WORKING NOTE AC130

Alan Creak
2000 December 20

STANDARDS FOR INFORMATION FILES

An attempt is made to define standards for the construction and formatting of files forming part of the information database.

Since I started work on the handbook enterprise in 1997 much has happened and many things have changed, but some things at least have settled down and have worked well for quite a long time now. One such thing is the style of layout and the conventions I use with the data files, but I've never defined them before. Now I shall try to do so, with some notes on the evolution of the items discussed and the reasons (where there are any) for choosing the current set of "standards".

In practice, they are pretty solidly standard. There's now a lot of software built round them – over 100 programmes (mostly shell and awk scripts) which all depend on the conventions to some extent. This gives a measure of my satisfaction with them, and also of the difficulty of changing them.

I am sure that other conventions could be defined, and I'm quite happy to think that some of them might be as good, or even better. My main claim for this set is that they're good enough; they are easy to use and easy to understand, they work reliably with both awk in Unix systems and Word macros with Microsoft Word (and presumably with much other software), and they are independent of any manufacturers' software.

BACKGROUND INFORMATION.

Well, there isn't as much as I'd like. In particular, there's hardly any which is unambiguously about the system I run at the moment; there are several working notes, but this is the first which describes the machinery as it now is, running almost completely in Unix. While many things haven't changed much, or at all, in the transition from Macintosh and Word, it isn't very clear what has and what hasn't.

The initial design of the system, and a diagram of the overall process, appears in a fairly early note[1]; while some of the detail depends on the implementation in Word, and has changed, the overall structure has survived remarkably well. An attempt to extrapolate this to the future[2] is also illuminating, though it has proved rather less enduring (as I foresaw – see the title). This note also defines some vocabulary; I have tried, not always successfully, to stick to that vocabulary, and shall continue to do so in this note, but some of it is based on assumptions current at the time and no longer very realistic. Then there's some more anecdotal information[3] based on a further year's experience of the system – but this is still in the Word era.

There is a lot of information about the transition to Unix, but it is all in my working logs. These records are undigested, and probably not very digestible unless you did the work recorded, which you didn't (unless you are me). There hasn't been time to sort it out into useful order.

THE FILE STRUCTURE.

All the system information is held in unformatted character files. This format is chosen because it is the only format I know which will always work, which can be read and written anywhere, which can be moved between machines without (too much) trouble, and which is not dependent on anyone else. Having been caught before, more than once, by writing my software depending on assurances that certain commercial software, hardware, formats, or whatever will *certainly* continue to be available in the future, I am not going to be caught again.

The files, therefore, have no special built-in structure, but that is not to say that they are amorphous – only that the format is explicitly denoted by mark-up symbols expressed as character strings. The files are composed of three sorts of syntactically significant object :

Mark-up symbols : These look something like |!|*something*!| or |!!*something*!| – details might change from time to time to suit software requirements, but these forms have proved sufficient so far, and any necessary new notation will be recognisably in the same pattern. (The "!!.....||" notation which I used in the early reports[4, 5, 6, 2] was less satisfactory; both !! and || appear in text

sufficiently often to be a nuisance. I've had no trouble with the current notation so far.) The symbols should be comprehensible, because the *something* is intended to be not cryptic.

Those beginning |!| are *block symbols*, typically setting attributes of some sort which remain in effect until redefined; those beginning |!! are *local symbols*, commonly defining some local type style or specific textual feature.

Ends of lines : These look like adjacent pairs of ends of lines. Single end-of-line characters are regarded as white space, and don't matter – that's because many E-mail systems introduce single line ends, particularly when forming their "reply" documents from the message originally sent. Two or more adjacent end-of-line characters are syntactically significant, and mark "true" ends of lines, and breaks in the text.

Lines : The rest is just text. A line is the text between two successive breaks, where a break can be either a "true" end of line, or a block mark-up symbol. Local mark-up symbols are regarded as text at this level of structure, and are left embedded in the lines.

TYPES OF FILE.

There are files of many sorts in the system. I assert that all the sorts have sensible functions; certainly I haven't invented new sorts of file for fun. I think this list is complete, but give no guarantee. In this section I present briefish definitions; in some cases, there is further information below.

Database and **narrative** : If I had to pick one of these dimensions of classification as primary, I think this would be the choice.

A *database* file is a set of records (or sometimes only one record – see files and collections); the records are commonly expressed as sets of fields, with each field identified by a leading block mark-up symbol. The field extends from its initial symbol to the next block mark-up symbol, or to the end of the file. Except that the fields of a record are kept together the order of items in a database file is not significant.

