The Hybrid Music System
for the BBC Microcomputer

AMPLE Toolbox

USER GUIDE

AMPLE Toolbox Issue Disc

ID: 41410
Passcode: 1945

First published 1989

Hybrid Technology Limited
273 The Science Park
CAMBRIDGE
CB4 4WE

Issue 1

Written by Chris Jordan and Tony Thompson

Contents

1 Introduction

AMPLE Toolbox is a collection of extensions to the AMPLE Nucleus and Studio 5000 software, consisting of:

TEDIT text editor, for entering and editing AMPLE words:

* 40/80 column, vertical and horizontal scrolling edit window
* insert mode editing, including line split and concatenate
* editing of any number of words simultaneously
  (allowing words to be easily combined and split)
* private text storage, retaining text on program run and save
* module format, allowing load or discard at any time
* CLEAR, GET, ADD, NAME and MAKE commands

IEDIT image editor - for entering and editing mode 7 displays:

* entry of text, graphics, colours and effects
* full-screen editing area
* block copy and character paint facilities
* second image buffer, with get, put and swap
* storage of full or part screens as words
* compact or full text word formats
* module format, allowing load or discard at any time
* CLEAR, GET, NAME, MAKE and TMAKE commands

UTILS utilities - for advanced program development:

* LEDIT line editor with CLEAR GET APPEND LIST RENUMBER & MAKE
* COMPILE to make minimum-sized run-only versions of programs
* DISCOMPILE to return compiled programs to an editable form
* MERGE to combine a saved program with the one in memory
* BROWSE to view and search a tree-display of the program
* SPARESHOW and SPAREDELETE to show and delete unused words
* ABBREV to find the minimum abbreviation of any word
* REPORT to locate and describe the position of the last error
* module format, allowing load or discard at any time

SideMod sideways RAM module store (Master 128 only)

Sidemod adapts any System Disc to store the user's selection of modules in Master sideways RAM, from where they are loaded when required, giving faster access to editors and removing the need for a system disc once the system has started.

1 Introduction

AREC AMPLE disc program recover

AREC recovers lost AMPLE programs from DFS and ADFS floppy discs, allowing recovery of programs that have been accidentally deleted or whose disc has become corrupted.

This User Guide gives all the information required to install and use the Toolbox software. For some subjects, it refers you to the AMPLE Nucleus Programmer Guide for further information.

2 Installation

Check that you have received the following:

* this User Guide
* AMPLE Toolbox Issue Disc
* label for AMPLE Toolbox System Disc

You will also need:

* BBC Micro Model B, B+, or Master 128, with disc drive and TV or monitor
* Music 5000 Synthesiser, installed and tested
* one blank disc, formatted to suit your system

You may also have other Hybrid Music System components, such as:

* Music 4000 Keyboard, installed and tested
* Music 3000 Expander, installed and tested
* Music 2000 Interface, installed and tested
* Music 1000 Amplifier, installed and tested

making a system disc

The AMPLE Toolbox Issue Disc supplied creates (for your ROM only) the Toolbox System Disc which you then use.

The Issue Disc will operate on both 40 and 80-track disc drive. If your drive has a '40/80 track' switch, you should set it to suit the type of disc you have chosen to use with your System. In the case of a dual drive with switches, both must be so set.

To generate an AMPLE Toolbox System Disc:

* insert the Issue Disc into the disc drive (if there are two slots, into the top slot)
* tap the BREAK key while holding down SHIFT
* follow the instructions that appear on the screen.

When the computer asks you to enter a passcode, read it from the inside front cover of this User Guide. Before pressing RETURN, check that the passcode you have entered, and the disc title and ROM ID shown at the top of the screen, are the same as those printed inside the cover.

When the message 'Generation complete' appears, remove the System Disc and stick the supplied label on it, folding the label around the right edge of the disc so that the black strip ends up on the back.

If the message 'Generation complete' fails to appear, refer to 'fault-finding' below.

testing

Start the system from your existing Studio 5000 System Disc. When the Main Menu appears, remove the System Disc, insert the AMPLE Toolbox System Disc in drive 0 and press f9. When the Toolbox menu appears, press RETURN to select the first option, TEDIT. After a moment, the TEDIT screen will appear, and you will be able to press TAB to enter edit mode, and type-in some text.

fault-finding

If the Issue Disc fails in any way, it displays an error message followed by 'Generation aborted'.

If the message is 'Disc fault', probably your blank disc is faulty or incorrectly formatted. Try reformatting it or using another disc.

If 'Disc fault' appears while the Issue Disc is being accessed by the drive, the drive or disc may be faulty. If this or any other problem persists, contact your supplier.

If on attempting to load one of the modules, you see the '! Bad module' message, you entered the wrong passcode. Re-make the system disc, re-using the same disc if you wish.

3 TEDIT text editor

TEDIT is an editor designed specifically for text, and therefore it offers various advantages over Notepad, the Studio 5000's general-purpose editor. It is used to edit AMPLE word definitions in textual form, and to enter the text of new word definitions and command sequences.

loading

Before TEDIT can be used, it must be loaded from the Toolbox System Disc into memory. It then remains available for use until it is unloaded.

The simplest way to load TEDIT is through the Toolbox 'jukebox' menu. This discards the user program, so if necessary you should save it first and reload it afterwards. There is a more advanced method that lets you to load TEDIT without disturbing the user program - see the chapter 'editor module management' for details.

To load TEDIT using the Toolbox jukebox, make sure you have saved the current program if you want to keep it, and:

* remove the Studio 5000 System Disc
* insert the Toolbox System Disc
* ensure you are in command mode or the Main Menu, and press f9
* select one of the TEDIT options
* remove the Toolbox System Disc
* insert the Studio 5000 System Disc

Once loaded, TEDIT remains available until the Main Menu or Toolbox jukebox needs to load a different editor, at which point the TEDIT module is unloaded (removed from memory) to reclaim the memory it uses. The Main Menu's 'Run program' and 'New program' options will also unload the TEDIT module, and you can also unload it directly with a command - see the chapter 'editor module management' for details.

command and edit modes

As with most Studio 5000 editors, TEDIT provides a **command mode** in which you enter commands to transfer material between TEDIT's text store and the program you are working on, and an **edit mode** in which you work on the material in the text area. You press TAB to swap between the two.

The screen is divided between the command area and the edit area. When in command mode, you work in the command area, which may expand to push the edit area off the screen. When you enter edit mode, the edit area re-appears, shrinking the command area to show just the last few commands you made.

commands

TEDIT provides the following commands for transferring between its text store and the user program:

CLEAR	clear text area (does not refresh screen)
"wordname"GET	get the definition of the named word, replacing the existing text
"wordname"ADD	add the definition of the named word to the text area, before the current line
"wordname"NAME	complete the definition of a word by adding "wordname" [at the start of the text, and] at the end
MAKE	execute the contents of the text area

You use CLEAR, GET, AND MAKE in the same way as their counterparts in, for example, Notepad. To set the name of a new word, either before or after entering its contents, you use NAME in the same way, too.

entering text

After loading TEDIT for the first time, or entering CLEAR, the text store is empty. At all times, a solid block marks the end of text, and when, as now, the text store is empty, the marker appears at the top left corner of the edit window.

When you press TAB to enter edit mode, the cursor moves to the end-of-text marker, and you can enter text, as follows:

* the characters you type appear at (immediate left of) the cursor, moving the cursor to the right;

* pressing RETURN moves the cursor to the start of the next line down, that is, it inserts a 'return character';

* pressing DELETE removes the character to the left of the cursor and moves the the cursor to the left, or if you had just pressed RETURN to move down a line, it removes the 'return character' and so moves the cursor back up to the end of the previous line.

As you enter text, the end-of-text marker is moved along with the cursor.

entering a word definition

To make a new word using TEDIT, starting from command mode, you:

* enter CLEAR if necessary, and press TAB to enter edit mode
* type in the contents of the word, for example:

```
0:CDEFG
```

* press TAB and set the name, for example:

```
"tune"NAME
```

* enter MAKE

If MAKE finds a mistake in the text, the cursor will be placed on it when you next re-enter edit mode.

There are two other ways you can set the name:

* by using NAME in the same way but **before** entering the word
* by not using NAME, but entering the full definition of the word including its quoted name, [and], as it would be shown by TYPE, for example:

```
"tune" [ SCORE
0:CDEFG
]
```

Note that, as NAME simply adds lines to the text area, you cannot change your mind and simply use the NAME command again, as is possible with other editors. In this case, you must either first clear the text area, or delete the unwanted name while in edit mode (see editing text, below).

editing text

The cursor keys let you move the cursor off the end-of-text marker around the text, so you can go back to make insertions and deletions, as follows

* characters you type are **inserted**. that is, they push along the characters to the right of the cursor

* pressing RETURN inserts a 'return' character, splitting the line at the cursor, and moving down the cursor, the text to the right of it on the line, and all lines below

* pressing DELETE pulls the text to the right of the cursor along, or if the cursor was at the left end of a line, it removes the 'return' character, joining the line to the right end of the one above, and moving up the lines below.

