Microsoft

TO: Darryl Rubin, Bruno Alabiso, Neil Konzen, Martin Dunsmuir, Mark Zbikowski, Mark Cliggett, Anthony Short, Paul Maritz, Nathan Myhrvold, Tony Williams, Chris Larson, Brian Berkowitz, Tom Lennon, Phil Barrett, Yaron Shamir, John Ludwig, Gordon Letwin, Charles Simonyi, Greg Whitten

FROM: Bill Gates

DATE: September 6, 1989

RE: Data Storage

One of the fundamental issues in computer design is how data is stored. This drives the access technique and determines how the user is to get at the information he is interested in. I try to raise in this memo a number of questions about our company strategy for data storage..

Today's DOS file system is the top level container for PC information. It is a hierarchical name space with the volume names being at the top level. Names are 8.3 and a limited number of attributes are stored with the file. We have added EA (extended attributes) to all OS/2 file systems, including FAT and we have added long names to the HPFS (high performance file system). We will add EAs but not long names to DOS. We solved the issue of EA diskette being used on old versions of DOS by putting the EA data into a funny file (or pretending to). If you copy or create a new file under old versions of DOS the attributes are lost. EAs are set or cleared through DOS calls - no control can be imposed.

Attributes for .EXE files are often stored in a published format called resources. These can be random accessed and they are available in Windows and OS/2. They are not available for data files (that is, if resources were added to a data file the applications that use the file would break). The icon of an executable is one example of something defined in a resource.

Applications often store attributes in their own private format - for example, WORD has a document summary dialog (both PC and Windows). This information is stored in a format that we publish to the outside world but that is not used by any other product. Likewise people like WordPerfect store attributes in their files in special forms - most of these forms are publicly known but there are a lot of them. I don't think Magellan or other OS viewers presently know how to display this information. Magellan knows how to find plain text and display information for popular formats.

CONFIDENTIAL

EAs are not indexed today so the only way to select a set of files by EA is to enumerate all candidate files. EAs are stored somewhat inefficiently. I don't know if updating an EA changes the file date - I don't think so. Standard EAs have been defined by the OS/2 group for author, type and other things. At present people are inventing new EAs for attributes the operating system is interested in fairly often. The shell interface to even look at or define new EA

values for a single file is weak but available in 1.2. There is no facility in the 1.2 shell to display a lot of EAs at once or to look something up by EA. Even though the search would have to enumerate all files these capabilities have to be added. Likewise we need to get our applications to store the document summary information in the EAs. There is a question about the attributes we summarize that no standard exists for - we could go too far in using EAs if, for example, we put statistics about the file in EAs.

Links. This name refers here to a file referring to another file. As far as the file system knows only a single unique link exists for each file and that is in its directory. However, applications often store file names in files. This raises some interesting problems - what to do about volume names? How to know if the file is moved, renamed or deleted? More commonly, if the file is updated, how does the guy who is linked know this and immediately update himself and pass that along to people referring to him? How to get a path name that works across the entire network? If the file is sent somewhere, how to know to take the file along with it? A fundamental question in links is deciding if a file can know everyone who is linked to it. In many cases this is feasible and nice. In many cases it is not feasible. One "solution" is to give files IDs (unique IDs) that can be used to find a file or at least verify it is the same file. We need to solve this problem very soon. Without links in the file system everyone is kludging their own structure at the application level. A user needs to have a universal way of seeing links. A number of proposals have been made but I don't know where they stand. Links are not planned for any operating system release I am aware of. I would like to get together with Bruno, Darryl and other interested parties and figure out how we get concrete about this plan. Even standardizing an EA that would list the files linked to a file would allow the user to see the links easily and some programs like the shell to update them. I believe we need a short term pragmatic solution to this problem and a long term solution. I think we will want to list this information in an EA eventually anyway, so asking people to do it now should put them on a path to the future.

HP NewWave. NewWave is an innovative piece of system software. There aren't many around so we should learn from it. One of its key innovations was its OMF that tracked relationships between objects. The only relationship it understood was containment so it had an in-memory tree structure showing who contained who. Actually since a file could be contained in multiple places it was a directed graph. It "solved" the issue of notifying everyone when a contained object changed. It did this by sending messages - in fact, NewWave would often send messages when it wasn't necessary. They dealt with issues like changing size, deleting, etc. Objects did not have unique names and there was no user interface for examining object relationships. They didn't solve the networking problem. Their biggest problem was their lack of integration with the file system. NewWave objects were all contained in strangely named files. Other NewWave areas of innovation included: (a) A better shell. We do as well in this area in Windows 3.0. (b) Defining a separation between user interface and action. This is a very important idea. Like all of the ideas in NewWave it is not original. The beauty of this is that it provides macro record, play and user interface redefinition for free. I am surprised Microsoft has

moved so slowly to do this. I view it as part of Greg and Tony's mission mid-term to help us do this. (c) Another key NewWave feature was the messages they used for containment - display, print, make new instance, resize, etc. They don't require metafiles - they call the owner for display and print. It is a high-priority short term item for Greg and Tony to define messages like this for Microsoft. The benefits for third party add-ons to spreadsheets, documents, presentations and forms is significant. We should include issues like user interface (does it show up on the insert menu? Does double click invoke a new window and the 'edit' operation?). (d) NewWave also attempted to deal with defining a central control language but I never saw any progress on the tough obvious issues of datatypes or user intention (See Halbert's PARC paper which I got from TonyW for an excellent discussion of this).