NOTE : That's the ideal. In practice, there are still some database files in which a specific field must begin each record. These will go in due course when I introduce a record separator.

In a *narrative* file, the order is paramount, and, in general, cannot be changed without causing some externally significant effect. Narrative files are mainly used to hold formatted text – such as, for example, the handbook text. Block mark-up symbols in such files represent layout features – section headings, special styles, the beginning and end of tables, etc.

Files and **collections** of **module** files : The records of a database might be brought together into a single file, which is called a *file*, or they might occupy individual *module* files living in some designated directory, which constitute a *collection*. (You can always call a file database a database file if there's any possibility of ambiguity.) Similarly, narratives might be held in sections or as complete documents. There is no algorithm to determine which organisation is used in a particular case; it's purely a matter of convenience. In practice, that means that in cases where the records must be reviewed by different people – handbook text, course information, etc. – I use a collection, and where the information is purely for inco's administration I use a file.

Simple and **composite** : A file in a database is a *simple* file in that it contains just one well-defined sort of information. (If it didn't, it wouldn't be much use as a database.) A *composite* file contains information of many sorts; such files are usually products of the system rather than internal components. The handbook text files are examples.

Before and **after**, **working** and **current**, **archive**, and **lastyear** : These are all different sorts of file *version*. The are descended from the *working* and *reference* collections of the earlier definition[2] as the system has become more complex; they are not yet completely developed, but should be.

The *before* and *after* files are used to give some level of backup during the annual handbook developing cycle; typically, a *before* file is sent out for editing, and the edited version becomes the *after* file.

The *working* and *current* files (also not thoroughly developed) have developed as facilities have been constructed to provide orderly procedures for necessary amendments or the on-line handbook files.

The *archive* files are rather different, as they don't form part of any production cycle. They are at present kept only for the courses and people collections, and they will end up containing a copy of the latest information we have on any course which, or person who, has been in the system since records began. (Last year.) This is not maintained out of any interest in the department's history – it's a convenience, as people who, and courses which, go might come back again, and it's handy to have the information rather than try to make it up again from scratch. Finally, the *lastyear* files are not archives, except by accident; they are the files used to make last year's printed handbooks, in case we need to know. The most extensive use of the lastyear files is in constructing the HTML handbook entries for the courses; links to the previous year's entries are inserted for those courses which appeared in the previous year, and prerequisite links to previous year's files are made if appropriate.

A file for an established handbook item will therefore probably exist in four (and perhaps more) versions, and we have to keep track of them somehow. In fact, as they are all collections, it's easy : the many versions of a file all have the same name, but they are kept in different and appropriately named directories.

Character and **Word** : The distinction between files of these sorts has nothing to do with the units of their storage; a *character* file is simply an unformatted character string, while a *Word* file is formatted by Microsoft Word.

Being formatted, Word files are, by definition, not part of the system, and I usually regard them as the end product. In fact, one can get the information back into the system without too much trouble, and this notion was explored as an aid to editing the handbooks in the early days[4]. It worked quite well, but hasn't been used since because editing has been carried out on the mark-up files. This is not entirely satisfactory, if only because in practice it has meant that I have to do it all, so other means of editing are desirable.

There is no reason why other word-processing software could not be used; I have absolutely no special affection for Word, but it was there, and I knew how to write macros to drive it.

Mark-up, **HTML**, **tabular**, and **non-standard** : Most of the files within the system are structured using mark-up symbols, and are reasonably called *mark-up* files. This category includes both files written according to inco's mark-up notation and *HTML* files. Sadly, though *mark-up* files are elegantly simple and well adapted to computer processing, they are not so well adapted for people. The module files of a database collection are not too bad, because the mark-up symbols can be read as section headings, but more compact database files are extraordinarily annoying for people to read; even in the narrative files, the mark-up can be quite distracting.

There has therefore developed, almost more by accident than design, a species of *tabular* file in which the database information is presented with the fields of a record separated by tabulation characters, so that an orderly table is seen if the file is inspected with something capable of tabulating. To use these files effectively, there is a set of scripts which carry out the conversions between *mark-up* (inco's, not HTML) and *tabular* files as required; any editing can thereby be carried out using the comparatively readable *tabular* form and the result returned to the system as a standard *mark-up* file. Anything else – there isn't much, except by accident – is *non-standard*.

There are also **programmes**. I note these for reference; so far, at least, they are not transformed or otherwise manipulated by the system except as programmes, but just lie there as non-standard files. They come in three varieties : shell scripts, awk scripts, and Word macros. Shell scripts and awk scripts are

briefly documented in a README file[7]. There is some description of Word macros in earlier notes[1, 5, 6], but nothing orderly and structured. Perhaps some day ...