Note that TEDIT has no special keys for insert and delete character and line, since they are not needed.

editing a word definition

If after entering a word, MAKE gives an error, you simply make the correction to the existing text and MAKE again. If you want to edit an existing word whose definition is not already in the text

store, starting from command mode, you:

* enter, for example:

  "tune" GET

* press TAB
* edit and MAKE as before.

If the word you GET contains the site of the last error in a word, then GET will automatically place the cursor at this point in the text.

additional editing keys

Though the character keys, RETURN, DELETE and the cursor keys are all you need to perform entry and editing functions, the following keys can speed up the process:

COPY	delete forward, removing the character (or 'return' character) at the cursor and pulling the text back
SHIFT COPY	delete the rest of the line, wiping all characters to the right of the cursor
CTRL COPY	delete the whole line, pulling up the lines below
SHIFT left/right	move left/right by one word, possibly to the line above or below
CTRL left/right	move to the start/end of the line
CTRL up/down	move to the start/end of the text

line copying

TEDIT provides the standard BBC line editing facility based on the COPY key, so that you can make copies of existing lines, with modifications.

To start copying, you press CTRL SHIFT COPY, and a block cursor appears to mark the insert position. You then move the underline cursor to the start of the characters you want to copy, and press COPY to copy each one. As you copy, you can use the DELETE key and the cursor keys as normal. To end copying, you press RETURN.

editing more than one word definition

Because TEDIT works with full word definitions, including name, [and], the text store can hold more than one definition at a time. For example, you can simply enter a second definition after the] of one already present, and MAKE will make one after the other, in one operation. Similarly, you can split a word into two definitions by an insertion in the single original definition, for example,

```
"tune" [ SCORE
0:CDEFG
] "tune2" [            % inserted line
1:Cbagf
]
```

The ADD command works just like GET except that it adds the definition to the existing text, inserting it before the cursor line. One powerful application of this is to gather a group of related words into the text area for editing together. Another use is to combine two words into one, for example:

```
"tune2"GET "tune"ADD   % (get in reverse order)
press TAB

"tune" [ SCORE
0:CDEFG
]                      % press CTRL COPY to remove this line
"tune2" [              % press CTRL COPY to remove this line
1:Cbagf
]
```

long text

There is no limit to the number of lines that the text store can hold. When there are more lines than will fit in the edit area, the edit area becomes a movable window on to the text store, and displays '+' indicators at the top and/or bottom to remind you that there are more lines, out-of-sight, in that direction.

As you move the cursor, the text scrolls automatically to keep the cursor in the edit area. Alternatively, you may yourself move the edit area up and down the text:

SHIFT up/down move up/down by half a screen-full

wide text

TEDIT allows lines to be longer than the screen width, whereupon the edit area acts as a window that automatically moves sideways to keep the cursor in view. If a line goes off the edge of the screen, a '+' indicator is displayed at that point to show that there are more characters in that direction.

TEDIT does have a limit to the line length, which, though it may vary from version to version, is guaranteed to be 80 characters or more.

changing screen mode

TEDIT provides a command 'TEDIT', one of the uses of which is to change the screen mode used for editing (the command is described fully in the chapter 'Editor module management').

The TEDIT command records the current screen mode as the editing mode for this session, so that it will always return to this mode for editing, even if you change mode by, say, running a program or entering a MODE command.

The 'jukebox' menu options set the appropriate mode before they issue the TEDIT command, but you can change editing mode by using MODE and re-entering TEDIT, for example:

```
3 MODE TEDIT
```

This does not clear the contents of the text store.

commands in text

The effect of TEDIT's MAKE is in fact to simply execute the contents of the text store as if it was typed in command mode - it 'makes' a word only because that is the effect of executing a full word definition.

This means that you can include just about any commands in the text area, and they will be executed by MAKE. One use of this is to execute automatically a test sequence for the word you are editing, by the addition of, say, the word name after the

definition itself at the end of the text, for example:

```
"note" [ @ 0:C ]
"randnotes" [
8 FOR( 7 RANDL note )FOR
]
SCORE 12, randnotes
```

You could also use TEDIT to execute just a sequence of commands, without any definitions. This would let you enter, check and, if necessary, edit them before beginning execution. Though any command that prints a message on the screen will over write the edit area, the display will be restored on returning to edit mode.

text storage

TEDIT holds the text being edited as part of the user program, in the form of data rather than words. An advantage of this is that SAVE saves the data along with the words of the program, and LOAD reloads it for you to continue editing, provided you are in TEDIT at the time.

To find out whether a program has TEDIT data, you use the SHOW command. This normally shows 'no data' after its list of words, but when TEDIT data is present, it shows 'TXT data'. The MEM command shows the amount of user memory the text consumes, as 'Data:'.

Since the text data uses up user memory, you may want to discard it before running the user program. To do this, you simply enter the command CLEAR. Similarly, to conserve disc space, you may want to CLEAR the text before saving the program.

This **public data** facility is discussed further in the chapter 'Editor module management'.

4 IEDIT image editor

IEDIT lets you design and edit mode 7 screen images using text, graphics and effects, and then store them as AMPLE user words for recall from within user programs.

loading

Before IEDIT can be used, it must be loaded from the Toolbox System Disc into memory. It then remains available for use until it is unloaded.

The simplest way to load IEDIT is through the Toolbox 'jukebox' menu. This discards the user program, so if necessary you should save it first and reload it afterwards. There is a more advanced method that lets you to load IEDIT without disturbing the user program - see the chapter 'editor module management' for details.

To load IEDIT using the Toolbox jukebox, make sure you have saved the current program if you want to keep it, and:

* remove the Studio 5000 System Disc
* insert the Toolbox System Disc
* ensure you are in command mode or the Main Menu, and press f9
* select one of the IEDIT options
* remove the Toolbox System Disc
* insert the Studio 5000 System Disc

Once loaded, IEDIT remains available until the Main Menu or Toolbox jukebox needs to load a different editor, at which point the IEDIT module is unloaded (removed from memory) to reclaim the memory it uses. The Main Menu's 'Run program' and 'New program' options will also unload the IEDIT module, and you can also unload it directly with a command - see the chapter 'editor module management' for details.

command and edit modes

As with other Studio 5000 editors, IEDIT provides a **command mode** in which you enter commands to transfer material between IEDIT's image store and the program you are working on, and an **edit mode** in which you work on an image. You press TAB to swap between the two.

When you are in edit mode, the edit area occupies the whole of the screen to show your full image. When you switch to command mode, the command area appears, covering up some of the edit area, and as you work in the command area, it may expand to temporarily push the edit area off the screen.

commands

IEDIT provides the following commands for transferring between its image store and the user program:

CLEAR	clear image area (does not refresh screen)
"wordname"GET	get the image of the named word, replacing the existing image
"wordname"ADD	add the image of the named word, overlaying the existing image
"wordname"NAME	set the name of the word to be made
MAKE	make a word with the given name, containing the current image in DISPLAY form.
TMAKE	make a word with the given name, containing the current image in standard text form.

You use CLEAR, GET, NAME and MAKE/TMAKE in the same way as their counterparts in, for example, Notepad. These commands are explained in more detail below.

entering text

After loading IEDIT for the first time, or entering CLEAR, the image store is empty. When you press TAB to enter edit mode, the blank edit area appears, with flashing cursor at the top left corner.

You may now enter text by typing as normal. Text entry in IEDIT is very similar to that in Notepad - you use the cursor keys to

move around the screen, and the same function keys to insert/delete line, and insert/delete character. The only difference is that COPY carries out block copying rather than character copying, as described later.

In addition, IEDIT has the following keys:

SHIFT f6	insert column
SHIFT f7	delete column
CTRL up/down/left/right	move to edge of screen

Note that you cannot enter any character in the bottom right corner of the screen, since attempting to display one at this position would cause the screen to scroll.

entering control codes

IEDIT's edit mode provides the teletext control codes on function keys, as follows:

function key	SHIFT function key
1 next alpha (text) colour	toggle flashing/steady
2 next graphics colour	toggle double/normal height
3 new background	toggle separated/contiguous
4 black background	toggle hold/release graphics
5	conceal display

Each of the single-code function keys - 'new background', 'black background' or 'conceal display' - just enters the appropriate code.

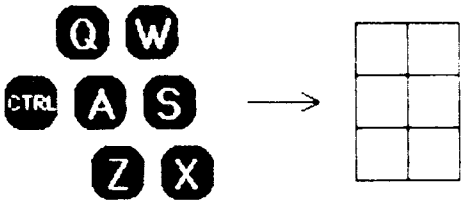
Each 'toggle' function key works similarly but enters whichever of its two codes is appropriate to that point on the line. It also leaves the cursor in place. For example, if the state of flashing/steady at the cursor point was 'steady', pressing SHIFT f1 would enter a 'flashing' code. If you then pressed right cursor, typed some text and pressed SHIFT f1 again, it would enter a 'steady' code.

Each 'next' function key either enters a colour code, or if the cursor is already on one, changes it to the next colour. It leaves the cursor in place, so you can tap it repeatedly until you get the colour you want. For example, to make an existing word appear in green, you would move to the space before it and tap f1 (next alpha colour) until green appeared. To return the following words to their original colour, you would move to the next space and again tap f1 until the required colour appeared.

