances by conventional methods. Today's windowing packages are a start, but they just disguise underlying conventional file methods that must be grappled with as usual when the day is done.

There are subtler problems to be solved. I would argue that the conventions of files as we know them put pressure on software design to take certain oversimplified forms. For instance, we are familiar with the database model, in which separately coded items, or records, may be searched according to various criteria. We are also familiar with the word processing model, where sequential text may be scanned, revised, and printed. But why can't these approaches be combined? What if one wants to provide categorical codes for sections of text inside their documents, to flag them for reference by another application? Today's software does not permit that. (A partial exception is the software product Dayflo from Dayflo Corp., Irvine, Calif.) You might also like to search for the coded pieces and sort them like database

Some people think hierarchically, and that's fine, but those who don't should not be forced to do so.

items, yet see them in their living contexts at the same time. While these functions are possible to program, the conventions of files strongly pressure the programmer to oversimplify both the data and their uses.

What we need, I submit, is a world without files. By that I mean a world where the user sees his or her latest work in its preferred appearance and format while still being able easily to trace interconnections and compare versions. Each application should be like a door to a world: the user opens doors into applications, with multiple views of materials, each a different context or way of working on it. We need multiple pathways to the same material at the user level. Whole environments—application programs and their open files—should easily be snapshot (frozen and put away so that they can be brought back to life quickly without reopening files one by one). This has been the intent of many integrated software packages, notably Lotus's Symphony, which tries to make available several different views of common materials.

Often the same pieces of text or, more common in the future, graphics, need to be worked on in several different contexts. As a result, different versions of the materials develop, growing apart in different ways. It should be possible, however, gradually to change and separate different versions, yet keep their commonalities traced back to the original version. This does not exist in any well-known system.

DIFFERENT STORAGE APPROACH

What I propose requires a different approach to storage. It is best explained by degrees, starting with the case of text storage and expanding toward a generalized structure. We called the old units files. Let us choose a new name for units that can link and overlap. I propose the term document, since text documents are often interconnected in the way we plan to permit. The purpose of a document is the same as that of a file—it is a useful collection of data—but it offers additional advantages.

Let us begin by collecting all text into a pool of dated bytes. Each byte knows when it was created. A document is a list of pointers into this pool. Conversely, each byte knows the origin and location of its corresponding pointers. (This is related to the "piece table" approach of such word processors as Samna from Samna Corp. of Atlanta.) Different documents may point into the same text pool, using the same materials in different ways. As the documents get modified they depart from one another, and their pointer lists change.

This data structure may be implemented for a variety of different functions to go to an arbitrary part of any document; to find what documents use a given string or byte, and where it falls in each; to find out whether two specific documents share a piece of text; to see the same text as it appears in two different documents. While this has certain unusual speed advantages, it also allows rapid jumps to arbitrary positions in text, since the program steps through pointers instead of text.

This facility would permit a user to reuse and rework the same materials repeatedly in different ways and for different purposes, unconfused as to their origin but always able to identify which sections are common between what documents and versions. Multiple documents and users could share material without redundant storage. This would be especially useful for boilerplate applications, where the same materials are often reworked for different documents.

Even for the lone user, this approach could bring clarification. It would be easy to keep track of the parallel maintenance of different versions, the continuing connections to the origins of each part, and differences between all working versions. Note also that the storage overhead from maintaining many documents (and saving back versions) would be reduced in proportion to the amount of overlapping material.

The payoff could be even greater in a file server for many users who are sharing material. The data would stay in place, they could participate in many documents of different kinds, and each document could evolve separately, but connections would be continually traced among them. Where the same materials and boilerplate are repeatedly used, as in law offices, such facilities can be vital.

The text pool, by the way, should contain only "pure" text, uncluttered with information about paragraphs, fonts, and the like. Such information will be different for different users and thus must be purged from the text to assure that only a clean base of shareable materials is in the common pool. Therefore, a separate set of format pointers is needed, isolating the formatting information from the "raw" data. Formats become sets of pointers into this pool.