DIRECTORY MANAGEMENT.

The use of several directories to manage different versions of modular databases was mentioned above. This part of the system should probably be regarded as still under development, but I have what seems to be a reasonably effective set of directories at the moment, and I shall describe it as an illustration.

In the earliest days of the system, it was simple. We had a set of files which came from last year's handbook, and we wanted to change them into a corresponding set for this year's handbook. It was obviously silly to throw away the old ones before we were absolutely certain about the new ones, so I kept the two sets. These are what I would now call the *before* and *after* collections. (At the time, I think I identified them by date; the dates then spread through the system, and most of them had to be changed for the following year. Static names are better.) Then, once the handbook is printed, we can throw away the old files, and this year's *after* files become next year's *before* files.

With the appearance of the HTML version of the handbooks, this simple picture didn't work so well any more. The new feature is the possibility of making changes (usually corrections or clarifications) to the material throughout the year[5]. The difficulty did not become apparent immediately; on the surface, the only difference is that we have to cycle through the *before – after* process a few more times, and with part of the handbook instead of all of it. (In fact, there's quite a lot more to it than that from the processing point of view, but that wasn't apparent at the time and in any case doesn't directly affect the file structures.) The complications arise when we start work on the new handbook, which happens around half way through the year. Typically, the new handbook is significantly different from the old (or we wouldn't need a new one), so the files change noticeably – but now what happens if we want to make another change to the existing handbook ? If we haven't taken some precautions, the files from which we made this year's handbook might well have vanished. It is true that we still have this year's *after* files, which are now the *before* files, but that forces some change to our procedure – and if we change them what do we do about the new *after* files ?

I have no perfect answer as yet, but my best guess so far is to use two sets of files, one (*before* and *after*) for the printed handbook and one (*working* and *current*) for the on-line material. to allow one level of undo. The *current* files are those from which the visible HTML files are produced; to edit a *current* file, one first copies it to the *working* directory, then the new version replaces the original *current* file when checked and accepted.

Two other directories useful in some cases are *archive* for people and for courses – they might go away and come back again, and *lastyear* for courses, to see whether we want to construct backward links, and to make the right links. They are otherwise unremarkable.

For each collection there is also a rawmail directory. This is used when dealing with the E-mail editing process – it is the machinery for the transition from before to after, but does not otherwise affect any matters discussed in this note.

THE FILES.

Here is a list of some of the files involved in the handbook system. I am sure that the major files are here; I am not sure that the list is complete, but it's at least representative, and will suffice for this survey.

courses	database, collection, mark-up	Details of all the courses we give;
handbook	narrative, collection, mark-up	One for each handbook; the atoms of the handbook.
handbook text	narrative, file, mark-up	One for each handbook; the full text of the handbook.
HTML	narrative, collection, mark-up	The handbook in HTML; many small files.
names	database, file, mark-up	For looking up people's names, and converting them into standard identifiers used within the system.
people	database, collection, mark-up	Interests, publications, and various forms of address of all members of the department, and some external lecturers.
presenters	database, file, mark-up	Lists courses ´ lecturers, proportion of course per lecturer, who's the supervisor.
sections	database, file, mark-up	One for each handbook; a contents list.
skeleton	database, file, mark-up	One for each handbook; the table of contents, with ancillary information about layout and formatting etc.

There is no explicit timetable file, except as a tabular file which typically doesn't last long. The timetable information is recorded in the courses collection, along with the other information about the courses. I don't keep the tabular file because it often goes out of date very quickly, and when a time is changed it's easier to make an immediate correction to the timetable information in the appropriate course file manually.

MARK-UP SYMBOLS.

Mark-up symbols got into the act right from the start – they were not so much a design choice as an unquestioned axiom. This was so near the start that I had no thought of databases; so far as I was aware, managing the handbooks was simply a text-editing task. If you look at a handbook, that makes sense; while there are regular parts (particularly courses and personal information, now the courses and people databases), it is in fact all text, and editing the regular parts did not seem to be any different from editing the rest. Experience over the years with various sorts of editor, particularly the rather common sort where what you see is not what you print, had convinced me that mark-up methods were the only way to go. I was happy to start my own sort, because HTML was too limited (or too complicated), TeX was too basic, and LaTeX didn't do what I wanted. Another good reason was that if I used a unique format other people could put examples of other mark-up languages into their handbook material if they wanted to. Also, I'd already had one go at making a mark-up system[8], and had found it reasonably straightforward until I tried to get clever, which I didn't think I wanted to do.

