NAMES OF FILES

We started the section with the assertion that a file must have "a name of some sort" : what sort of name ? Anything will do if it serves to select one from the set of files available. The most common types of name are text strings and icons. There are constraints on the sort of name which we can use, determined by what we want to do with the file names – icons are not easy to write into programmes.

What makes a file name easy to use ? That depends on what we want it to do. In the early computer systems, names weren't expected to do anything except name, so there was no special structure, and trivial names were acceptable. (The usual constraint was on the length of the name, which was commonly very tight in order to save limited and expensive memory; in the IBM 1130 Monitor System, names could be up to five upper case alphabetic or numeric characters in length, because they had a way of fitting five characters into two 16-bit words – and that left two bits for the file attributes !) While that worked up to a point, most people soon found it necessary to devise some scheme to encode the project, version number, and type of file into the limited space – so MX92S might be the source file for the second version of the ninth programme of the matrix package. There was no deeper structure, so once you had a couple of projects, and you were sharing the (tiny in capacity, but physically big and rather expensive) disc with two or three other people, interesting things could happen. Clearly, trivial names are not enough; we want a name to include some information about the file, and it would be a significant advantage if we could express the information comprehensibly.

Consider as an example the file which contains this text. When the text was first written in 1991, the file was new. It was prepared on an MS-DOS system, where it was called `files.340`, without any very significant surrounding structure. Later, it moved to a Macintosh, where it was called `340 Files`. Two other copies were made for safety, one on the Macintosh disc for ease of access, and an archived copy on a separate disc elsewhere. It's worth remarking that nobody bothered about the icons of the Macintosh files, so they looked just the same as the icons for any other file produced by Word; while it's not too hard to pick a sensible textual name for a file, how do you choose an icon which conveys at a glance the idea of the files chapter of the 340 lecture course notes ?

Within the Macintosh system[SUP4], we use the name for description and for classification. The file itself can be classified under some such headings as { lectures, 340, files, 1991, current }; for the archived copies, replace "current" by "archived". Two of those items were chosen to put in the file’s short name, and for the time being the rest stayed in the author's head.

If that's typical, and we think it is, then we want to design our file names to help in these descriptive and classifying functions. Various ways of doing this have been tried, mostly (for historical reasons) to do with textual representations. To accommodate the sort of classification we suggested, we must provide for long names.

With text, we can use names which just collect together the different descriptive terms which we want to associate with the file. As `lectures,340,files,1991,current` is a shade cumbersome for constant use, it helps if we can find a way of splitting the name into pieces, which leads to the idea of compound file names. Experience suggests that components of file names of up to at least 15 characters are useful; a fairly common limit is about 30, and is hardly ever a constraint. You should be able to compose your file name from any sequence of printable characters; there should certainly be some provision for separating parts of the name, as in `files.340` or `340 files`, though in practice it is necessary to reserve at least one character to separate components of a compound name. (Unix uses a stroke (/); MS-DOS uses a backward stroke (\ ); the Macintosh system uses a colon (:), and won't let you make file names containing colons, which is inexplicable if you only use the system at the icon level.)

We emphasise that the discussion above depends on just what you think the file names are supposed to do. Our long names will work for any context – and typically the context includes everything the file's owner does. The same naming conventions extend to our other lecture courses, our research, our administrative duties, and anything else, so everything can be encompassed by the same filename structure.

We also emphasise that *we have so far said nothing about how these names are to be implemented.* If you think that you know how a file called `lectures/340/files/1991/current` must be stored, forget it. There is no necessary connection between the file name and its implementation; we could, after all, simply regard that string as a 31-character name, and use the IBM 1130 technique leaving rather more space for the file names. For the moment, we are discussing what sort of name we want, and we'll get to implementation later. You will probably not be surprised when we do, but try to avoid preconceptions for the time being.

An alternative view is that the names are only intended to identify files within a limited context. It can be argued that it isn't the job of the file system to cope with the way people organise their lives, so that some of the descriptive terms – perhaps `lectures`, `340`, and `1991` – are really someone else's responsibility. If you accept that point of view, then you will work out your own ways to organise your files by activity (perhaps using categories such as `1991 340 lectures`), and within these activities you will be concerned with just a few topics (the sections of these notes), each of which might appear in a few different versions (`current` and `archived`). File names as handled by the system therefore now need fewer components, so it can be designed to handle, say, only two-component file names. A fixed-size file name simplifies the design considerably – you don't have to allow for large buffers in case someone uses a 23-component name, and you don't have to provide recursive procedures to analyse the name. The file of our example might simply be called `files:current`, and it's up to you to look after the rest. You will, of course, need a separate system account for each of your activities.