As you move the cursor around the screen, it changes shape to tell you what kind of item is at that position - an underline indicates a text or graphics character, and a block indicates a control code.

entering graphics characters

To enter a graphics character, you alter individual pixels using the following cluster of six letter keys, with CTRL:



The character you start with will often be a space, but any non-graphics character will have the same effect. Alternatively, you can start with a solid block obtained by pressing f5.

Remember that graphics characters only appear as such if preceded by a graphics colour code on the same line.

making an image word

To make an image and store it as a word, you start from command mode and:

* enter CLEAR if necessary, then press TAB to enter edit mode
* compose your image using character, graphics and code keys Begin at the top of the screen so that if your image turns out not to fill the full height available, the unused space will all be at the bottom
* press TAB to enter command mode
* enter, for example:

```
"image"NAME MAKE
```

This stores your image without the unused space below it

You now have a word called 'image' which when executed will display your image. You may test it by entering its name at the %

prompt:

```
image
```

To recall a previously made image for editing, you use the GET command as normal:

```
"image"GET
```

using an image word

Once you've made an image word, you can invoke it from within your program just by including its name, like any other AMPLE word. A common example is as the title display for a piece of music:

```
"RUN" [
7 MODE titleimage
"1234"PLAY
]
```

When you type RUN, the MODE 7 instruction makes sure the screen is in mode 7 and also clears it. the image word 'titleimage' produces its display, and then the % prompt reappears immediately below the title.

A special case arises when the image is the full height of the screen, such that there is no line below the image to leave the cursor on. In this case, the image word leaves the cursor at the top left of the screen.

help facility

IEDIT's edit mode has a pop-up menu with options for **help**, reminding you of function key and cursor key usage, and **code**, describing the code or character at the cursor. To access either one, press f0, move to it (using cursor left/right keys), and press RETURN. When you want to return to editing, you press RETURN again.

The other options on the f0 menu are described further on in this chapter.

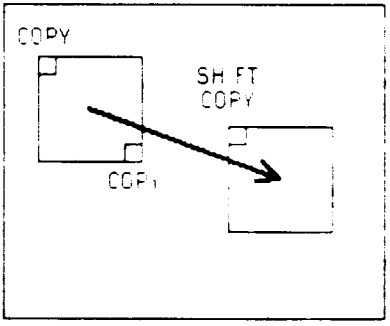
painting

You can take the character or code at the cursor position, and swiftly 'paint' it around the screen by holding down SHIFT while you move with the cursor keys. This is particularly useful for making designs with graphics characters, and for repeating a control code down a column. For example, to prepare an area of the screen for graphics codes, you might enter a graphics colour code at the start (left end) of the the area's top line, and then paint it downwards until you reach the area's bottom line.

copying

You can copy any rectangular area of the screen to another place on the screen using two markers. The procedure is as follows:

* move to one corner of the orginal area and press COPY
* move to the opposite corner and press COPY
* move to the top left corner of the destination area and press SHIFT COPY to make the copy.



You can make as many multiple copies as you like, where you like, but if the destination area overlaps the original, you may not get the result you expect.

Remember that many objects in your image will have control codes to the left of the visible part, so when copying any screen area that is less than the full width, be sure to include these essential codes in the rectangle you mark.

To help you in making precise copying operations, the f0 edit menu has an **info** option which displays the co-ordinates and positions of the cursor and both markers.

using image words in programs

It is quite simple to take a number of image words, and have them displayed in sequence by your program.

To make the images appear one under the other down the screen, you simply invoke one image after another, probably with some kind of pause in between, for example:

```
"message" [
7 MODE image1
#IN #2          % wait for key press
image2
#IN #2
image3
...
]
```

Once the screen is full, the display will scroll up as new lines are printed.

To make each image appear as a separate 'page', wiping out the previous one, you can include a '30 #OUT' instruction to move the cursor to the top of the screen before each image:

```
"message" [
7 MODE image1
#IN #2
30 #OUT image2   % home cursor, display image
...
]
```

Though you could use a 7 MODE instead of 30 #OUT, this would produce a momentary blank screen between pages. When using 30 #OUT you should make sure all images are the same length so that there is no way the bottom of one image can be left showing.

Instead of waiting for a key press, your sequence could use a fixed delay prduced by the following word definition:

```
"delay" [ % number delay
QTIME #- DURATION          % set period, ignoring current time
REP( QTIME SIGN )UNTIL(    % wait for period to end
IDLE )REP ]
```

used as follows:

```
...
500 delay                    % wait for 5 seconds
image
...
```

The number is the delay period in time units - at the normal tempo of 125, there are 100 per second.

synchronising images with music

Images may be displayed from musical parts by invoking the image word (or a non-IEDIT display word) from the score of a part itself. The following word definition is required:

```
"swait" [
REP( QTIME SIGN )UNTIL(   % wait for music event to end
IDLE )REP ]
```

You include this before the image word, to make the image appear on time, for example:

```
...
0:GgGAaAgfe
swait image
0:d////////
...
```

Another form of this uses a separate, silent player to 'perform' the images, so that the short time taken to actually output an image cannot disturb the rhythm of the musical parts. This is particularly convenient for images that appear at musical sections, since you don't need to mark any time in the image part. for example:

```
"part9a" [ swait imagea ]
"part9b" [ swait imageb ]
"part9c" [ swait imagec ]
...
"1239-abacada"PLAY % play music and image parts
```

One thing to remember when displaying images from any player is that they can appear right in the middle of you typing a command or using an editor! For this reason, its a good idea to equip your piece with a menu that offers a non-image performance, letting

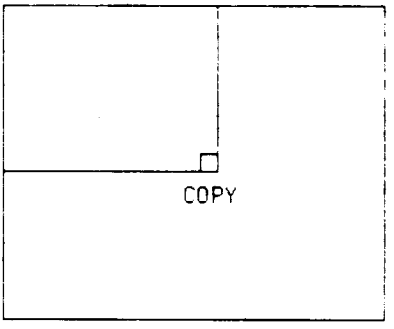
you use the Mixing Desk without disturbance, for example:

```
"RUN" [
7 MODE MENUDISP
%
%Dance of the Three Witches
%
%music and pictures%"1239-abacada"PLAY
%music only        %"123-abacada"PLAY
%
MENU ]
```

specifying the image area

MAKE and TMAKE normally store the complete image down to and including the last non-blank line in the image, but you can change this by specifying the bottom right corner, to store a taller, shorter or narrower area of the image.

To specify the bottom right hand corner of the area to be stored, you simply move to that position and mark it by pressing COPY. Now when you use MAKE or TMAKE, only that part of the rectangle between the top left corner and the marked position will be included in the word.



%MAKE % (or TMAKE)

This facility lets you include blank lines below the visible image, so that on printing, the corresponding area of the screen is cleared. It also lets you make narrower images for display in text windows created by your program, though the **windowed image** facility described later is sometimes better suited for this.

Note that though you use the same marker facility for both block copy and MAKE/TMAKE, MAKE/TMAKE take care to ignore markers that

are just left over from a previous block copy. In case you want to check before using MAKE or TMAKE, the 'info' option on the f0 edit menu displays 'm' after the marker co-ordinate if it applies to MAKE/TMAKE; the flashing on-screen markers are also only half height if they will not be used for MAKE/TMAKE.

Upon recalling an image word with GET, the marker is set automatically if it is needed to record the size of the image for MAKE. This means that you can GET an image, make minor changes, and then MAKE it, knowing that its size will be preserved. Note, however, that if your changes are outside the area marked, they will not appear in the final image. In this case, you will need to:

* clear the MAKE use of markers, by pressing COPY, COPY, then SHIFT COPY while in edit mode
* if desired, mark the new area to be stored with COPY

word formats

If you examine the definition of an image word with TYPE, you'll notice that MAKE creates a DISPLAY sequence just as you might enter in Notepad or TEDIT, except that it can also include the control codes found in the image.

The TMAKE command is equivalent to MAKE except it stores the image in standard text form, using $OUT and #OUT to display text and control codes, as if you had programmed it the hard way.

IEDIT gives you the choice of these two formats because each has its own advantages and disadvantages. MAKE's DISPLAY format uses up the minimum amount of memory and gives the fastest output, so this is the best choice in most cases. Only TMAKE's standard text form lets you edit, print and spool the definition like any other, so this is the one to use if you'll need to do any non-IEDIT work on the stored image.

GET works equally and identically on both formats, so you can convert an existing image word from one format to the other using GET and MAKE/TMAKE.

importing non-IEDIT images

Almost any mode 7 display produced by a user word can be carried into IEDIT. This is possible because GET gets the image from the named word not by examining its definition, but by executing it. Regardless of how the word creates the image and what it may do in addition, GET will transfer the image to the image store when the word finally exits.

One common usage is capturing the non-IEDIT-image title display of a piece, for which you would enter:

```
"RUN"GET
```

Having captured it, you could keep for later use by entering:

```
NEW
"image"NAME MAKE
"image"SAVE
```

The MERGE command in the Toolbox UTILS module provides a convenient way of adding it to another program.