In the first try, therefore, I invented a set of plausible mark-up symbols for the handbooks. Typically, a block symbol would identify the function of a passage of text, or sometimes a desired format – chapter heading, bulleted list, person's interests, course prerequisites, etc. – while a local symbol would prescribe desired type styles within the text. It worked reasonably well.

It soon became clear, though, that it wasn't well enough. The most pressing reason was the discovery that in fact material about – for example – people turned up in all sorts of places. There are tables of people's interests in the postgraduate handbook, there's the directory in the undergraduate handbook, and their names appear in the descriptions of course which they present. Getting everyone to check every relevant personal reference in the handbook was next to impossible. Information about courses is similarly distributed. It also turned out that if I let people edit their personal information and the descriptions of the courses as they wanted to, it was a very hard job to achieve even a semblance of uniform presentation throughout the handbook.

So the databases were born. They began as what had been the relevant portions of text from the handbook, collected together. They were therefore already in mark-up form, and a comparatively few steps of tidying up converted them into something that was a quite usable database, with the advantage that by using my own format I was independent of both commercial text processing and commercial database software. The databases were originally used as text as well – the order of fields was important, and to make the courses section of the handbook (for example) the courses database files would be formatted directly, with any unwanted fields deleted in the formatting process; now the order of fields is unimportant (unless there's some ancient software lying about somewhere).

TEXT FORMATS.

My original intention was to design a simple mark-up language; it was clear that the degree of sophistication I'd aimed to provide in my own system[8] was not necessary, which was just as well. This decision has proved to be sound, and apart from a little evolution in details the current language is quite like the original design.

Formatting is controlled by mark-up symbols at block and local levels. A block symbol controls the formatting of one or more lines (defined above), while a local symbol either causes text to be inserted where it appears, or marks one end of a string of text to be treated in some defined way. No other level has suggested itself so far.

A block symbol, of the form |!|*something*!|, always appears at the beginning of a line. As it is by definition a line separator, that's a truism, but what I really mean is that it should be preceded by an end-of-line text marker. (In practice, I have almost always presented them on lines of their own; I am not sure whether I've assumed that's so in the software. As I've regarded the current software as a mere stopgap until I can get the *real* software[4] going, there's no guarantee.) The symbols are associated either with line formats only, or with the function of a line, which usually also implies a format. In general, a symbol can appear unmodified, when it applies only to the immediately following line (|!|Heading!|), or modified by "on" or "off" to set a format over an arbitrary range of lines (|!|Bulleted list on!|, |!|Bulleted list off!|). Symbols implying functions (|!|Heading!|) often make sense only as stand-alone operators; the software checks some of these, but not necessarily all. Turning a symbol on does not turn the previous symbol off; they run on a stack, so the previous symbol takes over again once the current one expires.

A few block symbols have additional functions, particularly those to do with document structure. Three symbols are significant : |!|Heading!|, |!|Subheading!|, and |!|Subsubheading!|. Within a single narrative file, these have only relative significance, but when converted into a full document they are combined with information about the document structure from the skeleton file (see below) to determine whether they should be regarded as headings of chapters or sections, or subheadings of various levels. In addition, headings of chapters or sections are numbered in the printed handbook. (Such numbering is superfluous in the HTML handbook, because links can be made directly.)

Local mark-up symbols are all regarded as text and have no special structural function. In principle, they can appear anywhere, though obviously some places are more sensible than others. They come in three varieties :

- **type style** symbols (|!!italic on!|) usually (always ?) come in "on" and "off" pairs and do the obvious thing. (They might not do the obvious thing in HTML if they're not properly nested; the system doesn't check. I think it works in Word, though.)

- **special character** symbols (|!!break!|, |!!nbs!|, |!!paragraph!|, |!!tabulate!|) are replaced by the entities they denote. |!!break!| is a line break, not amounting to a new paragraph; |!!nbs!| is non-breaking space; |!!paragraph!| is a paragraph break; |!!tabulate!| is a tabulation mark.

- **transformation** symbols are a bit more complicated. They are given some information, and have to recast into some significantly different form which might depend on the sort of handbook being produced. There are four such symbols at the moment :

|*!!text!URL!The URL to be used!*| is used to include WWW links. In the printed version, this will come out as *text (The URL to be used)*, and in the WWW version it will work as you'd expect – *text*; text will be visible, and act as a link to the URL.

|*!!text!EMAIL!E-mail address!*| works in the same way as URL, but produces an inserted note of the E-mail address note or a mailto: link.

|*!!picture|filename|description!*| will (for a printed handbook) insert into the text an instruction to include the file filename; in the HTML version, *<img src = "filename.gif" alt = "description">* will be included.

|*!!section|identifier!*| is used to refer to a different section of the handbook. It will be replaced by (in the text handbooks) *section m.n* or (in the HTML version) an active link to the name of the section. The identifiers can be found from HTML lists for the undergraduate and graduate handbooks.

