Chapter 6:

THE EDITOR

INTRODUCTION

The Editor is the Valdocs word-processor and is also treated as the default application in the system. It is designed to be easy to use with little or no training. The following chapter describes its structure, operation, and files.

HISTORY

The 1.18 version of the Valdocs Editor developed from an existing STOIC "text editor", which only worked with files that would fit in memory, and an amalgam of older documents and original work. A "Restart file" concept and a block menu were also incorporated into the design, along with imbedded sequences and a state stack.

In a "what you see is what you get" wordprocessor such as Valdocs, a method of allowing for invisible commands had to be worked out. The use of ANSI extended ASCII sequences produced commands that are invisible to the user, but that control all characteristics of a document. Such sequences are referred to as the Imbedded Sequences.

Unlike most wordprocessors, Valdocs maintains the "form" of a document dynamically so the user does not have to call a form routine for a paragraph or sentence. Although the Control-Q menu contains a command to form the entire document, it is mainly intended for use with non-Valdocs files that have been brought over or produced from a formed document.

TWO FILE TYPES

The Editor is designed to work within a file-oriented environment. Editor files fall into one of two classifications: those with a .TMP extension, which are working files for use by the Editor alone; and those with the extension .VAL, which are valid Editor document files. The exact contents of the files with a .TMP extension are determined by which file it is, but none of the documents are valid. Those files with the extension .VAL can be manipulated by the Editor or any other application (e.g., PRINT or MAIL) that manipulates document files.

STORING A DOCUMENT

When the user presses STORE, all current work merges into a new file called TEXTACTV.VAL and is passed to the Indexer. The Editor never alters a source document, which preserves the integrity of the system. The Indexer then renames the file to a system-dependent name known to Indexer, and returns control to the Editor. Editor reappears with a clean slate to begin a new task.

The user may opt, via a menu option, to store the new file under a TPM name of his choice. This, too, will return the Editor to a clean slate status. If the user presses "undo" from the Indexer, TEXTACTV.VAL is returned to the Editor, which then renames the file to TEXTBKUP.VAL and treats it as a new source file. Thus the life of an active file, at least under that name, is quite short.

RESTART FILE

Should the user leave the system to go to any other application, without first storing the current work, the Editor will close all working files and store all working data currently in RAM into a file called TEXTRSRT.TMP. This file is recreated each time there is a chaining to another part of the system. It is used to restore the Editor to its most recent state, in the event that the user powers down or has problems.

When the Editor comes up, it tries to establish itself to the most recent state, preserving as much of the user's work as possible. The Editor first looks for a stray active file which, if found, is renamed to TEXTBKUP and loaded in as a source document. In lieu of that, the Editor searches for a restart file. If found, this file is used to reestablish the Editor's state. Otherwise, Editor searches for a backup file, which, if found, is brought in as a source document. Finally, in lieu of that, the Editor is brought up in its default "clean slate" state, and work may begin. Normally the system comes up in clean-slate or restart.

DELETED TEXT

The Editor handles deleted text in one of two ways. First, it checks to see if the text deleted is large or small. If large, such as a block or end-of-file, deleted data is stored in a restart file. If small, such as a character, word, or sentence, the deleted item is stored in a virtual stack. Items are popped off the stack by repeatedly striking the UNDO key. If, however, the system chains to another application, the Undo stack is cleared and all deleted data that was stored in the stack is lost.

OTHER FUNCTIONS

STOP: The Editor only recognizes the STOP key in one case: when it is formatting the entire document (from the CTRL-Q menu).

HELP: The Editor has its own help file on the disk. There is only one help message for Editor. The help file itself is not created with Valdocs because the imbedded sequences would be displayed with the rest of the text.

D-opcodes: The Editor uses D-opcodes only for resetting the screen. Characters are output through a regular console output call.

Fonts: The fonts (Bold, Italic, Size, Style) are simply calls to the bit-driver. Style currently performs underlining and Size controls variable line spacing.

VALUABLE DOCUMENT

Editor sees the valuable document as a stream of bytes. The Editor traverses this stream, displaying text to the user, continually modifying the text on a keystroke by keystroke basis, and keeping the document formatted at all times to preserve the "what you see is what you get" effect.

A valuable document consists of three types of information:

Hard Characters: entered by the user in the form of text via the keyboard. This is the text itself. Any non-control character (30h to 7Eh and A0h to FEh) may be entered. These characters are entered and deleted only by the user; the Editor does not create or destroy hard characters.