Alternatively, you might want to edit the image and return it to the program in place of the original. Since you don't want to replace the whole RUN word by just this image, you will have to make it under a different name, and insert it by hand in the definition of RUN.

the image buffer

In addition to the image store that holds the current image you are editing, IEDIT has a completely separate image **buffer** which can hold an independent image, giving you greater flexibility for assembling complex designs.

You use the image buffer through the following operations:

* put (f0 menu)	copies the image store to the buffer
* get (f0 menu)	copies the buffer to the image store
* swap (f0 menu)	swaps the contents of the store and buffer
* copy (SHIFT copy)	if the marked area is in the buffer (having been put or swapped there), copies it to the cursor position in the image store

Some of the uses of the image buffer are:

* keeping a set of commonly used picture elements from which

individual ones can be copied when required
* combining parts of two pictures to make a third
* temporarily storing sections of the image while rearranging them on the screen
* keeping a safety copy of your image, ready for undoing a serious mistake

The image buffer is discarded by CLEAR, but is not cleared by the IEDIT command.

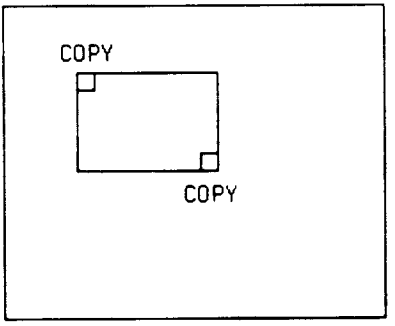
The **info** option on the f0 menu displays the positions of the markers and tells you if they are in the buffer, as opposed to the store.

windowed images

So far we have seen how to store just the raw data of an image, without any extra information about its size or position on the screen. This is ideal for images such full-size screens, titles (which you want to appear at the cursor position whatever it may be), and images that will be presented under the full control of more advanced programs.

In addition, IEDIT can also store **windowed** images, containing the image data as before, plus the definition of a text window which fully specifies the position and size of the image. Any rectangular area of the image being edited can be made into a windowed image word so that when displayed it will reappear at the same position, leaving the rest of the screen unchanged.

To store a part of the image being edited as a windowed image, you simply mark two opposite corners of the area using the COPY key (as you would for copying) before using MAKE or TMAKE:



%MAKE % (or TMAKE)

To display the image, you simply invoke the word in the normal way.

After displaying its image, each word leaves its window in place, with the cursor in the top left corner.

If you don't want to use the window further, you should cancel it with a 26 #OUT ('default windows' vdu code) so that the full screen is accessible again, for example:

```
7 MODE
winimage                    % display windowed image
26 #OUT                     % cancel window
```

Windowed images leave the window set since this makes it easy for you to modify the image, for example by over-printing additional text:

```
7 MODE
albumlogo
10 #OUT 10 #OUT             % move cursor down two lines
9 #OUT 9 #OUT 9 #OUT        % move cursor right three characters
"The Marching Forest"$OUT   % print additional text
26 #OUT                     % clear windows (if required)
```

The same result could be produced with a second windowed image containing just the title, in the correct position. To make a composite display from a number of windowed images, you just invoke them one-after-the-other, needing to cancel only the last window, for example:

```
7 MODE
albumlogo tracklogo authorlogo
26 #OUT
```

For more advanced applications, programs may also read the position and size of the window from the operating system.

Windowed images have many applications, including:

* as image elements for combination with each other to build a larger number of alternative screens conveniently, for example a set of titles with a common background or border
* as image elements for combination with displays from other sources, such as menus
* as a more-efficient alternative to a non-windowed words when the image is small (especially if it is not at the left screen edge) such as overlays for making successive frames of animated sequences

modifying windowed image words

As with words made with one marker, using GET to recall a windowed image will result in the markers being set such that a MAKE will recreate the original word. As before, modifications outside this area will not be included in the final MAKE, so be sure to check that the markers are where you want them before swapping back to command mode. This is particularly important in the simple case of wanting to reposition the image: after using GET, then inserting or deleting a few rows or columns, the original image area is hopelessly out-of-touch with the new position.

Windowed images can also be moved by editing them in TEDIT. Use TMAKE to obtain an editable version of the word, then GET it into TEDIT. You will see that the first line uses #OUT to send five codes that set up the text window. This is what sets the position of the image on-screen, so the image can be moved by updating the window limits. Be sure to keep the final window the same height and width as the original, or the integrity of the display will be lost.

adding images

The ADD command is similar to GET but instead of clearing the existing image, retains it and adds the new image over it.

To add a normal non-windowed image to the existing image, you position the cursor at the left end of the line you want the added image to start on, and then enter the ADD command, for example:

```
"banner"ADD
```

After adding the image, ADD positions the cursor immediatly below the bottom of it, ready for adding a further image below. ADD doesn't attempt to prevent you adding an image that is too big to fit, so this is your responsibility.

To add a windowed image to the existing image, you use ADD in the same way. ADD leaves the cursor in the top left corner of the added image.

Note that ADD does not affect any markers.

image storage

IEDIT holds its image store and image buffer as part of the user program, in the form of data rather than words. An advantage of this is that SAVE saves the data along with the words of the program, and LOAD reloads it for you to continue editing, provided you are in IEDIT at the time.

To find out whether a program has IEDIT data, you use the SHOW command. This normally shows 'no data' after its list of words, but when IEDIT data is present, it shows 'TEL data'. The MEM command shows the amount of user memory the text consumes, as 'Data:'.

Since the image data being edited uses up user memory, you will probably want to discard it when you have finished editing. To do this, you simply enter the command CLEAR. Similarly, to conserve disc space, you may want to CLEAR the image data before saving the program.

This **public data** facility is discussed further in the chapter 'Editor module management'.

5 UTILS utilities

UTILS provides eight additional program management commands and LEDIT, a BASIC-style line editor that can be used manually or under the control of batch files or user programs.

loading

Before UTILS can be used, it must be loaded from the Toolbox System Disc into memory. It then remains available for use until it is unloaded.

The simplest way to load UTILS is through the Toolbox 'jukebox' menu. This discards the user program, so if necessary you should save it first and reload it afterwards. There is a more-advanced method that lets you to load UTILS without disturbing the user program - see the chapter 'editor module management' for details.

To load UTILS using the Toolbox jukebox, make sure you have saved the current program if you want to keep it, and:

* remove the Studio 5000 System Disc
* insert the Toolbox System Disc
* ensure you are in command mode or the Main Menu, and press f9
* select the UTILS option
* remove the Toolbox System Disc
* insert the Studio 5000 System Disc

Once loaded, UTILS remains available until the Main Menu or Toolbox jukebox needs to load a different editor, at which point the UTILS module is unloaded (removed from memory) to reclaim the memory it uses. The Main Menu's 'Run program' and 'New program' options will also unload the UTILS module, and you can also unload it directly with a command - see the chapter 'editor module management' for details.

program management commands

Once the module has been loaded, its program management commands are available. They are:

ABBREV	display minimum abbreviation of word name
BROWSE	give program structure display
COMPILE	compile program
DISCOMPILE	discompile program
MERGE	merge program
REPORT	report error location
SPARESHOW	display names of unused words
SPAREDELETE	optionally delete unused words

Some of the commands can take some time to prepare themselves; for example, BROWSE has to decide which words to display first, before it can start. In this case, they print full stops as an indication that progress is still being made.

ABBREV display minimum abbreviation of word name
namestring ABBREV

ABBREV returns the minimum abbreviation for the given word name, whether Nucleus, module or user, in the current installation.

Note that the minimum abbreviation of a Nucleus or module word may change if a module is added or removed, or if, for example, user words with upper-case names are added or removed.

examples

```
%"ABBREV" ABBREV
AB.
%"MDELETE" AB.
MD.
```

BROWSE give program structure display

BROWSE presents a display of the structure of the program, allowing the user to move to inspect individual words in more detail using the cursor keys. Pressing TAB exits BROWSE and returns to the % prompt.

BROWSE initially displays a list of unused user words - the top level of the program structure - each followed by a full stop for each reference to a user word it contains - the next level down. A '>' character marks the current position, which may be moved using the cursor keys:

key	action
up/down	move up/down the list
right	enter the current word, to display an indented list of all the user word references within it
left	exit the current word, removing its list of references from the screen, and return to the list containing the reference to it
RETURN	display the definition of the current word. In 80-column screen modes, the definition appears to the right of the program display, but in 40-column modes, it replaces the program display and waits for a RETURN key-press before restoring the program display.

COMPILE compile program

COMPILE reduces the program to the smallest possible size by converting all user word names to a single character ('z'), and removing spaces, carriage returns (line ends) and comments. In this form, the program may be run but not fully read or edited.

As COMPILE processes each program word, it displays the number of bytes removed in each category, and in total. When compilation is complete, it displays the total number of bytes removed from the program.