For reasons which probably have more to do with laziness than design, the system requires that mark-up symbols be presented in the "correct" case. This is certainly a mistake, and some day I shall try to do something about it, perhaps.

SPECIFIC DATABASE FORMATS.

These specifications should not be taken as definitive. Because of the textual representation of the files, it is easy to add or remove fields, and it happens as administrative requirements dictate. (For example, the courses files used to have an |!|assessment!| field which was changed to three fields – |!|assignments!|, |!|test!|, and |!|examination!| – when it was found more useful to have the information separated.)

There is no standard way to determine the fields currently permitted in a database. This has both advantages and disadvantages. It obviously contributes to the desirable flexibility of the system – as the environment changes, the databases can respond easily, and it has certainly simplified the task of developing the system. On the other hand, it is necessary that all software should be constructed to notice fields it doesn't recognise (so that they don't get absorbed into other fields), and to do something sensible with them – usually, either to ignore them or copy them. Currently, not all software meets this requirement.

The order of the fields is not significant; they are presented in alphabetical order in the lists below. This has not always been the case, but should be. Some early software was written assuming either (in a few cases) that the order of fields in a record was known, or (depressingly often) that the identity of the first field in the record was known, and that the first field would always be present. I intend to sort this out in time by introducing a |!|next record!| separator field which must lead the contents of every record in a database file – obviously, there is no comparable problem with a modular database.

I have wondered about providing some form of data dictionary in which information about currently accepted field, and perhaps other sorts of metainformation, could be kept. It was my original intention to do so, and the early courses and people files (which were database files, not collections) were given header records listing permitted fields. I gave that up because dealing with it in Word macros was a nuisance, but with the better machinery of awk it might be worth restoring the convention. For modular databases, it's even easier – a single file listing the fields and called – say – template can be provided in the directory.

THE COURSES DATABASE.

This is a collection of module files, with one for each version of each course. There is just one file for a course only presented once in the year, but for those courses which are presented in both semesters, or on different campi, several versions are maintained. While much of the information is common, details of lecturers and timetable are likely to be different.

|!|assignments!|	The percentage of assignment credit in the final result – 20%, for example.
|!|attributes!|	A selection of the words project, practical, and plussage (only the first four characters are necessary), labelling the course as a project (no lectures, individually supervised, "independent" work), an official practical course (both practical work and examination must be passed to pass in the course as a whole), or assessed by plussage (the final mark is either the specified combination of coursework and examination, or the examination percentage alone, whichever is greater).
|!|contents!|	A rather detailed account of the course content, identifying specific topics covered.
|!|copy!|	This field recognises the similarity between different versions of essentially the same course. This course inherits any details not explicitly given in this file from the course identified in the next line.
|!|description!|	A general description of the topic, comprehensible to any student who might want to take the course.
|!|examination!|	The percentage of examination credit in the final result – 70%, for example.
|!|lectureroom!|	Where the lectures will be held.
|!|lecturers!|	A list of the lecturers who will present the course.
|!|number!|	The official course number.
|!|organisation!|	Useful information about the organisation of the course.
|!|prerequisites!|	The prescribed requirements for entry to the course.
|!|restrictions!|	Courses which may not be credited if this one is.
|!|test!|	The percentage of test credit in the final result – 10%, for example.
|!|testplace!|	Where the test will be held.
|!|testtime!|	When the test will be held.
|!|texts recommended!|	Useful books which are not essential.
|!|texts required!|	Set textbooks which the students are expected to study during the course.
|!|timetable!|	When the lectures will be held.
|!|title!|	The name of the course.
|!|tutors!|	Tutors assisting with the course.

THE HANDBOOK FILES.

These are the basic components of the handbooks from which much of the text in each of the final documents is built. As narrative files, the mark-up symbols denote formatting instructions rather than the nature of the items to which they are attached.

Most of the handbook files are edited by people; the files are sent to the people by electronic mail and new versions are returned. Some files are constructed automatically from database material; for example, the various course descriptions and lists of courses are derived from the courses database, and information on academic and other members of the department comes from the people database.