Soft Characters: soft spaces and soft carriage returns. These characters format the text, and are inserted and removed by the system as needed. Soft character insertion is performed as a matter of course in such functions as word-wrap and line-centering.

Imbedded Sequences and Control Characters: placed into the text by the system, at the user's direction. Normally these characters affect the state of the system in some way or other, although the user never sees the actual imbedded sequences. They are only inserted into the text--never deleted. Examples of imbedded sequences are: margin settings, bold on/off, and forced page break.

The effect of an imbedded is nullified by entering a counter-balancing imbedded. Thus text characters may appear to be close together, but are, in fact, separated by many sequences, which accounts for the system slowdown as it tries to read a large number of sequences. The user can tell whether or not an imbedded sequence occurs on a line of text, by noting an asterisk (*) on the far right side of the screen.

Imbedded Sequence Parameters and Identifiers

An imbedded sequence consists of a series of parameters followed by an ASCII identifier of one or two characters. The sequence is deliminated by a leading 9B Hex character and a following 9C Hex character. These "markers" allow for the detection of a sequence in a document.

a semi-colon (;). Valdocs 1.18 has either one or two parameters in its sequences.

The indentifier can be one or two characters. A lowercase letter identifies a single character; a space (20 Hex) and an uppercase letter identify a double character. The following is an example of a typical sequence, which gives the command "turn underlining on":

```
(9B Hex)1;s(9C hex)
```

In the imbedded sequences listed below, the parameters will be called P1 and P2 (optional). The identifier will be called the ID . (In the above example there is no P2, P1 is set to 1, and the ID is "s".)

ID = "v" Conditional and Unconditional Page Breaks

P2=0 Page break.
  P1=0: Unconditional, new page after next carriage return.
  P1=Non Zero (conditional), if within P1 lines of end of page, new page after next carriage return.

P2=1 Set line number.
  P1=0: Do nothing (NOP).
  P1=Line number (set the line number).

ID = "m" Font Selection

P2 not present
  P1=0: Set font normal.
  P1=1: Set font bold.
  P1=3: Set font italic.

P2=3
  P1=1: Set font bold italic.

ID = "s" Underlining

P2 not present.
P1=1: Underlining on.
P1=0: Underlining off.

ID = "t" Tabs and Margins

P1=0
  P2=Tab set in pixels, count starts with one. In Valdocs 1.1x a character was 8 pixels wide.

P1=1
  P2=Tab release in pixels.

P1=2
  P2=Left margin release in characters, number of characters left of margin.

P1=3
  P2=0:Right margin release off.
  P2=1:Right margin release on. P1=4.
  P2=Left margin setting in characters.

P1=5
  P2=Right margin setting in characters.

ID = " H' Centering

P1=2
P2=0:Centering off.
P2=1:Centering on for this line only.

ID = "w" Page Format Commands

P1=1 and on first line of document only!
P2=Page length in lines, size of text plus the top and bottom margins.

P1=2 and on first line of document.
P2=Top margin in lines.

P1=4 and on first line of document.
P2=Bottom margin in lines.

ID = " F" Wrap and Justify commands

P1=0
P2=0:Wrap of text off.
P2=1:Wrap of text on.

P1=1
P2=0:Right justification of text off.
P2=1:Right justification of text on.

By and large, application programs using Valdocs files can ignore these sequences.

Note that the Soft-Space and the Soft-Carriage-Return (SCR) are the codes for space (20 Hex) and carriage-return (0D Hex) with the high bit set (80 Hex). The Non-Break Space is inserted by the user to prevent wordwrapping because text has been entered past the right margin; a SCR is inserted after the space where the editor wraps the text. A Non-Break-Space (1F hex) is a means of telling the Editor not to wrap a line at this point.

File Conversion from Valdocs

A quick conversion to an ASCII file can be accomplished by copying the Valdocs file to a new file, with the following conversion criteria:

1. If a 9B hex is found, delete it and all characters up to and including a 9B hex.

2. If a Non-Break Space (1F Hex) is found, convert it to a 20 Hex (Space) character.

3. After the two above steps, if the high bit (80 Hex) is set, mask it out (set bit to zero).

The use of the TAB key in Valdocs does not insert a Tab character. TAB will insert a number of spaces, depending on the "size" of the tab set.