COMPILE converts all user word names except those starting with an upper-case letter (such as RUN) and those that you have specifically protected by including them in the definition of a user word called NOCOMPNAMES. This allows you to protect the names of words that are called as commands, rather than just as instructions in other words. Examples include key words that the AMPLE Nucleus or modules look for, such as 'part1', and special command words used in menu options and macros. If these names were removed, the program would fail to function correctly. Since these words are usually unused as instructions in words, the SPARESHOW command is useful for detecting them.

After compilation, the NOCOMPNAMES word may be deleted to save memory.

Note that if COMPILE finds a DISPLAY or MENUDISP instruction, it retains all following spaces, line ends and comments in that word to ensure that the text of user displays is not removed.

example

```
   %SPARESHOW
part1     part2     part3     RUN      % list of key names
 %"NOCOMPNAMES" [ part1 part2 part3 ]  % protect key names
 %COMPILE                              % begin compilation
```

DISCOMPILE discompile program

DISCOMPILE expands any program to a fully readable and editable form, entering spaces and line ends where required, and giving arbitrary names to user words whose names have been removed by COMPILE.

Though DISCOMPILE is designed to restore a COMPILEd program to a readable and editable state, it may also be used to re-format any program to remove over-length lines, such as those created by direct entry of definitions at the % prompt, use of an 80-column editor or RENAMEing of word names. The maximum line length permitted is that of the current text window (usually the screen width).

As DISCOMPILE processes each program word, it displays the number of bytes added in each category, and in total. When compilation is complete, it displays the total number of bytes added from the program. DISCOMPILE never removes any type of item.

Like COMPILE, DISCOMPILE does not modify the contents of a word following DISPLAY or MENUDISP, thereby ensuring that the text of user displays is not disturbed.

example

```
%7 MODE                    % set 40-column line length
%DISCOMPILE                % begin discompilation
```

MERGE merge program
namestring MERGE

MERGE loads the named program **without** removing the one in memory, adding the two sets of words together to make a new, merged program.

If a loaded word has the same name as an existing one, a warning message is given so that you can RENAME the loaded word to something different. To avoid confusion when words are re-defined, you should normally do this immediately.

Any public data in the merged program is discarded. Any public data already in memory is retained, so if you are using an editor that uses the public data facility, for example TEDIT or IEDIT, your current text or image will not be affected.

REPORT report error location

REPORT shows where in the program the last error occured by printing the numbered line of the user word with a '!' indicating the reference to the system word that generated the error.

The line number that appears is the number of the line as assigned by LEDIT's GET (see later in this chapter). Remember that though the word may already be in the buffer, its line numbers could have been changed since the last GET, so to avoid confusion, you should use GET to get the current definition of the word before making corrections.

REPORT's record of the last error in a user word is not affected by errors in the input line or in non-user words used as commands.

REPORT does nothing if, since the error, you have used a command, such as DELETE or LOAD, that could have removed the user word.

example

```
 %RUN
No number in part1  % error message
 %RPEORT            % typing error
Mistake             % error message
 %REPORT            % command
30.SCORE :          % line printed by REPORT
         !          % indicates ':' caused error
 %                  % prompt
```

SPARESHOW display names of unused words

SPARESHOW displays the name of each user word not used as an instruction in any user word (including itself). It is very useful for identifying words that were left in a program after development, and may be deleted to save memory.

Note that a word found unused may be an important part of the program which is accessed only as a command, rather than an instruction inside a definition. Examples include RUN, key 'interface' words such as 'part1', and command words for use in menu options or macros.

Any word reported by SPARESHOW may be removed immediately using DELETE without any possibility of the '! In use' error, or using SPAREDELETE.

Note also that SPARESHOW will not report a word that uses itself even if it is not used by any other word in the program, since strictly speaking it is 'in use' and could not be directly removed using DELETE.

example

```
 %SPARESHOW
part1     part2     mix       purensim  RUN
 %"purensim"DELETE
```

SPAREDELETE optionally delete unused words

SPAREDELETE searches the program and for each user word not used as an instruction in any user word (including itself), displays its name and, if confirmed by the user pressing 'Y', deletes the word. Entering 'N' or any other character causes the word to be retained.

Note that as words are deleted, other words further down the list may become unused, and will therefore be offered by SPAREDELETE: do not be concerned if SPAREDELETE's list is longer than SPARESHOW's.

Similarly, words earlier in the list may become unused: in this case, repeated use of SPAREDELETE will be needed to remove all the unwanted words.

the LEDIT line editor

The UTILS module also contains a line-based editor, LEDIT. This provides a direct, command-driven method of editing any type of textual word definition, regardless of the length or number of lines. It supports conventional manual editing at the % prompt, 'batch' editing using files, and editing controlled by the user program itself.

The UTILS option on the Toolbox jukebox automatically enters LEDIT, just as the TEDIT option enters TEDIT.

LEDIT provides the following commands:

CLEAR	clear text store
GET	get text of word into text store
APPEND	add text of word to text store
.	put text on numbered line in text store
LIST	display text in store
RENUMBER	renumber text lines
NAME	complete the definition by adding name, [and]
MAKE	execute text

You use CLEAR, GET and MAKE just like their counterparts in other editors. To set the name of a new word, either before or after entering its contents, you use NAME in the same way too.

Unlike screen editors, LEDIT has no edit mode - the TAB key is not used. All editing is carried out through the above commands, along with the BBC's COPY editing facility for editing lines.

entering lines

After loading UTILS from the Toolbox for the first time, the text store is empty. Next time you might have some text left over from the previous use, so you should enter the command CLEAR to make sure the text store is empty before entering new lines.

To add a line to the text store, you enter a line number followed by a dot ('.') and the text you want to put on that line. Any number will do, but 10 is a convenient choice for the first line.

To enter a sequence of lines, you just using successive line numbers, for example:

```
%10.SCORE
%20.12 -1: C//G e//c
%30.       A/A/ /AAA
```

(The % signs are the prompt - you don't type them in yourself.)

By choosing numbers 10 apart, you leave room for insertions, as described in a moment.

To display the lines in the text store, you use the command LIST. It lists all lines, in numerical order.

If at this point you enter MAKE, the lines will be executed as if you entered them at the % prompt, allowing you to try them out before making into a word.

creating a word definition

To create a word in LEDIT, you enter its contents line by line, and then set the word name using the NAME command, for example:

```
"tune"NAME
```

NAME adds the quoted name and '[' on a new line at the start, and ']' on a new line at the end, renumbering the lines if it has to.

Now when you enter MAKE, the word becomes defined.

An alternative way of setting the name is to enter it yourself, along with the '[' and ']'. To create the 'tune' word this way, you would enter:

```
10."tune" [
20.SCORE
30.12 -1: C//G e//c
40.       A/A/ /AAA
50.]
```

and then MAKE as before.

editing lines

If MAKE reports a mistake, you will need to change one or more lines in the text. To make a minor modification to a line, you use LIST to display the original (if it is not already on the screen) and copy it using the cursor and COPY keys, making modifications using DELETE and character keys as you go. When you press RETURN, the new line will replace the old.

If you need to insert a new line, you simply enter it with a line number between the two originals, for example:

```
%LIST
 10."tune" [
 20.SCORE 12, -1:
 30.C//G e//c
 40.a/a/ /aaa
 50.]
%
%25.bCbC bCbC      % insert line between lines 20 and 30
```

If the line numbers of the originals are adjacent, you will need to use the RENUMBER command first. This imposes new line numbers, starting at 10, and increasing by 10 for each line, giving you 9

places to add between any two existing lines. Remember that after RENUMBERing the listing, any lines left on the screen will have invalid numbers, so you should use LIST again before making any changes.

To completely remove a line, you enter just the line number and dot, with no characters after the dot, for example:

```
%25.                % delete line 25
```

editing existing words

To get the definition of existing word for editing, you use the GET command as in other editors, for example:

```
"tune"GET
```

GET provides convenient line numbers just like RENUMBER.

editing more than one word

Since MAKE simply executes each of the lines in order, you can define as many words as you like with one MAKE. You can even use MAKE to execute a command sequence, to try it out before inclusion in a word.

To make more than one word, you just make sure that the text has the complete word definitions, one after the other. As an example, consider the "hello" word:

```
 %"hello"GET LIST
10."hello" [
20.   "Hello" $OUT
30.  NL
40.  "Goodbye" $OUT
50.]
 %35.]"goodbye" [
```

MAKE will now give two words, 'hello' and 'goodbye'.

The APPEND command lets you get more than one word into the text store. It is equivalent to GET but adds the definition of the new word to the end of the exiting text. APPEND is particularly useful for editing many words together, combining words, and

moving lines between words. For example:

```
 %"hello"GET "goodbye" APPEND LIST
10."hello" [
20.   "Hello" $OUT
30.  NL
40.]
50."goodbye" [
60.  "Goodbye" $OUT
70.]
 %40.
 %50.
 %          % the two words have now been combined into one
```

creating text files

Sometimes when using AMPLE you need to create a text file on disc, usually for executing with EXEC. This might be a commonly-used command sequence, or the definition of a word you want to later add to a program.

You could create a text file using the *BUILD command, but by using LEDIT (or any other AMPLE text editor), you gain the advantage of being able to edit the text before committing it to disc.