Is that better ? It certainly works – many widely used operating systems were based on two-component names, and were successful. If you want to know where you're spending your money, it's a good system, because it becomes comparatively easy to tell what activity people are engaged on at any time. Of course, one can then ask whether it's the job of the file naming system to help with the accounting.

Do not dismiss these systems as primitive; if nothing else, they illustrate an important point which it is worth remembering when designing any file naming system. It is that, while working on a particular task, it is likely that several of the descriptive terms will apply to all the files you use. That means that it would be sensible to provide some means in our system whereby we can fix the values of several such terms for the time being – in the example above, `lectures`, `340`, and `1991` – so that we didn't need to keep specifying them all the time. When we discuss the implementation of the file system (*DISC FILE SYSTEMS*), we shall see that this is usually possible.

With icons, there seems to be no compact way to present the full name. The closest equivalent to the long textual name would be to present all the relevant icons in some sort of frame; that would be less than useful unless you always invented a new icon for each of the items. (A collection of the Macintosh icons for folder, folder, folder, folder, and Word would not be very informative !)

We haven't discussed whether or not we want the order of the components of the long name to be significant. That depends on how we want to use the files. If all the classifying properties are seen as equally important, then there is no reason to impose any sort of ordering. On the other hand, if some of the properties identify differences in the way we use the files, we might want to emphasise them in some way.

Taking the `340 files` example, the most significant difference is between the current and archived files; we very definitely don't wish to mix them up, so this distinction is very significant. Similarly, it's convenient to think of the 340 files as a group, distinct from files connected with other courses; and so on. Another way to see whether order is important is to ask what sort of file groups make sense. "All the archived files" certainly makes sense; they form a quite distinct group, kept separate from the current files. "All the 340 files" is useful. "All the lectures files" is a lot less useful; files concerned with different lecture courses have little connection with each other. "All the Files files" is close to nonsense; even if there were more than one Files file, the relationship would be remote. (Recall that the `Files` file is a file containing a discussion on files.)

In this case, then, there's a strong argument for order, and an ordered name something like `current-1991-340-lectures-files` is appropriate. This seems to be the more common case, but there are a few instances in which the classifying properties seem to be much harder to set into any significant order, when an unordered name would be better. So far as the interface is concerned, a file could be identified by listing its classification terms as a set, as illustrated above; it isn't obvious how you'd do it with icons. It would not be hard to construct a file system which worked with unordered names, but we know of no example. The ordered name is universally (?) used in practice; it is often called the *pathname*.

This does not resolve the problem of what to do when the ordered classification becomes inconvenient. Suppose, for example, that a student called Peter is working on a topic in Artificial Intelligence. It is quite probable that there will be files which could equally well be indexed under "Peter" or under "Artificial Intelligence", and we would really rather like them to appear in both. In this case, a set-based file name would work very well, but as we have seen such systems are uncommon. If the names are ordered, though, the best we can do is permit a file to have more than one name; we could then call the file in question `...:Peter:AI` and also `...:AI:Peter`. How to do that is an implementation question (see *FILES IN THE SYSTEM*). It isn't specially difficult to do once we decide that we need such a facility, but it is certainly an added complication, and absolutely doesn't happen automatically. Perhaps there is something to be said for the set-based name after all.

We have discussed only one way of classifying the files. Several systems or programmes impose restrictions on file names by enforcing their own classifications. For example, it is not uncommon to find that all your Pascal source code files must have some standard suffix, such as `.pas`; the constraint might be imposed either by the operating system or by the compiler. That might, or might not, fit in with the way you want to organise your file names; whichever, it's untidy, because it mixes something more properly represented as a file attribute with the file name.

Summing up these remarks on file names, then, we find that their primary functions are description and classification, which gives us two topics to discuss.

Description is the essential function of each component of a name, and the system should establish conventions which ensure that it is easy to design the components to perform their descriptive function effectively.

- *Conventional textual names* have worked well for a long time, and still do. They are extremely versatile; they can be made descriptive; they fit into programmes; they can be entered in instructions without assistance from the computer; you can sort them into order; they can be written down easily.