The file system will be used by our networking products to store not only classic type file information but also information about entities such as users, printers, gateways and any other network objects. The benefit of this is that the file system distribution and security capabilities are then leveraged for this information as well. I applaud this because of the unification, however I think it will show that our data storage strategy has significant holes in areas like storage efficiency, linking, logging and indexing.

SQL Databases. SQL databases are becoming a strong standard for moving data between machines. In fact, machines that will never have consistent file systems will be communicating SQL data quite freely. A key element of IBM's SAA strategy is to unify SQL across all of their platforms and eventually provide distributed query (read and update). Microsoft is building an API standard at the workstation to try and hide as many SQL differences as we can (Kyle Geiger and Bob Muglia in Adrian King's group are in charge of this effort and are seeking to enlist Lotus and Oracle as key partners). SQL tables are excellent for storing information that is uniform for all items (records) and that is low level (string, date, number). Instead of dealing with types of a higher order instead you use a number or string field as an identifier and do EQUIJOINs against the table that stores the information about the identified type. For example, when an employee is in a specific department number, you get the name and other information about the department by joining with the department table rather than following a pointer to the pointer - there is no 'department' type. In other words, SQL records do not store anything but numbers, strings and dates. Higher order interpretations - for example, knowing that the number is a department number - are only known to the application code or data dictionary. The benefit of this is that finding and updating all pointers is never an issue. Joining would seem to be a very slow way to get information but SQL databases have been optimized to use B-tree ISAM and caching to achieve very reasonable performance. I am surprised that no SQL database cheats and uses a pointer stored from previous EQUIJOINS as a hint (this would not affect its behavior at all) but as far as I know nobody does so. Older databases called hierarchical or networked used pointers as do a new type of database called object oriented (the word unique identifier is often preferred to pointer because pointer implies a low level machine address). The issue of using a pointer or a string to identify attributes (same as column or field) that are high level objects described elsewhere should not be a major

one. SQL is great for business information because there tended to be only tens of entity types - like customer, order, employee and hundreds to thousands of each of these. Some issues like special employee types (do you make another table with just the special information?) or grouping (say you sell products in a bundle and you want queries to recognize that the office product is actually a form of the word processing or that educational SKU units should be included) get messy but there are ways of dealing with these issues. I believe a very high percentage (over 80%) of the data on enterprise level machines (mainframes) will be stored in an SQL form so SQL capabilities are critical to workstation software and SQL performance is critical to mainframe quality. SQL tends to be weak when there are lots of types of entities with lots of relationships and not many of each of them.

IBM's strategy is to use SQL for distributing network information. They have "addressbook" code for every one of their platforms. They will use SQL to distribute the data. They will use SQL front ends to update and browse the information. This is a direct contrast to our strategy (there are two ways to bridge them which I discuss later). IBM's addressbook defines the standard columns for the user entity. This has already been done (have we looked at it?). Microsoft's current SQL strategy is not strong. We rely on an outside vendor. We have very limited local SQL capabilities. Our vendor in not SAA compliant. We are considering getting closer to IBM in this area but that is not a simple thing to figure out. We don't have a plan to support SQL on our file objects.

Another storage format is to simply define an ISAM. Apple has done this with System 7. I am surprised it didn't receive more notice. Microsoft has an ISAM that is a subset of its Omega file format. The size is quite large even without the multiuser locking capability. The migration of ISAM oriented code to SQL oriented code is not well defined. You could say we have ISAM already with our directories since we do key look up but we have never exported an ISAM service based on our directory handling code. I wonder if the HPFS ISAM solves concurrency issues and should be exposed at some point.

Many applications come up where there are a diverse set of objects (=entities) that have to be dealt with. An example is storing engineering information. There is no top level container like a document that works well for engineering data - several forms of access are needed. Unfortunately SQL does not work well for this. One small reason is that SQL databases often don't handle variable sized records very well (images, drawing, audio, documents and many other objects are very variable sized). SQL products are being updated to do this better - primarily by storing the variable size information separately (like the DBASE memo field) but they limit the ability to use these "fields" as queryable values. This still doesn't address the need to have inheritance in the record objects. The whole issue of variants is a very difficult one. Take for example a mail database. Some messages are a few simple fields like to, from, text. Some are room reservations and contain a date. Some are project status and contain a lot of interesting fields (amount slipped this report). A user with a lot of messages wants to be able to index all of the messages with a common field - very easily. There are two products that do