To create a text file with LEDIT, you enter the text as a DISPLAY sequence, and then include it in a definition with *SPOOL commands, for example:

```
CLEAR
10."makedelins" [
20."SPOOL delins"OSCLI
30.DISPLAY
40.%"purensim"DELETE
50.%"medieval"DELETE
60.%"zizzysyn"DELETE
70."SPOOL"OSCLI
80.]
90.makedelins
MAKE
```

This creates a text file containing:

```
"purensim"DELETE
"medieval"DELETE
"zizzysyn"DELETE
```

editing by batch file

Since unlike all other Main Menu and Toolbox editors, all LEDIT editing is carried out through commands, LEDIT is well suited to being driven by a 'batch' file - a text file containing a preprepared sequence of edit instructions to be carried out as a batch. Any sequence of LEDIT commands can be entered in to a text file and then executed when required using the EXEC command. This makes a very convenient way of carrying out simple repetitive edit sequences with minimum effort from the user.

Text files can be created using a standalone text editor, or one or any AMPLE text editors - Notepad, TEDIT or LEDIT itself. As an example, we will use LEDIT to prepare a batch file that adds a copyright message to any program's RUN word of the following format:

```
10."RUN" [
20.7 MODE DISPLAY
30.%  Song for Ro          % (first line of existing title)
40.%  ...                  % (reaminder of existing title)
90.]
```

To create the batch file on disc, you enter the following:

```
CLEAR
10."makeaddc" [
20."SPOOL addc"OSCLI
30.DISPLAY
40.%LEDIT "RUN"GET
50.%25.% 1989 (C) Robonk
60.%MAKE
70."SPOOL"OSCLI
80.]
90.makeaddc
MAKE
```

The final MAKE command creates and executes a word 'makeaddc', creating a file 'addc' on disc, containing

```
LEDIT "RUN"GET
25.% 1989 (C) Robonk
MAKE
```

Now, to use 'addc' to add the copyright message to a program, you would make sure that UTILS was loaded, and enter, for example:

```
"rosong"LOAD
*exec addc
"rosong"SAVE
```

You can reuse 'addc' on any program which has the same format RUN word.

program-controlled editing

Program-controlled editing goes one stage further than batch files, by creating and executing LEDIT edit commands from the user program itself. This is a very powerful technique that lets the user program define new words, define and execute words to define new words, or even completely redefine itself.

The following examples of program-controlled editing employs the standard method of issuing AMPLE commands from within programs using the '$+' word. To fully understand the examples, you must be familiar with this method - it is described in full in the AMPLE Nucleus Programmer Guide under the index entry 'using the imput line'. Because a sequence of editing commands are issued, each command line ends with a user word name to return control to the program.

The first example is a word to place given text on a given line in the definition of a word of a given name, by producing a command sequence of the format:

```
"namestring"GET
linenumber.textstring
MAKE nextnamestring
```

The definition is:

```
"setline" [
% namestring linenumber textstring nextnamestring setline
% note: builds a command line right to left
"MAKE " $+                    % -> "MAKE <nextnamestring>
13 $CHR $+                    % end of next command
$12 "." $+ $STR $+            % make linenumber.textstring
$+                            % add to left end of line
13 $CHR $+                    % end of next command
$12 """GET" $12 $+ """" $+    % make "namestring"GET
$+                            % end of next command
$+ ]                          % add to command line
```

To test it, first make a target word:

```
CLEAR
10."say" [
20.DISPLAY
30.% hello
40.% there
50.]
MAKE
```

and then a word to make a particular edit:

```
"editsay" [ "say" 30 "%hi" "" setline ]
```

Now, when you enter 'editsay' as a command, it will carry out the sequence:

```
"say"GET
30.%hi
MAKE
```

When using this in a program, you would replace the "" in 'editsay' with the name of the user word to which control was to be returned - remember that the $+ method requires that the command-line-generating word exit to the % prompt to execute the line.

Command-generators like 'setline' can be difficult to debug in their active form, so a useful technique is to replace the final '$+' with a '$OUT', to print the command rather than execute it. You then restore the '$+' when you've made sure the definition is producing the correct command.

The second example is a word to create a multi-line word definition from strings provided by the calling program.

```
"create" [
% namestring "" string1 ... stringn resumenamestring create
% note: builds one large command line, right to left
"MAKE " $+ 13 $CHR $+
"]"
100 REP(                                % start lines at 100, going dow
 LEN 0 #= )UNTIL(                       % until null string,
 "." $+ #11 $STR $+ 1 #-                % make line number and '.'
 13 $CHR $+ $+
 $12                                    % get next line to top
)REP $2
$12
""" [" $12 $+ "."" $+ $STR $+           % make '"namestring" ['
$+                                      % and add
13 $CHR $+ "CLEAR" $+
$+ ]                                    % add to comand line
```

Here's a simple example of its use, to create a variable:

```
"createtotal" [ "total" "" "GVAR" "next" create ]
createtotal
```

This issues the following command sequence:

```
CLEAR
98."total" [
99.GVAR
100.]
MAKE next
```

and 'total' is defined. 'next' is the name of the user word that returns control to the program. You can define an empty 'next' to test this example.

This definition of 'create' makes a single large command line (with carriage return characters separating the component commands) so the definition being created is limited by the string stack size. You could make a more-advanced version which overcame this by issuing more than one command line.

text storage

The number of lines that LEDIT can hold is restricted only by the available memory.

LEDIT holds the text being edited as part of the user program, in the form of data rather than words. An advantage of this is that SAVE saves the data along with the words of the program, and LOAD reloads it for you to continue editing, provided you are in LEDIT at the time.

To find out whether a program has LEDIT data, you use the SHOW command. This normally shows 'no data' after its list of words, but when LEDIT data is present, it shows 'LNT data'. The MEM command shows the amount of user memory the text consumes, as 'Data:'.

Since the text data uses up user memory, you may want to discard it before running the user program. To do this, you simply enter the command CLEAR. Similarly, to conserve disc space, you may want to CLEAR the text before saving the program.

This **public data** facility is discussed further in the chapter 'Editor module management'.

6 Editor module management

You can gain further benefits from the Toolbox editors by taking more-direct control over their management. This entails using certain additional AMPLE Nucleus commands - this chapter tells you how to use them, but full descriptions are only available in the AMPLE Nucleus Programmer Guide.

module loading by menu

The Main Menu and Toolbox jukebox manage the loading and unloading of modules entirely automatically, so you don't normally need to worry about editor modules. However, if you are going to make use of direct module management facilities, it's useful to know how they work, so a brief description is given here.

When you select an editor from the Main Menu, the Menu looks to see if it is already in memory. If it is, the Menu just enters the editor, but if it is not, the Menu first unloads the previous module it loaded, freeing-up its memory, and loads the required module from disc.

The Toolbox jukebox works in the same way, and co-operates with the Main Menu so that between the two of them, they only have one editor module loaded at a time, leaving the maximum amount of memory free for the user program. The Main Menu's 'Run program' and 'New program' options will unload a module loaded by the Toolbox jukebox, in the same way as a module loaded by the Main Menu itself.

unloading the menu-loaded module

Since each loaded module reduces the amount of memory for the user program, you may sometimes want to unload the current editor module without loading another. This gives you free memory to run the program, make changes to small words at the % prompt, and load modules directly as described below.

To unload the current editor module, you simply select 'Run program' from the Main Menu. 'New program' has the same side-effect.

loading modules directly

The MLOAD command lets you load modules directly, as opposed to via a menu. For Toolbox modules, the main advantage is that the user program is not erased by the jukebox, so you can load a module whenever you need it without having to save the program first.

MLOAD attempts only to load the named module - it does not try to unload any previous module or enter any editor in the loaded module. Unless you particularly want more than one module present, you can ensure that the last menu-loaded module is not present by selecting 'Run program' from the Main Menu.

To load the module, insert the Toolbox System Disc into the drive and enter the quoted name of the module followed by MLOAD. If, as is likely in the case of TEDIT and IEDIT, you also want to enter the editor, you add the name of the editor as a command after the MLOAD, for example:

```
"TEDIT"MLOAD TEDIT  % load TEDIT module and enter TEDIT
"IEDIT"MLOAD IEDIT  % load IEDIT module and enter IEDIT
"UTILS"MLOAD        % load UTILS module
"UTILS"MLOAD LEDIT  % load UTILS module and enter LEDIT
```

Once the module has loaded successfully, you can replace the Studio 5000 System Disc if necessary.

If there is insufficient memory to load the module, MLOAD will give the '! Too big' error. If you get this unexpectedly, it may be that the last menu-loaded module is still present. See 'unloading the menu-loaded module' above.

TEDIT is special in that it memorises the current screen mode in entry, so you may want to set the screen mode also, for example:

```
"TEDIT"MLOAD 3 MODE TEDIT
```

Once you have entered an editor, you use it just as if you had loaded it from the jukebox.

An MLOADed module remains present until you remove it explicitly.

finding out what modules are loaded

To get a list of the names of loaded modules, you may enter the MCAT command. The extra information you see is not needed at this stage, so is explained only in the AMPLE Nucleus Programmer Guide.

unloading a module directly

Having loaded a module using the MLOAD command, you may unload it to reclaim the memory it uses with the MDELETE command.

