The Technology of the QUAD Editor, Part II

Jim Kramer
Hewlett-Packard
St. Louis, Missouri

INTRODUCTION

The QUAD editor is a text editor that was contributed to the Users Group library last year at the Orlando Users Group meeting. It has several features which make it notable and useful, the most important of which are that it texts files instantaneously and that it can undo any or all editing changes. A paper in the proceedings of that meeting, titled "The Technology of the QUAD Editor," described the implementation of these features.

In the past year QUAD has acquired many new capabilities, including the ability to maintain multiple versions of a file, to cancel the effects of the preceding command, and to compile programs. The purpose of this paper is to describe the implementation of these new capabilities.

A BRIEF DESCRIPTION

QUAD is a line-oriented editor similar to EDIT/3000 and TDP/3000. Its most important capability is instantaneous texting, and sometimes instantaneous keeping of files. A file is texted just by opening it and changes are kept in a separate work file. If the only changes to a file are modifications of existing lines, then keeping is done by posting the changes back to the texted file. Since changes are kept in a separate work file it is easy to undo any or all changes just by removing them from the work file: QUAD's UNDO command does this.

It is important that QUAD be able to find lines in the texted file quickly. QUAD starts out with no knowledge of the location of lines in the file, and must find requested lines using binary search. However, QUAD keeps a record of all blocks read during the search process and uses this record to shorten subsequent searches. The method is described in a paper titled "A New Tool for Keyed File Access (Sometimes)" in the proceedings of the Users Group's 1980 North American meeting in San Jose.

Features that are new to QUAD in the past year include the following:

1. Operating directly on changes to the work file, by means of the MODS key word in a line range. Changes can be listed, kept, and otherwise operated on. For example, "List Mods" lists all changes that have been made to the current file, and "Keep Modfile(Mods)" saves the modifications in a file named Modfile.
2. Cancelling the most recent command which modified the file. The Cancel command does this.
3. Maintaining multiple versions of the file being edited. The Freeze command prohibits further changes to the current version and starts a new current version. Prior versions can be read at any time, but not modified, by using the VERSION keyword. For example, "List Version 1" lists the first version, and "Keep Filename(Version 1)" keepts it.

TICKET FILES

The important characteristics of QUAD work files — variable length keys and data and re-use of space — are provided by a file access method which I call ticket files.

With most file access methods, the user who wants data stored specifies where it is to be stored — a record number. With ticket files the user does not specify; instead he just supplies the data to the access method and receives back a "ticket" telling him where the data has been stored. In order to retrieve the data at a later time, he must supply the ticket.

It is important to recognize that this technique gives enormous flexibility to the file access manager. The data can be put in the most convenient spot, for example a block that is already in a buffer in main memory. Within the block the record can be placed wherever there is space. With ticket files a record need not even be placed contiguously within the block — it can be broken into pieces.

Ticket files turn out to be perfectly suited to those applications in which data is found through pointers: tickets are really just pointers.

In order to make ticket files satisfactory as work files, it was necessary to implement a keyed sequential access method based on ticket files. The implementation is significantly different from KSAM and actually more powerful: both keys and data can be variable length, space is re-used, and keyed sequential access can be either forward or backward.

When a key is stored, a ticket is stored with it. The

ticket points to data. Thus storing data by key is a two-step process:

1. Store the data and receive a ticket.
2. Store the key and the ticket.

Retrieving data by key reverses the two steps:

1. Supply the key and receive the associated ticket.
2. Use the ticket to retrieve the associated data.

THE WORK FILE BEFORE MULTIPLE VERSIONS

Before describing how the current QUAD maintains multiple versions of a file, I will describe how earlier versions maintain the work file.

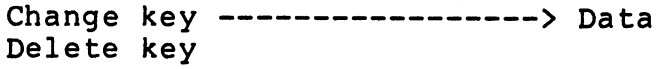
A ticket file is used as a work file, and contains two types of keys within the key structure: keys describing deleted ranges, and keys describing new or changed data.

To do a deletion, QUAD makes a single entry in its work file which is just a 17 character key. The first character is a "D" (for delete), the next 8 characters are the lower line number in the range, and the last 8 are the upper line number.

Since deletion is achieved with a single work file entry, it is very fast, and the speed is independent of the number of lines being deleted.