this quite well. One is the Lotus IRIS Notes project and the other is MIT Object LENS. I recommend everyone read about these since they solve some of the interesting problems. The Object LENS article is very well written. Notes does an excellent job of using logging to know how to incrementally update its indexes based on everything that has happened since the index was last updated. An index is not restricted to be based on a field value - it can be a formula of several field values or a simple formula like UPPER (field). Also field values include lists (a great convenience, but you have to be very careful to map things if you want to preserve a relational model, which they don't). Notes uses this time based log to also do efficient replication. I believe they replicate at the note level because they feel notes will be reasonably short (that is they never send just a piece of a note update like the one field value that changed). Notes punts on allowing attributes to be anything other than low level types. Object Lens, on the other hand, allows for object valued fields and does some excellent user interface work on top of this. They don't deal with the the issues of distribution. One of the impressive things about these systems is that once you have forms and indexable variant types and an interface to deal with them most of the special case stuff is no longer special case. All the applications structures can be viewed and updated through our interface. For example, the NOTES addressbook is just a set of notes indexed by user name. The difficulty of dealing with variant objects prevents database tools like DBASE from becoming the central metaphor for storing and finding office information.

There are several database activities aimed at dealing with the issues above. Ullman's books (Volume 2 just came out) provide the best hardcore foundation. Dyson's newsletter has talked about the startups working on this problem. Our favorite is a company called Object Design who will be visiting in the near future. IBM has two efforts going internally. One is called the Repository. It is a huge project that has moved around from lab to lab. It is sort of an object database but since they built it on standard SQL engines I don't think it will ever be very good. It may ship in the next six months, is targeted to the mainframe only, and focused on system configuration, software development and network configuration. We are interested in the last area - at least to agree on entity definitions (although many of the interesting ones are in their addressbook already). Today the Repository is in Toronto under Wheeler. Another more interesting project is called IRF and is being handled out of Germany (Sinfeldan or something like that). There was a group doing a document management library and another group doing an engineering document library for CIM and they got together on this project. It is very interesting because they claim it is object oriented. We should learn more about it. I asked if they would store third party application formats in the library and have standard messages for text enumeration to do content search . The people we were talking to had no idea. The issue of being forced to totally know the format of something or knowing how to invoke code to look into something is critical to making the library stuff work.

Our long term direction is to make every directory an object that handles today's operations and new operations. We need to standardize a number of these extended operations - for example, ISAM lookup. We could put in a

simple emulation for this today - enumerate for the set that matches. I think the key to making all of this work well is logging. I don't understand our staging plan.

Optimizations within a directory will depend on building new data structures (indexes of various types) that need to be brought up to date. Rather than us deciding on a specific structure and deciding whether to update on change or update when the index is used or some other time I would like to see us standardize some hooks to be used for logging. If a directory can see all of the the operations on it then it can log them. Perhaps we need to standardize the log file - although this raises a tough issue of granularity. Do we just log that something about a file has changed or do we log what a specific attribute has changed? I know our networking group has plans to do some of these things. One of the grat things about logging is that it allows anyone to update their standard structures. For example, say an SQL addressbook on a mainframe wants to shadow changes made in our user directories - copying the whole thing might be too expensive. Conversely of the SQL addressbook keeps track of dates for each record we can query all of their changes in a recent time period. Another example is a free text index on a set of files. Knowing that the content of one of the files changed allows you to know the index is out of date. One problem with a log is that it can get very long and redundant. At some point (fairly often) all of the log users have to say, "I saw all of the data up to a certain point of time" so the log can be pruned. I wonder if we need to own SQL querying code that becomes standard for directory operations at least on servers sometime not too long from now. If we supported logging and included the ability to scan the log to update B-tree index files on attributes it would be a step in this direction.

Microsoft is attempting to make the file system do everything - store objects, manage fast and general queries (including SQL), replicate, be secure, fast "join", compact storage of table like information, etc. This makes it sound like an object oriented database, which it will be over time. However, I thing we need a rational plan for going about this. For example, we don't expect other systems to duplicate what we are doing and yet we need to connect with them. I don't even know if our file system is "concurrent" enough. At what level should multi-user updates get locked out - should each attribute be exclusive? I don't understand how attribute types like "USER" or "FILENAME" automatically get a 'class based' interface for user selecting and updating. What utilities are needed? I would like to evolve the DOS/Windows file system as well and that has to fit into the strategy. Likewise the use of resources has to be unified.

Concretely our application strategy requires resolving the link issue. Our network strategy requires solving all lot of these issues. If we stick 10,000 users into our currently structure, our space and speed will be completely unacceptable compared to the straightforward SQL approach.

All this memo has done is raise issues. In the next few months a strategy memo for how we will address these issues over a period of years would be very valuable. I will work with all of you to see how we discuss these things and come up with this.

WHG/jlg