- *Icons* are newer. They are claimed to be "more intuitive"*. (Which did we do first : language, or rock paintings ?) They don't work without a computer; they are quite hard to make; you can't sort them. In the Macintosh implementation, one icon can hide another, but – unless it was a deliberate decision to permit such hiding, which is hard to believe – that's just bad engineering, and not a criticism of icons as such. (The Macintosh even provides a special feature – "Always snap to grid" in the Views control panel – which *guarantees* that icons will coincide if you put them sufficiently close together !) It is very easy indeed to find bad implementations of good ideas.

  In an interesting experiment, the time take for people to find files using three different sorts of icon were compared[SUP5]. In each case, a file was represented by

---

* - It would be good to believe that all who use this phrase recognise it as an example of the figure of speech called transferred epithet. An interface can no more be intuitive than it can be courageous or sinful; such attributes are reserved for people. The distinction is important, because it seems to be thought that you can make an interface "intuitive" by following certain rules about interfaces. As intuition is in fact a human characteristic, you can only make an interface which is easy to use by taking into account attributes of people. That's a lot harder, and it gets us into areas like semiotics and system genies where computists can feel rather uncomfortable.

both an icon and a file name, but the icons used in the three cases were blank, a specific picture identifying the file precisely, and a picture identifying a category (like the "Word file" icon). The average search times for those cases were 7.5, 10.5, and 12.5 seconds respectively, with rather large variances. One would want more evidence before drawing firm conclusions, but the result is suggestive.

In a sense, it's too easy to be critical about icons, but they are really only meant to do one thing : appear on a screen as part of a user interface. It's unfortunate that they're often promoted as the answer to all your computing problems. The Macintosh system uses a textual name for its internal transactions – we don't know of any system in which files have only iconic names.

Classification is a function of the name as a whole. We have already discussed this at some length, and noticed that we might require an arbitrary number of components to give a satisfactory classification.

- We can associate *textual* components by concatenation – indeed, there is no other practicable way to do it. Further, that is how we usually describe objects, in text or speech. We readily understand phrases like "the large green heavy stone ball" without feeling the need to combine the descriptive words in any way (the lghsaretreaogevnenye ball ?) and, interestingly, without attaching much significance to their order. We could therefore easily adopt the convention that the order was unimportant; in practice, that isn't done, but it's worth making the point that the emphasis on order is purely conventional. It is correspondingly easy to think of the whole concatenated string as a unit.

- *Icons* adapt themselves less well to the demands of any classification system which is not based on hierarchy. There is no operation corresponding to concatenation of text strings to yield another text string with which one can compose an arbitrary number of icons into something which behaves just like an icon. While other arrangements might be possible, the universal convention is to depict some icons as *containing* others. For example, the last part of the full name for the file we used as an example earlier is `...:Current Documents:340:340 Files`; opening the class represented by an icon named (!) `Current Documents` causes a display of several other icons, one of which is called `340`; opening the class represented by `340` causes a display of several other icons, one of which is called `340 Files`. Here, there is no way of escaping the sequence; and there is no easy way to specify the whole name at once using the graphical approach.

COMPARE :

Lane and Mooney[INT3] : Chapter 12; Silberschatz and Galvin[INT4] : Chapter 10.

REFERENCES.

SUP4 : *Inside Macintosh*, volume 4 (Addison-Wesley, 1985).

SUP5 : M.D. Byrne : "The misunderstood picture : a study of icon recognition", *Sigchi Bulletin* **34#4**, 37-38 (October, 1991).

QUESTIONS.

Is it the job of the file-naming system to help with the accounting ? ("Yes" or "no" won't do – give reasons !)

How would you implement a non-hierarchic system, using a set of characteristics to identify the file ? Is it really hard to manage with a graphical interface ?

What other aids for identifying and classifying files can you think of ? How could they be used in practice in an operating system ?

Is it really impossible to sort icons ? How does a Chinese dictionary work ?

Is it really impossible to combine icons as we combine the components of textual file names ? Icons which are different in size and shape from others would be a nuisance, and to try to compress several icons into one would render them all incomprehensible. Would it work to display all the components cyclically in turn ? How would the system know how many items to include in the cycle ?

How could you account for the differences in time taken to locate files using different sorts of icon ?