You put the name of the module in quotes before the MDELETE command. AMPLE won't let you unload an editor while you are still in it, so you need to add the QUIT command to exit it before MDELETE, for example:

```
QUIT "TEDIT"MDELETE          % unload TEDIT
QUIT "IEDIT"MDELETE          % unload IEDIT
QUIT "UTILS"MDELETE          % unload UTILS
```

Unlike filenames, the **precise** name of a loaded module must be entered, that is, in upper-case.

Many editors clear the screen when they are quitted, so don't be alarmed if the screen goes blank for a moment.

loading and unloading Main Menu modules

The Main Menu editor modules may be loaded and unloaded directly in the same way as just described, though unless you want more than one editor loaded simultaneously, you will probably prefer to use the Main Menu to do this. The names of the Main Menu editor modules are:

Notepad	PAD
Staff Editor	STAFF
Mixing Desk	MIX
Recorder	REC

For example, to load and enter Notepad, you enter:

```
"PAD"MLOAD PAD
```

unloading the menu-loaded module directly

You are free to use MDELETE to unload the module last loaded by the Main Menu or Toolbox jukebox - when either menu comes to unload the module itself, if the module can't be found, it continues regardless.

switching between editors

One powerful feature of AMPLE is its ability to have many editor modules present at the same time (memory permitting), to switch instantly between them using their entry commands or menu options.

Once you have loaded an editor module using the Main Menu, Toolbox jukebox or MLOAD command, you may load a second using the MLOAD command (the Main Menu and Toolbox jukebox would automatically unload the first one). Then, to switch to either editor, you simply enter its entry command, for example:

```
"UTILS"MLOAD        % UTILS module, containing LEDIT
"MIX"MLOAD
LEDIT               % enter LEDIT
...                 % do some editing
MIX                 % switch to Mixing Desk
```

In the case of a Main Menu editor, you may find it more convenient to simply select it from the Main Menu, rather than entering its name. You may even use the Toolbox jukebox to enter already-loaded editors.

Though you can switch freely between loaded editors, the data for each is not necessarily retained while you use another - see below.

public data storage

Unlike the Main Menu editors, the Toolbox editors use the AMPLE Nucleus public data facility, so that the material being edited is stored as part of the user program and is saved, loaded and discarded along with it.

The SHOW command displays the type of data present in the program using a three-letter abbreviation, or 'no data' if there is none. The types of data of the Toolbox editors are:

```
TEDIT:              TXT     (text)
IEDIT:              TEL     (teletext screen)
LEDIT:              LNT     (line-numbered text)
```

Only one type of data can be present at a time.

The MEM command shows the number of bytes of memory used for public data as 'Data: '.

clearing editor data

The Nucleus CLEAR command discards all material being edited (returning to the initial 'no data' state), freeing the memory it uses. If you get short of memory for running or editing your program and you do not want to retain the editor data, you use CLEAR to reclaim this memory. If you do not want the editor data stored on disc as part of the saved program, you use CLEAR before SAVE.

When you are in an editor which does not use public data, such as any Main Menu editor, the CLEAR command is in fact re-interpreted accordingly by the editor and so does not affect the public data. If you were in a Main Menu editor and wanted to clear the public data, you would use the Nucleus command QUIT.

Because only one type of data can be held at a time, the existing data is cleared on entering or accessing data in an editor that does not recognise the data type. This means that after using a second Toolbox editor, on returning to the first you will find that its data has been cleared. However, after switching to a Main Menu editor, on returning to the Toolbox editor you will find its data is still there.

In summary, the only operations that clear public data are:

* the commands AMPLE and LOAD (which also clear the words)
* CLEAR, when in a public data editor or no editor
* QUIT
* entering or accessing data in an editor that does not recognise the data type

In particular, the data is NOT cleared on:

* running the user program
* loading and entering an editor that recognises the data type
* using any Main Menu editor (since they don't use public data)
* use of NEW

using loaded editor data

When you load a program, any editor data saved with it reappears ready for immediate use. If you are in the appropriate editor at the time, you can access it immediately. In the case of a screen editor such as TEDIT or IEDIT, the data will not be visible on screen until, for example, you press TAB to enter edit mode.

If you reload a program while in an editor that does not recognise the data type and then try to edit it, the data will be cleared the first time the editor accesses it - when you press TAB for example. If you want to retain the data you have loaded, you should use SHOW to check the type of the data, and then enter the appropriate editor before trying to access the data.

7 SideMod sideways RAM module store

SideMod adapts any System Disc to store the user's selection of modules in Master sideways RAM, from where they are loaded automatically when required. This gives faster access to editors and removes the need to keep the Studio 5000 System Disc in the drive once the system has been started. If the user includes also the Toolbox modules, then these become available at all times, independently of the Toolbox System Disc.

SideMod also makes more efficient use of space on the Studio 5000 System Disc by leaving more catalogue entries free for user files.

SideMod works only on the BBC Microcomputer Master 128. Sideways RAM must not have been disabled (the computer is shipped with it enabled, set by internal links). SideMod is not compatible with the Econet, so there must be no Econet fitted, or if Econet is fitted, it must not be in use.

licence for use

Like AMPLE Toolbox package itself, SideMod is supplied for use with the AMPLE Nucleus ROM identified on the inside front cover of this User Guide, and no other. SideMod may only be used to adapt System Discs for use with this particular ROM.

preparing a Studio 5000 System Disc

SideMod makes permanent changes to the System Disc, so to be safe you should make a copy of the System Disc and use the copy.

If you are using standard DFS discs on a dual drive,

* insert your Studio 5000 System Disc in drive 0
* insert a blank formatted disc in drive 1
* enter

```
*BACKUP 01
```

If you are using standard DFS discs on a single drive,

* have ready your Studio 5000 System Disc
  and a blank formatted disc

* enter

    *BACKUP 00

  and follow the instructions that appear on the screen.

If you have made an ADFS version of the Studio 5000 System Disc, you may use this instead, provided the the modules are in directory $.M, and directory $.C also exists.

including Toolbox modules

You will normally want to include also the Toolbox modules in sideways RAM. To do this, you must copy them from the Toolbox System Disc to the Studio 5000 System Disc copy you have prepared.

In the case of a 40-track Studio 5000 System Disc, there may be insufficient free disc space for the Toolbox modules, so you should first maximise the available space by removing all non-essential files, including the example files, from the copy you have prepared. To do this, enter:

*DESTROY *

and answer 'Y' to each question that appears.

If this still does not give sufficient space, there is no simple alternative but to include a smaller number of Toolbox modules.

To copy the Toolbox modules, if you are using standard DFS discs on a dual drive,

* insert your Toolbox System Disc in drive 0
* leave the copy of your Studio 5000 System Disc in drive 1
* enter

    COPY 01 M.

If you are using standard DFS discs on a single drive,

* have ready your Toolbox System Disc
    and the copy of your Studio 5000 System Disc
* enter

    COPY 00 M.

  and follow the instructions that appear on the screen.

carrying out the adaptation

To carry out the adaptation, have ready the Toolbox System Disc and the copy of your Studio 5000 System Disc, and

* insert the Toolbox System Disc into the disc drive (if there are two slots, into drive 0)
* tap the BREAK key while holding down SHIFT
* select the SideMod option from the menu

If you have already started from the Studio 5000 System Disc, you may alternatively:

* save your program if you want to keep it
* insert the Toolbox System Disc into the disc drive (if there are two slots, into drive 0)
* ensure you are in command mode or the Main Menu, and press f9
* select the SideMod option from the menu

Follow the instructions given on the screen. When you are asked to insert the System Disc you wish to be adapted, you should insert the copy made for this purpose. SideMod automatically senses the type of drive (dual or single) and number of tracks (40 or 80) and type of disc (DFS or ADFS).

After asking for your System Disc, SideMod will show you a list of all the modules on it. Those Studio 5000 modules that are normally loaded from disc during use of the system are already marked 'selected' for transferral to sideways RAM (those that are loaded once on start-up are not selected). At this point you can change the selection of any module, by moving to its name with up and down and pressing RETURN. The only change you are likely to want to make is to select any Toolbox modules you have added.

In special cases, you might want to unselect certain rarely-used modules to reduce the amount of sideways RAM used by SideMod. Any module not selected for transferral to sideways RAM is left on the disc, from where it will be loaded as normal when required.

Once you have confirmed the selection of modules, SideMod proceeds to transfer all selected module files into a single large sideways RAM image file. It also modifies the !BOOT file so that upon starting the System Disc, this sideways RAM image file is loaded and the system is directed to search sideways RAM for each module before looking for it on disc.

Once SideMod completes sucessfully, your adapted System Disc is ready for use.

testing

To test the adapted Studio 5000 System Disc, use it to start the system as normal. Don't worry that the list of start-up commands is slightly different, but watch out for any error messages - lines starting with '!'.

When the Main Menu appears, remove the System Disc from the drive and select each of the editors in turn - they should appear without attempting to access the disc drive.

If you included the Toolbox modules, test them in the same way, removing the Toolbox System Disc after its jukebox appears.

fault finding

If the computer's sideways RAM is not available, during adaptation of the System Disc or use of the adapted System Disc, an error message will appear.