There are very many mark-up symbols. Indeed, there are too many mark-up symbols; they were all introduced for real reasons at some time, but much of the detail had to do with the use of Word macros as the processing engine, and is now redundant. Another source of superfluous mark-up symbols is the other databases, all of which were originally formatted in isolation. Now they are combined before the final formatting, the separate sets of symbols are an embarrassment. Perhaps some day I shall sort it all out; in the meantime, it's inconvenient, but it's also harmless and it works. Because of the vast number of symbols, I shall not even try to list them all, so the table below is a selection which I hope will illustrate the significant points. Most of the symbols have comprehensible names, so it isn't hard to work out what they mean.

|!|Bulleted list on!|, |!|Bulleted list off!|	Lines between these symbols are converted into entries in a bulleted list.
|!|Heading!|, |!|Subheading!|, |!|Subsubheading!|	Relative structure headings which show the relative importance of heading lines within the file. These are converted into absolute symbols when the files are composed into the Handbook text.
|!|paper list table on!|, |!|paper list table off!|	Typical of brackets denoting the beginning and end of text for a table. Several table formats are defined; the differences are largely matters of detail – usually the styles to be used for the table entries.
|!|Text it!|	Simple formatting – the next line is presented as italic text.
|!|Text normal on!|, |!|Text normal off!|	Very common bracketing the whole text of older files, but usually redundant. It's a relic of the old Word macro conversion method, where the pair were used to ensure that only the useful text was converted. Still useful if you want to ensure that you're using the ordinary text format.

THE HANDBOOK TEXT.

A handbook text file is constructed from the handbook collection by guided concatenation. The guidance comes from the skeleton files, and is augmented by some judicious translation; the result is a much simpler file for the Word macros – it is essentially a sequence of lines each preceded by a block mark-up symbol identifying a style to be applied, or some action to be taken, or both – for example, |!|Chapter heading!| means "insert a page break, add the chapter number as a prefix to the following line, and apply the Chapter heading style to the line".

There is not a great deal of translation, though as time has passed since the initial preparation of the handbook text was shifted from Word macros to Unix (almost all awk scripts) more and more of the work has gone too; awk is a *lot* easier to use than Word macros. So far as the mark-up is concerned, there are three sorts of conversion which are interesting :

- All (or perhaps almost all ?) bracketing pairs of block mark-up symbols (|!|*X* on!| and |!|*X* off!|) are removed, and replaced by simpler constructs – typically unqualified |!|*X*!| symbols followed by the previously bracketed lines concatenated into a single line with |!!paragraph!| local mark-up symbols marking the necessary boundaries.

- The various Heading symbols (such as |!|Heading!|, |!|Subheading!|) are converted from the relative forms used in the handbook collection into absolute terms. Each entry in the skeleton file is marked with a level number, which fixes the position of the internal relative scale against the absolute scale. Depending on the level number, a |!|Heading!| might become a |!|Chapter heading!| or a |!|Section heading!| or a |!|Section subheading!|; introductory items are identified by level zero, for which |!|Heading!| becomes |!|Prechapter heading!|. This technique makes it very easy to reorganise the handbook by changing the skeleton file, and has proved very useful in practice.

- The local transformation mark-up symbols are converted as required. Other local symbols are left in the text so that files transferred between operating systems are simple text. The result is a very simple file which is easy to convert into formatted text with a quite simple Word macro, and which should be easy to handle with any other comparable word-processing software.

Here is an illustrative selection of other block mark-up symbols :

|!|Address!|	An indented format for addresses and similar material.
|!|Leave!|	From the people file; a specific format for a short note that someone will be on leave during the year.
|!|Message!|, |!|Message signature!|, |!|Message signature small!|	Used to format the Head-of-department's message at the beginning of the handbook.
|!|publications!|	From the people file; a specific format for a list of publications.
|!|Text indented!|, |!|Text normal!|	Two standard formats. |!|Text normal!| is assumed unless otherwise instructed.

THE HTML FILES.

The HTML files are HTML files. There are no clever tricks. HTML 3.2 is used because that's what was current when I started to construct the HTML system , and I don't know any good reason to change. With the exception of four or five diagrams, it's all text; that's not an arbitrary rule, but just the consequence of no one having sent me anything except text.

THE JOBS FILE.

This file associates members of the department with various administrative responsibilities, and with aliases which I use to refer to those responsible for various functions – as in the |!|Funcode!| field of the skeleton file.

There are two sorts of entry in the file, which is not necessarily a good idea. Most of the entries are formal responsibilities, assigned by the Head of Department and changed from time to time. Others are informal, assigned by me (usually with the approval of the HoD) specifically for matters to do with the handbook.

|!|Job!|	The job title – for example, Information Coordinator.
|!|Jobcode!|	The alias – for example, jobinco.
|!|Name!|	The name – for example, Creak A.