MARKING DATA FROM OUTSIDE

This same approach may be deeply generalized. What we have found is a way to mark the data from outside, so that those applications that need to share data and the markers may do so freely, but the markers do not clutter the data for applications where they're not wanted. We really want a much more general facility, one permitting arbitrary markings in a pool of data that may be used for different purposes by different users of these data, but that do not obstruct or encumber the data for users who don't want them. This can be accomplished through a generalization of the format coding system.

We would like to mark and link data for many purposes. Markers can be used to point at specific items, to hold a place somewhere in a text or other document, and to indicate items or sections of a certain type. We may also want links between different parts of our data to show comments, to show structural interconnections, to show corresponding parts (e.g., between code and documentation), and so forth.

We will want to move markers and links out of the data but keep them where different users can use them for different purposes. Each of the different types of links and markers would therefore be kept in a separate, individual pool.

These markers and links may be of many types, but by pooling them we gain numerous advantages: we can find which ones attach to any given section of data; we can search them by type, by time of entry, by owner, and so on. Let's consider how this will work in a text system for the storage and maintenance of linked materials and arbitrary forms of annotation. Each document can be read in sequence, every use of a piece can be traced to the other documents to which it has migrated, and overlapping pieces can be coded in numerous ways without obstructing other operations.

What we call here links are easily used for database-type coding. Thus the data structure easily supports the application described earlier: coding parts of documents and searching for them so they may be seen in their living document contexts. Word processing and database applications map data to existing file structures; they grow toward each other as the programmer continues to add features, but in the process become increasingly awkward. In contrast, suitable storage facilities would reflect a deeper structure, making intermediate applications easier and specialization unnecessary. This style of data structure exerts no pressure to design applications a given way and so escapes the styles of problem analysis and divisions into simplified solutions that are fostered by conventional files.

This notion generalizes to a new storage paradigm with wide-ranging impli-

★EXTRA★

FIND CARGO LOST AT SEA!

ITT rushes aid to fleet

A California shipping agent, who used to worry about keeping in touch with a worldwide fleet of container ships by telex, called his operation "shipshape" today.

Bill Lowe, a manager for Los Angeles-based Merit Steamship Agency, Inc., added that ITT Worldcom experts had made it "much, much easier" to transmit cargo schedules.

Merit, a nationwide shipping agency, is one of the few agencies to represent a number of steamship companies in worldwide cargo transport, Mr. Lowe explained.

"We dispatch over 40,000 telex messages a month," he said. "So we really needed a first-rate communications network to keep our customers on top of their imports and exports."

ITT Worldcom solved all the sticky problems that Merit faced with its last international carrier, Mr. Lowe pointed out. "And they even cut some of our costs—by 50%!"

Service...and Savings.

Among Merit's major clients is Japan Line, which each year moves thousands of containerized shipments, inland and overseas.



BACK IN TOUCH—*Shipping agent Bill Lowe after rescue.*

ITT Worldcom created a super-efficient network between Merit's 14 U.S. locations and Japan Line's Tokyo headquarters.

This network includes customized communications software, plus ITT's sophisticated Worldbridge℠ family of networking, office automation and electronic messaging services.

Let the ITT Lifesavers go to work for you. Call 1-800-922-0184. Or mail our coupon.

ITT WORLDCOM. WE COULD BE YOUR LIFESAVER.℠

Name _____ Title _____

Company _____

Address _____ City _____

State _____ Zip _____

ITT World Communications
Dept. CR-A, 100 Plaza Drive
Secaucus, N.J. 07096

COMMUNICATIONS SERVICES VIA ITT

©1986 ITT World Communications Inc

CIRCLE 43 ON READER CARD

Using software based on today's file model is comparable to having to deal with a desk clerk in order to make love.

cations. I believe it is a unification that can exactly represent the intrinsic structure of all data. Though it is not generally recognized, this representation is needed in all fields and applications, not simply computing. Engineering, law, medicine, computer science, art history, entomology, and intelligence work have the same need to represent linkage and origins of data, commonality between documents, and historical backtrack. This is true for all types of data, including text, graphics, business data, and scientific data. Without these facilities, I submit, there is no way to build the kinds of features that the true paperless office will require, eliminating the debris of loose and lost files that are accumulating everywhere.