'! Bad Address' means that one or more of the sideways RAM banks needed has been disabled by changing the settings of internal links 18 and/or 19. You must re-enable the sideways RAM - refer to the instructions you followed when you disabled the sideways RAM originally, or to the Master Reference Manual. '! RAM occupied' means that one or more of the sideways RAM banks needed is already occupied by a ROM image. The command *ROMS will display the name of the ROM image. You should turn off the computer and try again.

If you must disable some of the sideways RAM banks or store a ROM image in sideways RAM, you may do so provided the bank(s) required by the adapted System Disc are left free - SideMod tells you which these are at the end of the adaptation process.

The error message '! Not found' means that the disc has the wrong directory structure, most probably caused during copying onto ADFS. You should ensure that the modules are in directory M, directory C exists, and $.!BOOT exists. If you continue to have problems, make a copy of your original Studio 5000 System Disc, adapt it with SideMod, and only then carry out your copying to ADFS or other modification.

The error message '! Disc full' suggests that you are using a 40-track Studio 5000 System Disc with extra files on it. You should try again with a copy of your System Disc from which all non-essential files, including the example files, have been

removed by using the command:

*DESTROY *

and answering 'Y' to each question that appears.

If the adapted System Disc fails to work, or gives unpredictable errors from OS commands for example, your computer probably has Econet installed and enabled. You can check this by entering the command *ROMS - if the display shows 'ANFS' opposite ROM 8, the Econet is active, at least in part. You should disable the Econet as follows:

* enter:

*UNPLUG 8

* while holding down CTRL, tap BREAK
* start up your System Disc as before

After doing this, the Econet remains disabled indefinitely, even if you turn the computer off and on again. If you want to restore the Econet for another application, you should:

* enter:

*INSERT 8

* while holding down CTRL, tap BREAK

using the adapated System Disc

The adapted System Disc's use of sideways RAM is entirely automatic, so there are no special instructions you need to know to use the adapted disc (unless you need to switch filing systems - see below).

Once you have started the system from the adapted System Disc and the Main Menu has appeared, you may remove the System Disc, replacing it only when you need to load an example file from it.

If when you adapted the disc you included the Toolbox modules in the selection, you will now be able to load them using MLOAD without having to insert the Toolbox System Disc. If you still prefer to use the Toolbox jukebox, you can load it from the Toolbox System Disc as before, or more conveniently, transfer it to your own working discs.

If you unselected any of the modules that SideMod suggested, these

and any others not included in sideways RAM will be loaded from disc when required as normal. You are still free to change the MPREFIX string.

selecting filing systems

If you select a filing system with, for example, the *DISC or *ADFS commands, loading from sideways RAM will be disengaged. To re-engage it you should enter the command:

```
*/C.RCode
```

The file 'C.RCode' is present on the adapted DFS System Disc, so if you are in the DFS, you should make sure the System Disc is in the current drive. If you want to be able to re-engage sideways RAM while in another filing system, you should copy this file to the new filing system. You can then re-engage sideways RAM fro the new filing system by entering the command as normal.

8 AREC program recoverer

AREC is a utility that recovers AMPLE programs from DFS or ADFS floppy discs. Since it does not rely on any catalogue information, it can recover programs that have been deleted or lost due to corruption of the disc catalogue.

starting AREC

To use AREC, have ready the disc containing the lost programs (the source disc). If you want to keep any recovered programs, also have ready a disc to receive them (the destination disc). This should be of the same type (DFS or ADFS) as the source disc. If you are using a double-sided drive and DFS, you may alternatively use the other side of the first disc as the destination.

If you just want to examine the source disc and not keep any recovered programs, you will not need a destination disc.

Now,

* insert the Toolbox System Disc into the disc drive (if there are two slots, into drive 0)
* tap the BREAK key while holding down SHIFT
* select the AREC option from the menu

If you have already started from the Studio 5000 System Disc, you may alternatively:

* save your program if you want to keep it
* insert the Toolbox System Disc into the disc drive (if there are two slots, into drive 0)
* ensure you are in command mode or the Main Menu, and press f9
* select the AREC option from the menu

using AREC

AREC asks you a number of questions before starting work. Each one shows a bracketed list of allowed answers, which may be different for DFS and ADFS. You may either type in one of these answers and press RETURN, or just press RETURN to use the first answer in the list. The questions are as follows:

DFS, ADFS or quit (D/A/Q):

Select the filing system appropriate to the source and destination discs.

DFS: **Disc size (no. of tracks; 40/80):**
ADFS: **Disc size (S/M/L/40/80/160):**

Enter the number of tracks on the source disc. If you enter 40 (or press RETURN), only the first 40 tracks will be searched, regardless of the number actually on the disc.

DFS: **Source drive no. (0/1/2/3):**
ADFS: **Source drive no. (0/1/4/5):**

Enter the number of the drive containing the source disc.

DFS: **Destination drive no. (1/0/2/3/X):**
ADFS: **Destination drive no. (1/0/4/5/X):**

Enter the number of the drive containing the destination disc, or if you don't want to keep any recovered programs, enter X. If you're using a single drive and separate source and destination discs, enter the same drive number as you did for the source - AREC will then ask you to swap discs when required. If you enter a different drive number, AREC will not give you a chance to swap discs.

DFS and ADFS: **Destination directory ($/name):**

Enter the name of the directory you want the recovered programs to be stored under. Remember that on the DFS, only a single character is allowed, and under the ADFS, the directory must already exist. If you answered X to the previous question, this question is not asked.

AREC now searches the disc, displaying the number of each sector it searches and the start sector number and length of any AMPLE program it finds. Unless you entered X as the destination drive, when a program is found in a recoverable state, you will see a message telling you that it is being saved (if necessary AREC will ask you to swap discs). Each program is saved on the destination disc under the destination directory with a name consisting of 'rec' followed by its start sector number in hex, for example, rec002, or rec01A.

If AREC finds the start of a program but then before the end of the program finds an unreadable sector or the start of another program, it aborts recovery of the first program since anything but a complete AMPLE program is not usable.

using recovered programs

In almost all cases, the fact that a program has been recovered is an indication of complete success - the recovered program is identical to the original, except for its name, and you can load and use it in the normal way. Very occasionally, AREC may find and recover something that looks like an AMPLE program but is not, so when you try to load it, you will get the '! Bad program' message.

9 Summary of key controls

This chapter gives a summary of the key controls of TEDIT, IEDIT and UTILS BROWSE, for quick-reference use.

TEDIT text editor edit mode

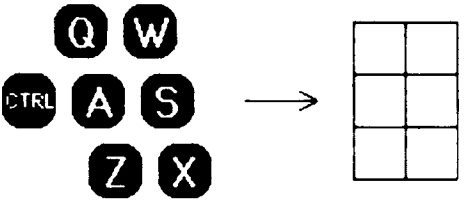
TAB	go to command mode
left/right/up/down	move by one character
SHIFT up/down	move up/down by half a screenfull
SHIFT left/right	move left/right by one word
CTRL up/down	move to start/end of text
CTRL left/right	move to start/end of line
RETURN	split line
DELETE	delete back/join line to previous
COPY	delete forward/join line to next
SHIFT COPY	delete end of line
CTRL COPY	delete whole line
CTRL SHIFT COPY	start character copying
COPY	copy character
RETURN	end character copying

IEDIT image editor

TAB	go to command mode
left/right/up/down	move by one character
SHIFT up/down/left/right	copy character ('paint')
CTRL up/down	move to top/bottom of screen
CTRL left/right	move to start/end of line
RETURN	move to start of next line
DELETE	delete back, replacing by space
COPY	set marker
SHIFT COPY	copy block
f0	call-up menu (code help info get put swap)
f1	next alpha (text) colour
f2	next graphics colour
f3	insert new background
f4	insert black background
f5	enter graphics block
f6	insert line
f7	delete line
f8	insert character
f9	delete character forward, closing up

IEDIT image editor

TAB	go to command mode
left/right/up/down	move by one character
SHIFT up/down/left/right	copy character ('paint')
CTRL up/down	move to top/bottom of screen
CTRL left/right	move to start/end of line
RETURN	move to start of next line
DELETE	delete back, replacing by space
COPY	set marker
SHIFT COPY	copy block
f0	call-up menu (code help info get put swap)
f1	next alpha (text) colour
f2	next graphics colour
f3	insert new background
f4	insert black background
f5	enter graphics block
f6	insert line
f7	delete line
f8	insert character
f9	delete character forward, closing up
SHIFT f1	toggle flashing/steady
SHIFT f2	toggle double/normal height
SHIFT f3	toggle separated/contiguous
SHIFT f4	toggle hold/release graphics
SHIFT f5	insert conceal display
SHIFT f6	insert column
SHIFT f7	delete column forward, closing up



toggle graphics pixel

UTILS BROWSE

TAB	return to % prompt
up/down	move up/down the list
right	enter the current word
left	exit the current word
RETURN	display the definition of the current word
RETURN	in 40-column modes, restore structure display

Index

S

T

U

W