The name format should be arbitrary; in fact, it is constrained to be of the form shown because that's how it was on the first HoD's list of responsibilities which I received, and I've been too lazy to write a proper name-looking-up procedure. Some day I should do it.

Notice that this file gives only administrative responsibilities; lecturing responsibilities are separate, and are obtained from the presenters file.

THE NAMES FILE.

This file is the only list of people; while a set of people can be determined from the people database, that can easily become out of date as people come and (less frequently) go. It also serves as a source of information for converting people's ordinary names into the identifiers used within the system.

This file does not record any detailed information about people; its purpose is solely identification. For further information, use the names file to determine the person's identifier, then refer to the corresponding people file.

|!|identifier!|	The standard identifier – for example, alancrea
|!|name!|	The name – for example, Alan Creak
|!|others!|	Other names people might use – nicknames, short names, etc.

It would be sensible to extend this database to include other sorts of identification – for example, all E-mail addresses, the university "UID", etc.. A preferred E-mail address is kept in everyone's people file, but most of us have more than one.

THE PEOPLE DATABASE.

This is a collection of module files, one for each member of the department. The same form is used for every member, but some fields are more significant for (say) academics than for administrative staff. The module names are the identifiers, also listed in the |!|identifier!| fields within; these are strings of up to eight characters which are the prefixes of what you get by concatenating the person's preferred friendly name and surname, and truncating if necessary. (For example, I am *alancrea*.) These identifiers are easy to remember, and are used throughout the system.

|!|arrived!|	The year in which the person's connection with the department began.
|!|email!|	The person's preferred e-mail address. (The suffix "@cs.auckland.ac.nz" is assumed unless otherwise specified.)
|!|group!|	Academic, admin, or technical.
|!|history!|	Any background relevant to the job – qualifications, past experience, etc.
|!|identifier!|	A convenient abbreviation of the person's name used in inco files.
|!|interests!|	Any interests or activities which are connected with your job.
|!|name!|	The person's name.
|!|phone!|	The person's telephone number.
|!|position!|	The name of the person's present position.
|!|publications!|	A list of up to four publications relevant to the job and department.
|!|room!|	The person's room number – <building>.<number>.
|!|sort!|	A sort key.
|!|style!|	Academic title – Dr, Professor, etc.
|!|www!|	The URL for the person's World-Wide Web home page.

THE PRESENTERS FILE.

This lives in courses.mkp, which should really be called something like presenters.mkp; it is a database file which contains one record for each instance of a person presenting a course. (I might change the name some time soon, but see below.) The fields identify the course, by number, the person concerned, by identifier, the proportion of the course for which the person is responsible, and whether the person is the course supervisor or a mere lecturer.

This file has achieved a prominence far beyond its significance, because it is the only authoritative record of the courses we are supposed to be presenting. It is derived directly from the schedule produced by the Head of Department listing the next year's courses and who is responsible for them; all other sources are subject to the vagaries of people who might or might not have replied to requests for information.

There are therefore several programmes which read this file, looking only at the coursename fields, and rejecting duplicates. That's fairly untidy. An alternative, and in some ways neater, file organisation would be to permit several presenters to be identified within a single course record. I haven't done so, because it would affect the whatever-normal-formness of the database, and complicate it in certain respects – particularly in the imposition of something like a field order requirement to ensure that presenters' proportions are associated with the right person.

|!|course!|	The course number (almost always COMPSCInnnAB)
|!|function!|	"Lecturer" or "Supervisor"
|!|name!|	The person's name
|!|proportion!|	The proportion of the course for which the person is responsible.

THE SECTIONS FILES.

There is a sections file for each printed handbook, called (U)Gsections.mkp. These files are computed from the skeleton files, and are used to resolve the |!!section|identifier!| mark-up symbols; in the printed handbooks, the identifier in the symbol must be converted into a section reference, and the database contains a list of identifiers and section numbers. It also contains the title of the section.

|!|Component!|	The name of the chapter or section
|!|Section!|	the number of the chapter or section
|!|Source!|	The identifier – with the suffix .mkp. it is also the name of the mark-up text file.

THE SKELETON FILES.