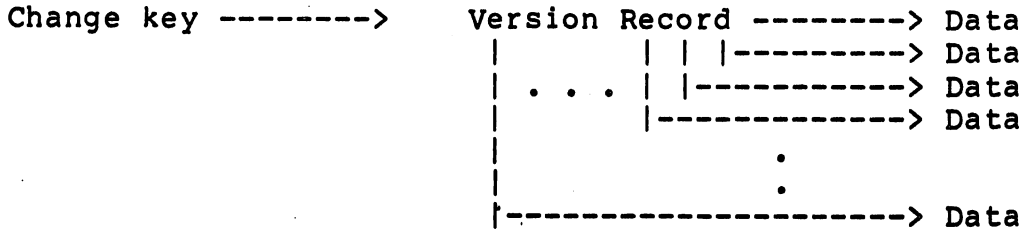
A change entry consists of both a key and data. The key is just the letter "C" followed by the 8 character line number, and the data is the line of text corresponding to that line number.

Schematically this structure is as follows, with an arrow representing a ticket held with the key and pointing to the data.



IMPLEMENTING MULTIPLE VERSIONS

Multiple versions were implemented by introducing a version record for each key, as follows:



Again the arrows represent tickets. In this case a ticket is stored with the key and points to a record called the version record, which itself contains one or more tickets pointing to data. With each such ticket there is a number identifying the version to which the data belongs.

Using this structure QUAD can, for each line number (represented by the change key in the figure), maintain multiple versions of each line.

Version records are themselves variable length, each being large enough to hold tickets for all versions of the corresponding line. There are generally many fewer versions of a particular line than there are file versions, because a given line will not change with each version of the file. A version record is restricted by QUAD internal buffering to 31 versions, but in general this will allow hundreds of file versions.

To operate on a version of the file, QUAD must integrate all deletions and changes for that version and all previous versions with the originally texted file. The algorithm to accomplish this is one of the most difficult I have had to write, and is complicated by trying to optimize performance. One performance problem that arises is that modifications to early versions which have since been deleted can slow down access to later versions.

IMPLEMENTING THE CANCEL COMMAND

The Cancel command cancels all changes made by the most recent command to change the file. Two consecutive Cancels have no net effect: the second cancels the effects of the first.

The Cancel command was almost trivial to implement once multiple versions had been implemented. The technique used was to reserve space in each version record to save a version number and ticket. Then when a command changes a line, the previous version number and ticket can be saved in this space. The Cancel command then just restores the saved version number and ticket to its prior place.

The only other implementation requirement for the Cancel command was to link together all the changed version records. This was easily accomplished using the tickets of the version records.

COMPILING FROM WITHIN QUAD

QUAD allows compiling for five languages: COBOL, FORTRAN, RPG, SPL and PASCAL. The syntax of the commands for compiling is identical to the syntax for the corresponding MPE command. However to compile from the file currently being edited, it is necessary to replace the text file part of the command with a line range enclosed in quotes. For example:

```
/SPL (ALL),$NEWPASS
/FORTRAN (20/40),USL,*LP
```

The ability to compile from the file being edited turns out to be especially useful for FORTRAN, because it permits compilation of single subroutines.

All compilations are done by invoking the requested compiler as a son process. File equations are set up for all specified files, and the compiler is passed a PARM to tell it which files were specified.

Whenever a line range is being compiled, QUAD passes the line range to the compiler through a message file. Message files are a new file type for MPE as of MPE IV.

This was my first experience with message files, and I encountered the following problems:

1. Unless the message file is built to contain only a single block, all blocks are posted to disc. QUAD uses a single block message file to prevent this posting.
2. If QUAD fills the message file before the compiler opens the file, QUAD's next write will fail with an end-of-file error. In this case, QUAD must loop, pausing and trying to write until the compiler gets the file open. Once this occurs QUAD will automatically be suspended by the file system on trying towrite to a full file, as long as the compiler has the file open.
3. QUAD must be careful not to send a null file (no records) to the compiler, because the file system will suspend the compiler indefinitely on its first attempt to read a record regardless of whether the file has any writers.

CONCLUSION

QUAD was created to quickly list files and make simple changes. I believe it or a similar tool belong in every 3000 shop as a significant resource saver.

A few users now use QUAD rather than EDIT/3000. This pleases me because I think QUAD deserves it, although there are still things that EDIT can do which QUAD cannot. However, I suspect that most users of these tools would quickly abandon them for general editing were a good full-screen editor to appear. I know I would.

If there is any permanent significance to QUAD, I believe it is to be found in the ticket file access method, which I have found to be enormously flexible and easy to use. QUAD does not take full advantage of its flexibility, and I am looking for an application that does.