On-line publishing so far has sold chunks of text too big to be useful—the customer is forced to buy a side of beef when all he wants is a hamburger. Users need instead to be able to browse on-line through forests of interconnected material, paying only for the small pieces as they go. The approach I have described is excellent for on-line publication since everything's origin is identifiable down to the byte level, and royalties can be calculated accordingly. When a document is read out by a user, the owner of each byte may be minutely rewarded from the user's account with no intricate mechanism, just as the user of a jukebox automatically pays royalties to a song's owner and performer. We may even envision a new literature where linkage, intrinsic everywhere, now becomes a part of the structure of the writing itself.

PROBLEMS, CHAOS, AND GROWTH

The problem of massive digital archives is growing at an extraordinary pace. There is an increasing chaos of different programs and formats for the millions of megabytes that our society is storing for future historians and others. It is not known whether the software used to produce some of these data will even continue to exist, let alone be maintained, when historians want to study the material it created. Even next year, the boss may not be able to find out what his employees have created. There are word processors and spreadsheets of every conceivable kind, there are forests of graphics and 3-D shapes produced by a variety of systems, and much more. The increasing need for archival storage demands that a universal archival form be found to which all existing data structures and arrangements may be mapped. What is needed is a stable and generalized form of storage on which persons of goodwill can agree, leaving out nothing that is represented in any other system. I believe this can be achieved.

Separate, isolated computer files have their limits. What will replace them will be an automatically expanding storage structure. We need a stable indexing scheme that works across node boundaries if we are to have an indefinitely extensible network of instantaneous accessibility. And that is what must be. ◉

Ted Nelson, a self-styled top-down idealist, has been designing interactive systems for personal computers since 1960. His ideas are further explained in his book *Literary Machines* (available from Project Xanadu, 8480 Fredericksburg, Suite 138, San Antonio, TX 78229).

The Marketplace...

JOB MARKETPLACE, SOFTWARE SERVICES, BUY, SELL, LEASE

SOFTWARE CONVERSION SOLUTIONS

Dataware provides the software translation system for your complex conversion problems. Over 18 years of conversion experience has resulted in thousands of satisfied customers, worldwide.

- **COBOL to COBOL**
  Circle No. 500
- **AUTOCODER/SPS to COBOL**
  Circle No. 501
- **EASYCODER/TRAN to COBOL**
  Circle No. 502
- **BAL/ALC to COBOL**
  Circle No. 503
- **DOS/ALC to OS/ALC**
  Circle No. 504
- **PL/1 to COBOL**
  Circle No. 505
- **RPG/RPG II to COBOL**
  Circle No. 506
- **RPG/RPG II to PL/1**
  Circle No. 507
- **DOS to MVS**
  Circle No. 508

Dataware offers services & software to meet your needs. For more information, call or write today.

Dataware, Inc.
A Computer Task Group Company CTG

3095 Union Road
Orchard Park, NY 14127-1214
Phone: (800) 367-2687
TELEX: 510-100-2155

COMPUTER SCIENTIST. As part of Software Design Team supporting development of new multiprocessor scientific computers, develop components of highly complex operating system software using "C" language. Must develop software emulator for hardware system architecture; emulate and test hardware vectorizing capabilities; implement and modify operating system functions such as stack backtrace, which interface directly with hardware; and must implement existing system tools such as debugger to support software development, installation and testing. 40 hrs/wk. 8 to 5. $32,175/yr. Must have M.S. degree in computer science. Degree listed above must have included coursework or thesis work in: (a) "PASCAL" or "C" computer language; (b) compilers; (c) operating systems; (d) advanced data structure; (e) assembly language programming; and, (f) X-editor. Send resume to Illinois Job Service, 401 S. State St., 3 South, Chicago, IL 60605, Attn: J. Haight, Ref. No. 6592-H. An employer paid ad.

FREE...

layout and typesetting of your copy with your Marketplace ad.

Call Liz Martin at (312) 774-8115, ext. 665 to reserve your space in the next issue of DATAMATION.

DATAMATION®