These are database files in which are encoded the structures of the handbooks. There is one file for each printed handbook; it lists the files in which the handbook text is saved, defines the handbook structure by identifying which files start chapters or sections, and also identifies the functionary responsible for maintaining each file. It also includes information used by the system when composing the HTML handbooks, such as comparatively rare instances where other files should be substituted for the printed handbook files.

|!|Component!|	The name of the item
|!|Contents!|	"Y" if the item should be listed in the table of contents
|!|Continued!|	"Y" if this text is to be appended to the previous entry.
|!|Funcode!|	The job of the person responsible, as an E-mail alias
|!|Functionary!|	The job of the person responsible, in words
|!|HTML!|	Name of an alternative HTML file, if there is one.
|!|Level!|	Syntactic level
|!|Source!|	Name of the mark-up text file (less ".mkp")

TABULAR FILES.

In real life, people don't write mark-up files. There is a good reason for that: they are very difficult to read. Most of the handbook is simple text, or at worst lists of this and that, so the mark-up is comparatively unobtrusive and trivially easy to understand, but – particularly when presenting collected information – rather more structured presentation is desirable.

The most common example of such presentation is the table. My usual way of encoding a table in the mark-up text is to bracket the table with an appropriate block mark-up symbol, to separate the rows of the table by using a new line for each, and to separate the items within the rows using |!!tabulate!|. Here is a real example :

```
|!|paper list table on!|
Number|!!tabulate!|name|!!tabulate!|exam|!!tabulate!|test|!!tabulate!|exercises|!!tabulate!|note

COMPSCI702|!!tabulate!|Topics in Software
            Engineering|!!tabulate!|25%|!!tabulate!||!!tabulate!|75%|!!tabulate!|1

COMPSCI708|!!tabulate!|Multimedia and Hypermedia
            Systems|!!tabulate!|40%|!!tabulate!||!!tabulate!|60%|!!tabulate!|1

COMPSCI711|!!tabulate!|Parallel and Distributed
            Computing|!!tabulate!|70%|!!tabulate!||!!tabulate!|30%|!!tabulate!|

......

|!|paper list table off!|
```

This table was constructed automatically from the courses collection, which is quite easy, but people wishing to construct their own tables are faced with a fairly awkward job.

> *I note that it is far from an impossible job, though colleagues have protested at the intolerable burden. You make the table as a table in Word (or whatever – if Word can do it, presumably other software can do something similar), select the table, convert table to text, and replace all tabulations by |!!tabulate!|, and save as text only. And you have to put in the |!|paper list table *!|.*

In practice, very few tables are edited with the text; any explanatory text is edited separately, but the tables come from elsewhere. Some of those in the handbook – such as the example above – are synthesised from various databases in the system; others – notably the timetable – are prepared primarily as tables for use by people, and it is sensible (and more reliable) to use these tables rather than require their authors to deal with the reformatting.

Because of these considerations, and to assist with review and editing of the various smaller database files, there has grown up a convention, not amounting to a standard, for conversion between mark-up file and table. This is where the tabular files come in.

They are composed of lines of text with tabulation. Scripts are available to carry out the conversions; these are, at the moment, all special-purpose scripts, one for each case (or two if conversion in both directions is required), in which it is assumed that the set of fields, and their order of appearance in the tabular form, are known.

A good example of the use of such files is the skeleton file. It is very useful to have a copy of the two skeleton files for reference when working on the handbook, so the mark-up skeleton file is converted into an equivalent tabular version. From force of habit, I then move the file to Word on a Macintosh for display and printing, though it could just as easily be managed within Unix with a suitable display programme. I make amendments to the tabular version when they are needed, then save the file (text-only) and convert it back into a new version of the mark-up skeleton file. This process sounds cumbersome; in practice, it isn't, and it's far more reliable than trying to edit the mark-up files directly.

A useful development for these files would be the addition of a template. This would be exceedingly easy, as the template is essentially the table header; it would be marvellously efficient, as I could use the same conversion software for all and any mark-up file. The only obstacle is the absence of an accessible template in the mark-up file.

REFERENCES.

1 : G.A. Creak : *Background for a document generator*, unpublished Working Note AC115 (1997 September).

2 : G.A. Creak : *Handbook preparation : the future (perhaps)*, unpublished Working Note AC120 (1997 December).

3 : G.A. Creak : *Lessons from the 1998 handbook exercise*, unpublished Working Note AC127 (1999 February).

4 : G.A. Creak : *Designing the document factory*, unpublished Working Note AC117 (1997 November).

5 : G.A. Creak : *Making the HTML version of the handbooks*, unpublished Working Note AC118 (1997 December).

6 : G.A. Creak : *Going back again*, unpublished Working Note AC119 (1997 December).

7 : currently at inco/machinery/README.

8 : G.A. Creak : *Regal and Imperial*, unpublished Working Note AC126 (1999 April).